ECE661 Fall 2024: Homework 4

Che-Hung Huang, 00299-50990

huan1160@purdue.edu

1 Theory Question

The LoG of an image can be computed as the DoG since

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

where

$$ff(x, y, \sigma) = \iint f(x', y')\, g(x - x', y - y')\, dx'dy' \quad \text{and} \quad g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

To see that equation (1) holds, we write $h(x, y) = \nabla^2 g(x, y) = \frac{-1}{2\pi\sigma^4}\left(2 - \frac{x^2+y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$, and so

$$\sigma \nabla^2 ff(x, y, \sigma) = \sigma \iint f(x', y')\, h(x - x', y - y')\, dx'dy'.$$

Next, we also have

$$\begin{aligned}\frac{\partial}{\partial \sigma} ff(x, y, \sigma) &= \iint f(x', y')\, \frac{\partial}{\partial \sigma} g(x - x', y - y')\, dx'dy' \\ &= \iint f(x', y')\, h(x - x', y - y')\, dx'dy',\end{aligned}$$

where the second equation follows from a direct computation. Thus we see that equation (1) holds.

Computing DoG is more efficient than directly computing LoG since the 2D convolution in LoG is not separable, but the 2D convolution in DoG is separable, and thus DoG can be calculated using two 1D convolutions.

2 Programming Tasks

2.1 Harris Corner Detector

To implement the Harris corner detector, we start with the Haar wavelet filters H_x and H_y of size M. Here we choose M to be the smallest even number greater than the scale σ. In NumPy, H_x is given by `np.hstack([-np.ones((M, M // 2)), np.ones((M, M // 2))])`, and H_y is the transpose of H_x. Next, we apply the Haar filters H_x and H_y to our image and denote the results by d_x and d_y, respectively.

To determine whether a pixel is a corner, we consider a $5\sigma \times 5\sigma$ neighborhood B of the pixel and calculate the following matrix:

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

Next we threshold the value

$$\frac{\det(C)}{[\mathrm{Tr}(C)]^2}$$

to determine whether the pixel is considered as a corner. To detect a strong corner, we can threshold the Harris corner response

$$R = \det(C) - k \cdot [\mathrm{Tr}(C)]^2,$$

where k is an empirical constant. Finally, we classify a pixel as a corner if it is a local maximum of the function R. In this way, we can rule out false positive cases and weak corners from the image.

If we have a pair of images that capture the different views of the same scene, we can match the corresponding interest points using SSD (Sum of Square Differences) and NCC (Normalized Cross Correlation):

$$SSD = \sum_i \sum_j |f_1(i,j) - f_2(i,j)|^2 \quad \text{and} \quad NCC = \frac{\sum\sum(f_1(i,j)-m_1)(f_2(i,j)-m_2)}{\sqrt{\sum\sum(f_1(i,j)-m_1)^2 \sum\sum(f_2(i,j)-m_2)^2}},$$

where m_1 and m_2 are the means of $f_1(i,j)$ and $f_2(i,j)$, respectively, and the summations are taken over a window around corners being considered. For every corner in the first image, its corresponding corner in the second image is the one that either minimizes the SSD or maximizes the NCC. Thus we can use two for-loops to match all corners in the pair of images.

We apply the above procedures to the provided images:

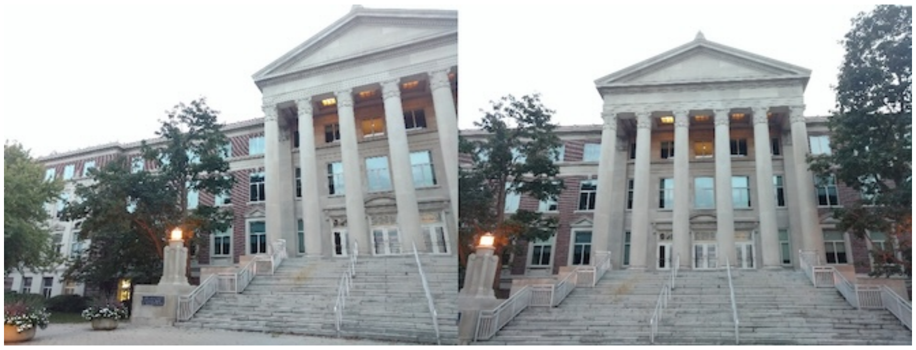
Figure 1: Provided image pair 1

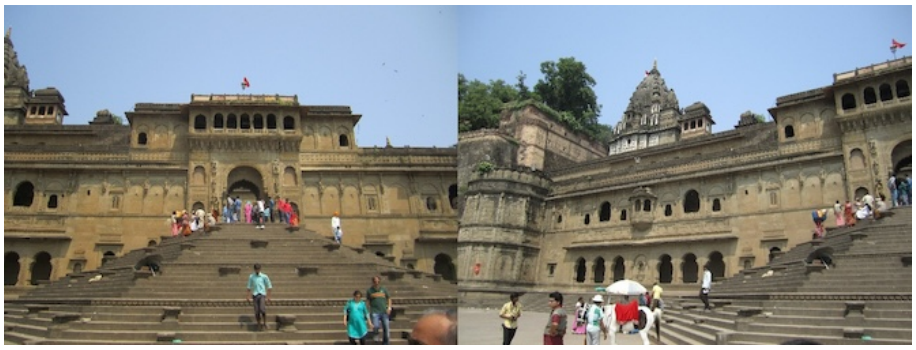
Figure 2: Provided image pair 2

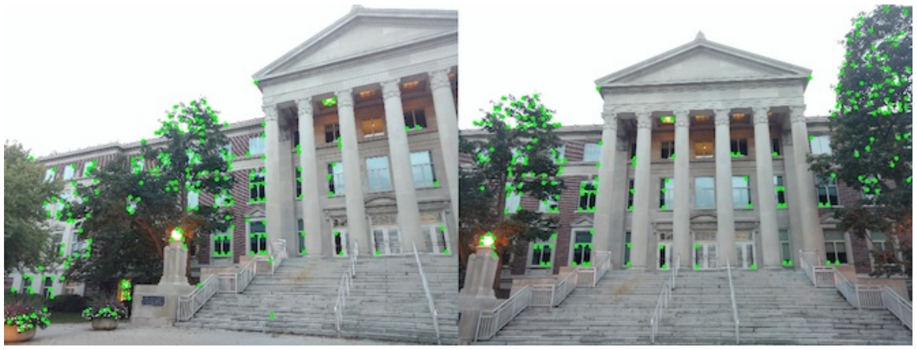
Figure 3: Corners in image pair 1 (with $\sigma = 1$)

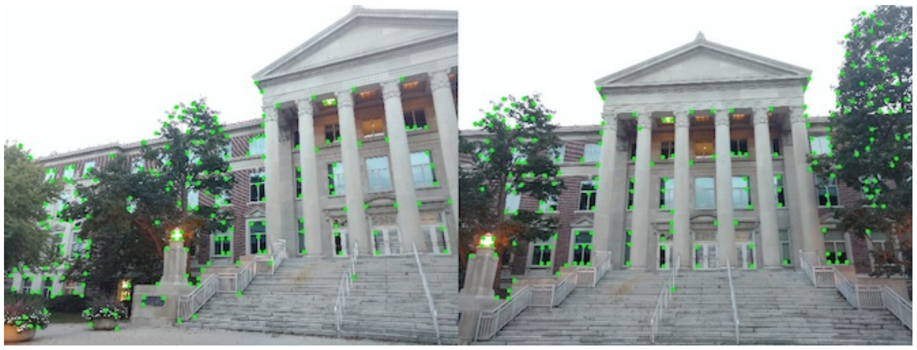
Figure 4: Corners in image pair 1 (with $\sigma = 1.2$)

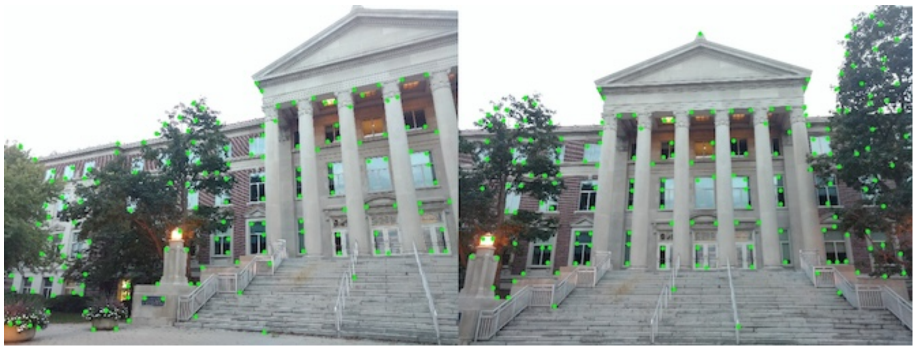
Figure 5: Corners in image pair 1 (with $\sigma = 1.6$)

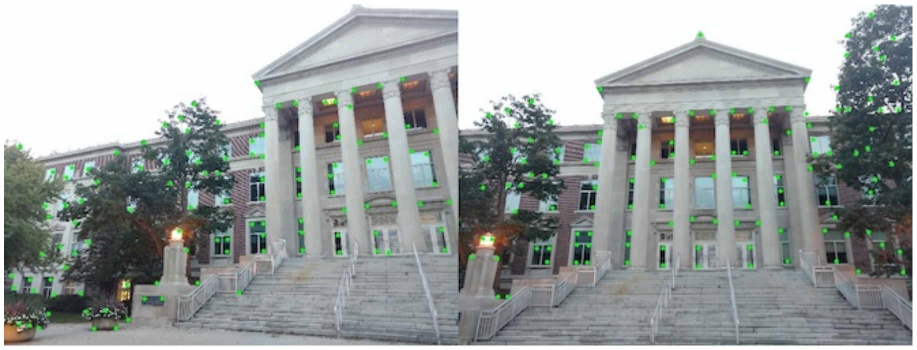
Figure 6: Corners in image pair 1 (with $\sigma = 2$)

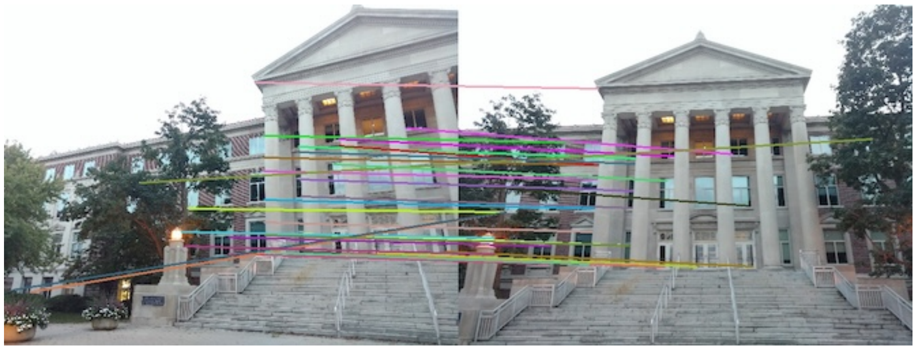
Figure 7: Matching in image pair 1 (using SSD and $\sigma = 1$)

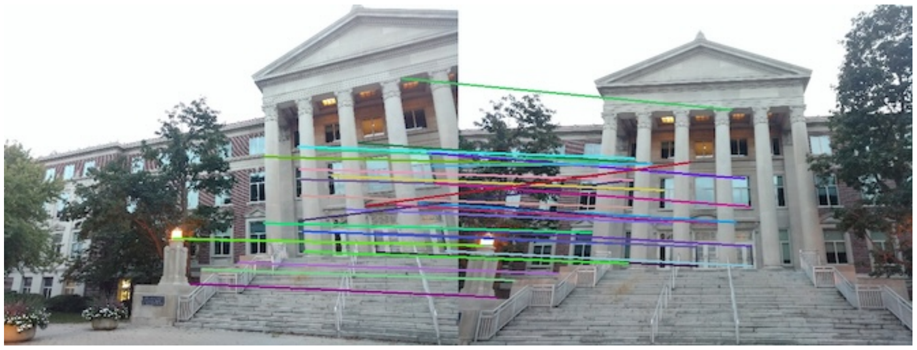
Figure 8: Matching in image pair 1 (using SSD and $\sigma = 1.2$)

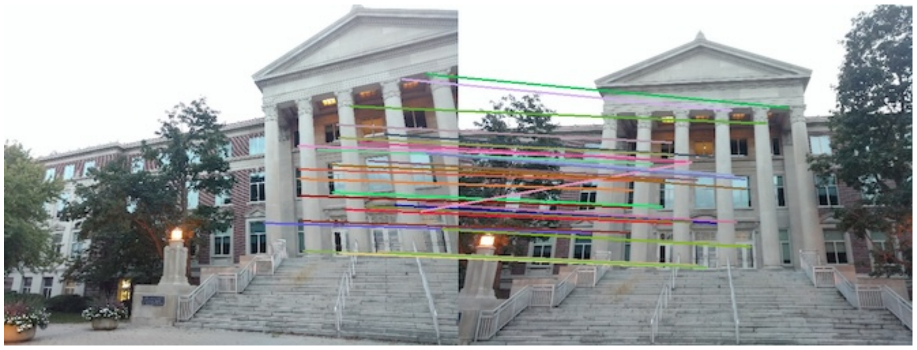
Figure 9: Matching in image pair 1 (using SSD and $\sigma = 1.6$)

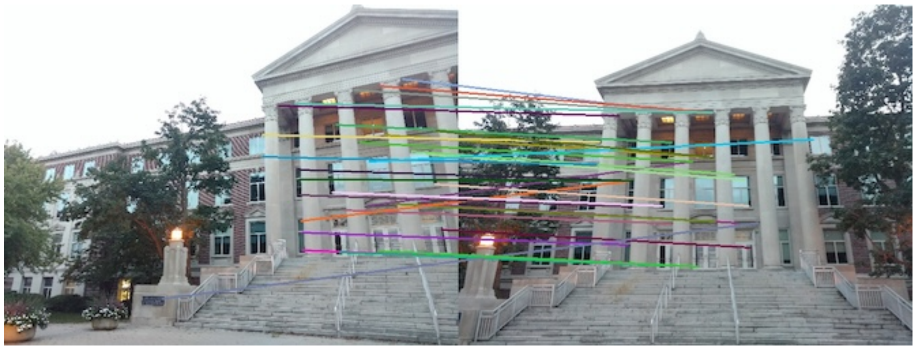
Figure 10: Matching in image pair 1 (using SSD and $\sigma = 2$)

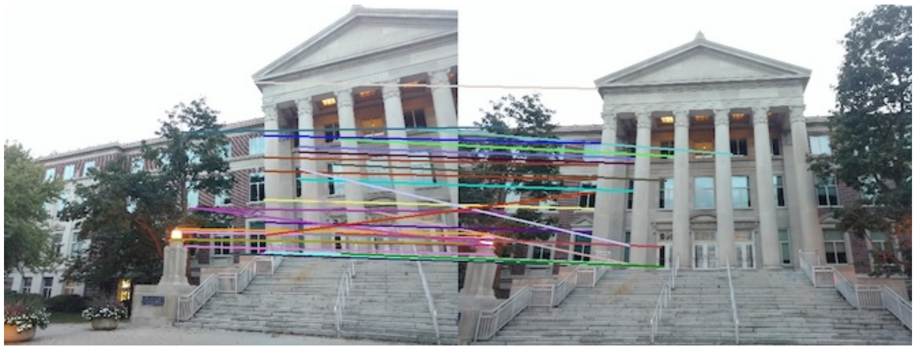
Figure 11: Matching in image pair 1 (using NCC and $\sigma = 1$)

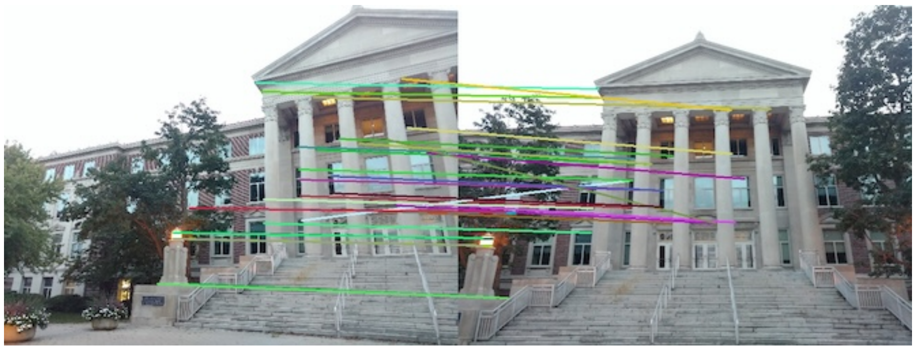
Figure 12: Matching in image pair 1 (using NCC and $\sigma = 1.2$)

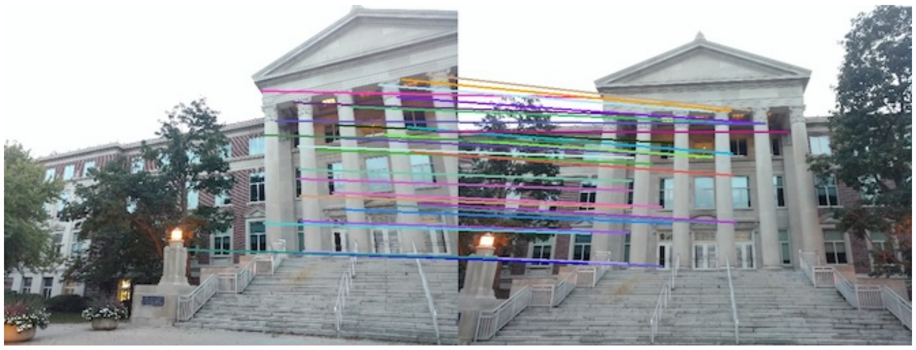
Figure 13: Matching in image pair 2 (using NCC and $\sigma = 1.6$)

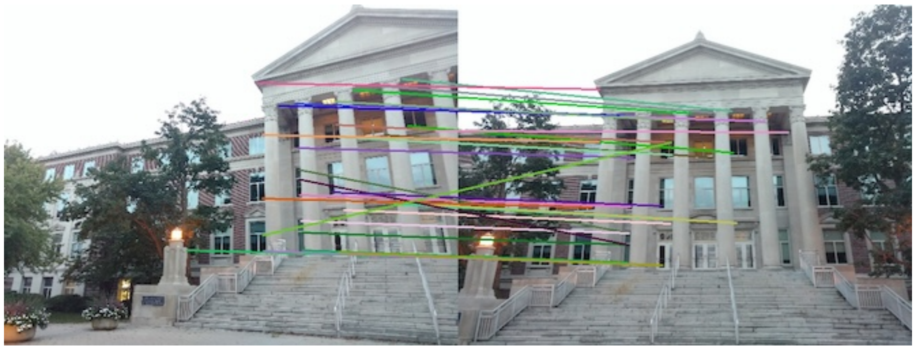
Figure 14: Matching in image pair 2 (using NCC and $\sigma = 2$)

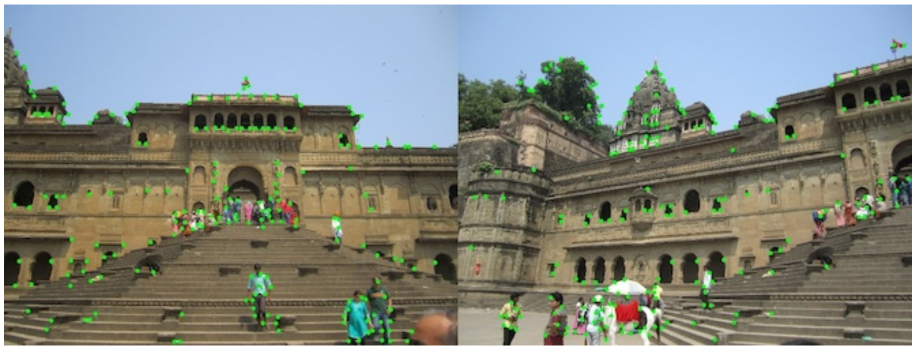
Figure 15: Corners in image pair 2 (with $\sigma = 1$)

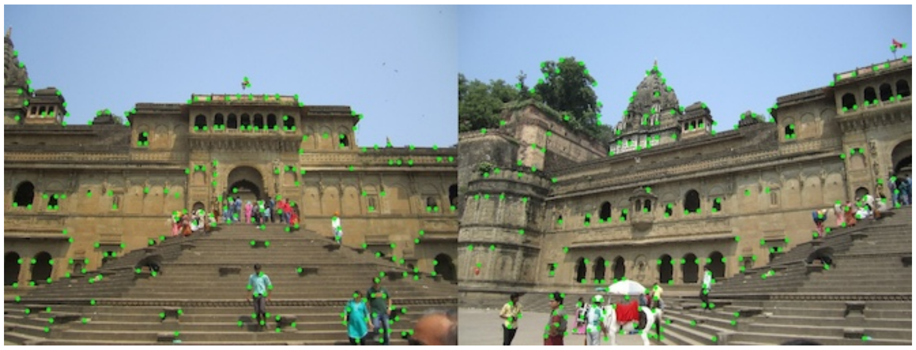
Figure 16: Corners in image pair 2 (with $\sigma = 1.2$)

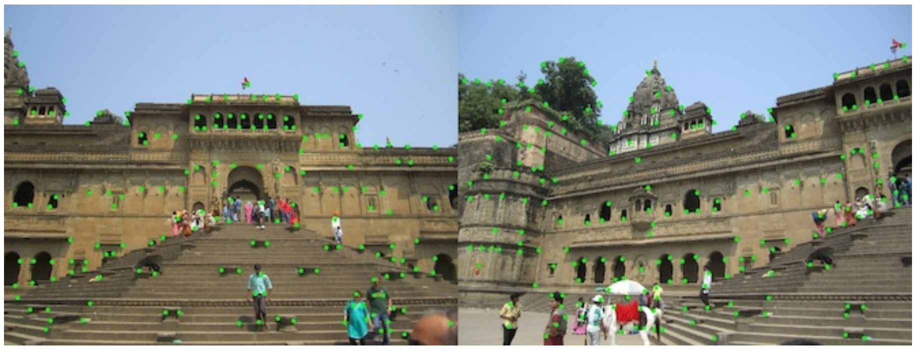
Figure 17: Corners in image pair 2 (with $\sigma = 1.6$)

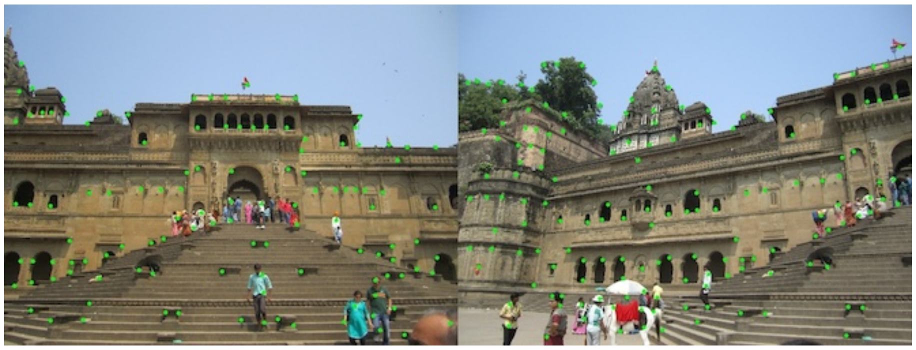
Figure 18: Corners in image pair 2 (with $\sigma = 2$)

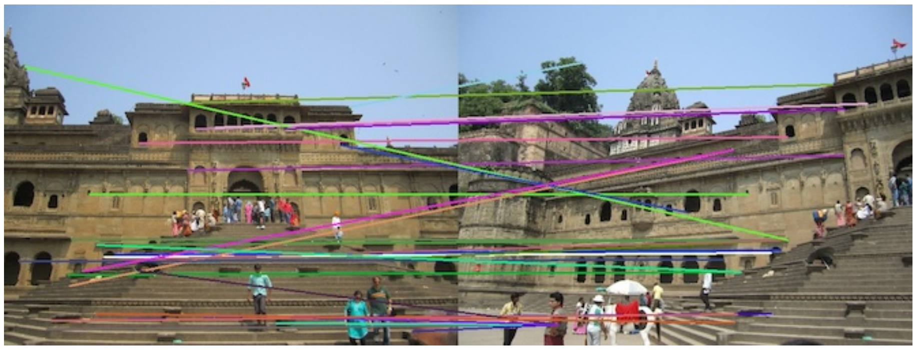
Figure 19: Matching in image pair 2 (using SSD and $\sigma = 1$)

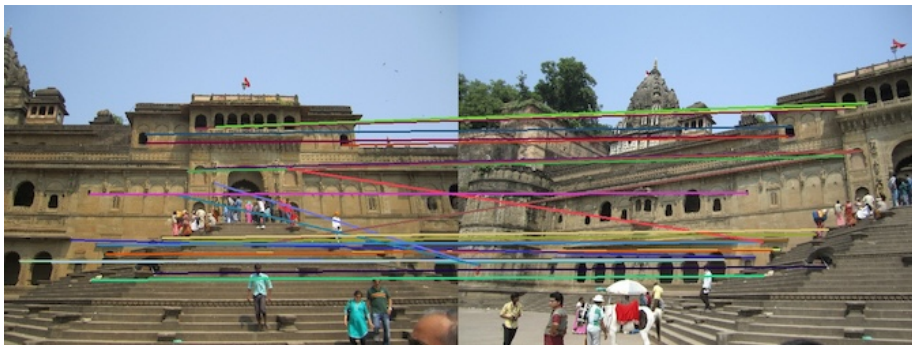
Figure 20: Matching in image pair 2 (using SSD and $\sigma = 1.2$)

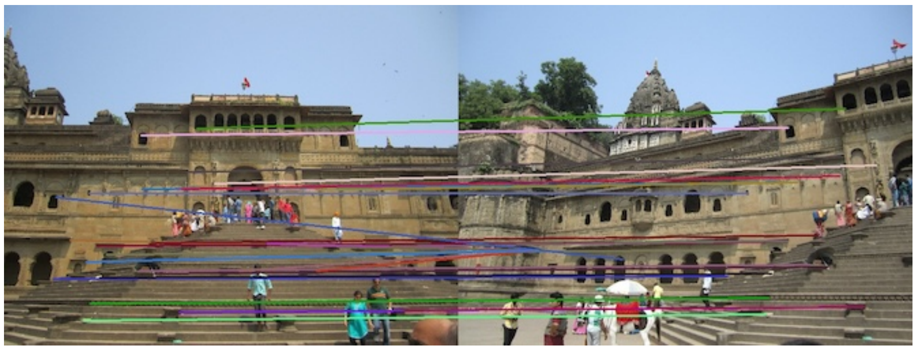
Figure 21: Matching in image pair 2 (using SSD and $\sigma = 1.6$)

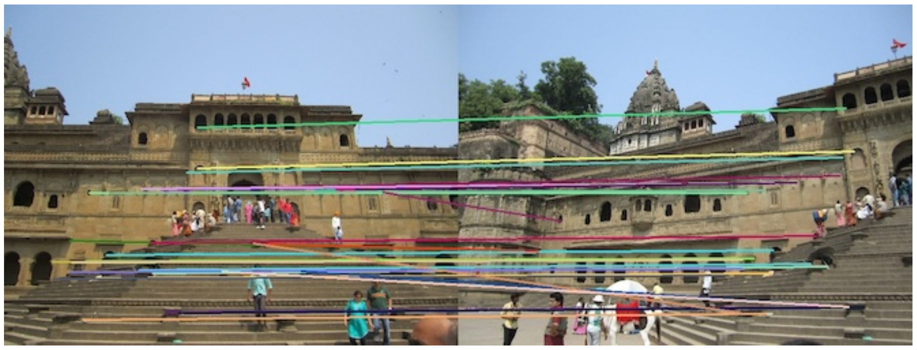
Figure 22: Matching in image pair 2 (using SSD and $\sigma = 2$)

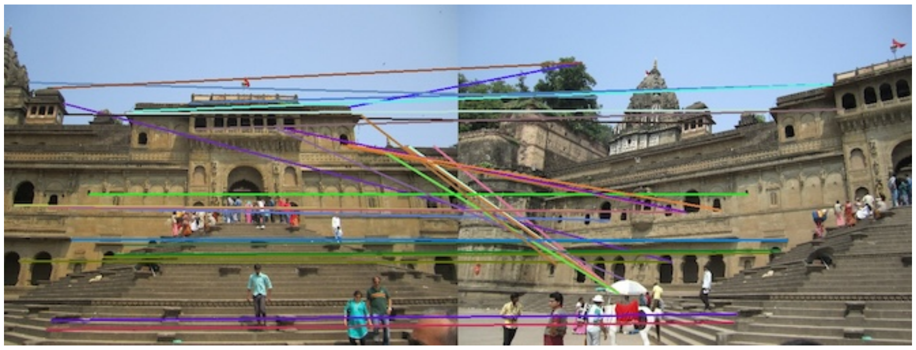
Figure 23: Matching in image pair 2 (using NCC and $\sigma = 1$)

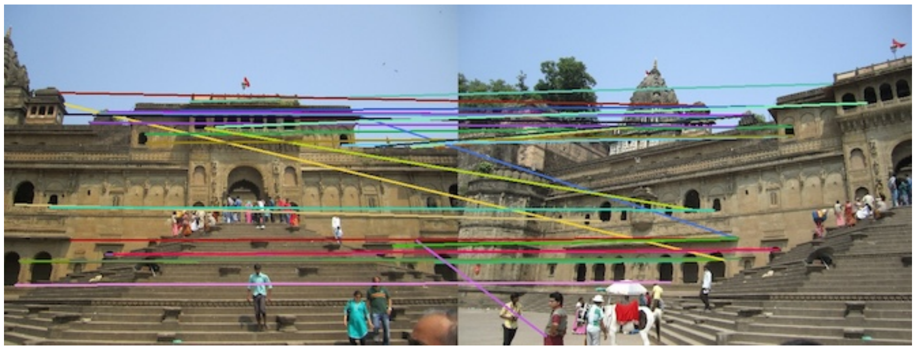
Figure 24: Matching in image pair 2 (using NCC and $\sigma = 1.2$)

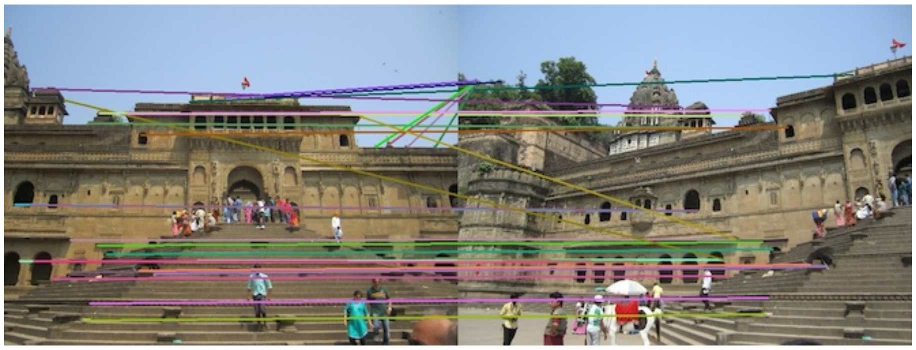
Figure 25: Matching in image pair 2 (using NCC and $\sigma = 1.6$)

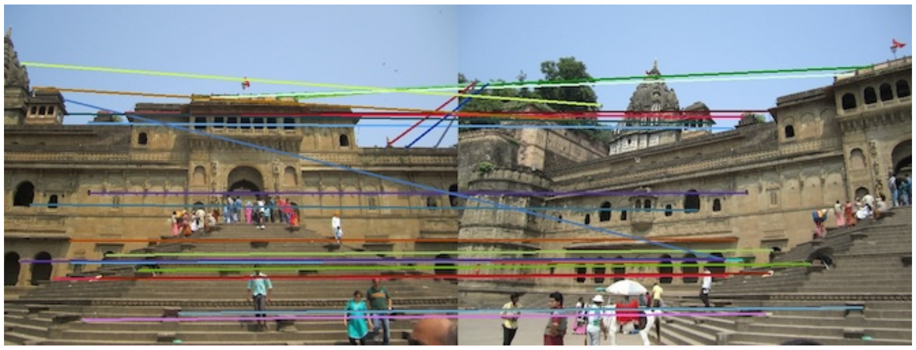
Figure 26: Matching in image pair 2 (using NCC and $\sigma = 2$)

Observations: We observe that larger values of σ correspond to larger features, and they provide more robustness to noise. On the other hand, smaller values of σ allow us to detect more sophisticated features, and the resulting number of interest points will be larger. We can see in the previous images that SSD achieves better performance compared to NCC across different scales. Also, setting $\sigma = 2$ provides better result for both SSD and NNC compared to setting $\sigma = 1$. As we mentioned, this is due to the property that larger scale values σ are more robust to noise than smaller scale values.

2.2 SIFT

SIFT (Scale Invariant Feature Transform) consists of several steps: The first step is to find the local extrema of the DoG $D(x, y, \sigma)$. This is done by comparing each pixel value $D(x, y, \sigma)$ with the 26 pixel values in its 3D neighborhood. Next, we can locate these extrema with sub-pixel accuracy: if we let $J(\boldsymbol{x}_0)$ and $H(\boldsymbol{x}_0)$ be the gradient and Hessian of the function $D(x, y, \sigma)$ at $\boldsymbol{x}_0 = (x_0, y_0, \sigma_0)^T$, then

$$\boldsymbol{x} = -H^{-1}(\boldsymbol{x}_0)J(\boldsymbol{x}_0)$$

gives the precise location of the local extremum. After obtaining the locations of local extrema, we apply a threshold to the magnitude $|D(x, y, \sigma)|$ of DoG to rule out weak local extrema.

The next step is to find the dominant local orientation for each local extremum of $D(x, y, \sigma)$. This is done by: (1) computing the gradient magnitude $m(x, y)$ and gradient orientation $\theta(x, y)$ of the σ-smoothed image ff(x, y, σ) (2) weighting $\theta(x, y)$ with $m(x, y)$ (3) finding the peak in the 36-bin histogram of $\theta(x, y)$ in the range of 360°. After obtaining the local orientations, our last step is to create a SIFT descriptor for each local extremum. First, we consider the values of $\theta(x, y)$ relative to the dominant local orientation. Then we divide the 16×16 neighborhood of the local extremum into 4×4 cells, thus obtaining 16 cells. For each cell we calculate an 8-bin histogram of $\theta(x, y)$ (weighted with $m(x, y)$). Finally, combining the sixteen 8-bin histograms gives us a 128-element descriptor for every local extremum.

We apply the OpenCV implementation of SIFT to the provided images. We observe that SIFT achieves higher accuracy in matching interest points compared to Harris corner detector with SSD/NNC. Thus matching interest points using descriptors is more effective than directly comparing neighborhoods of the interest points. This is reasonable as descriptors capture more sophisticated features associated to interest points using the gradient information.

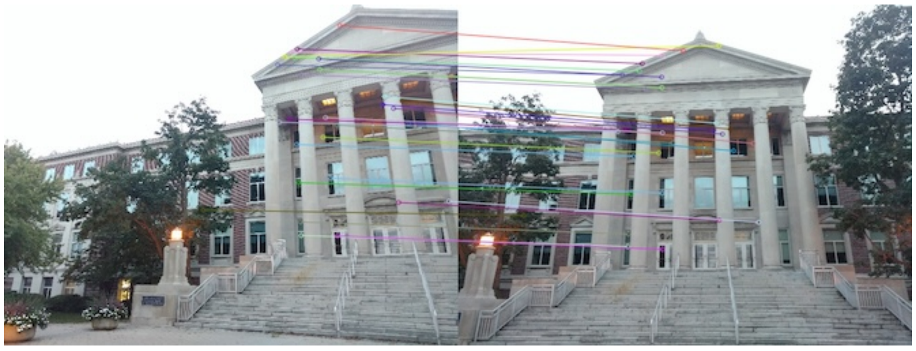

Figure 27: Matching in image pair 1 using SIFT

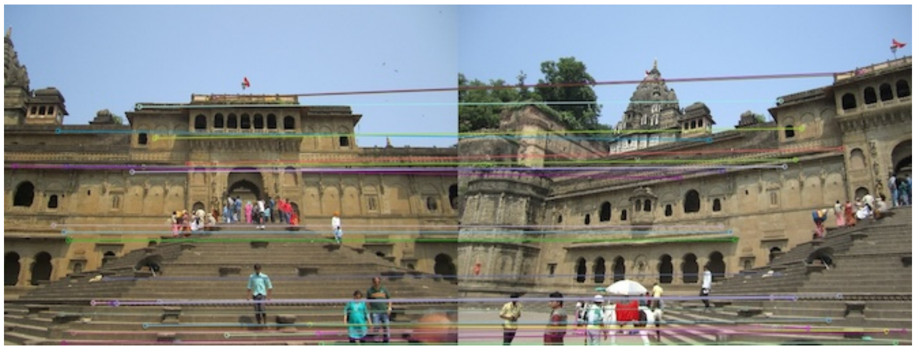

Figure 28: Matching in image pair 2 using SIFT

2.3 SuperPoint and SuperGlue

We apply the deep learning based approach for interest point detection and matching; this approach is based on SuperPoint and SuperGlue. Similar to SIFT, this approach achieves high accuracy in interest points matching in different views. Furthermore, we observe that the deep learning based approach detects more interest points and matches them with much higher accuracy. Thus this approach provides an over-all better result compared to SIFT and the Harris corner detector.

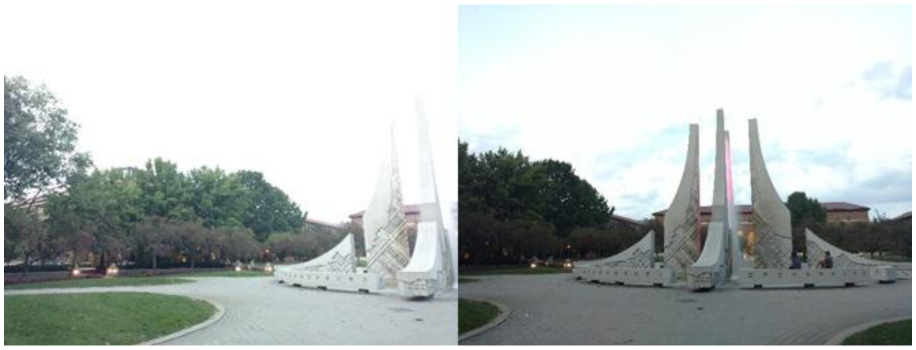

Figure 29: Provided image pair

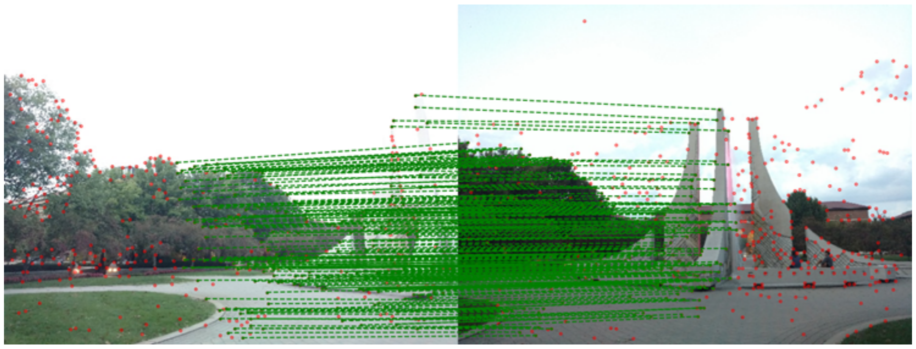

Figure 30: Matching in the provided image pairs

3 Custom Images

3.1 Harris Corner Detector

We re-run the previous processes on custom images:

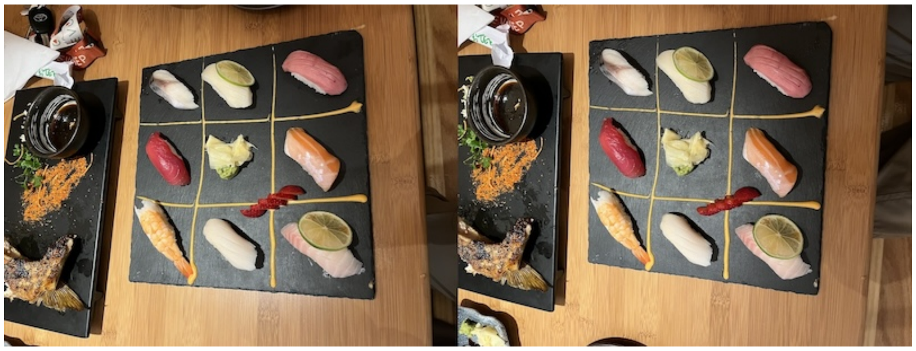

Figure 31: Custom image pair 1

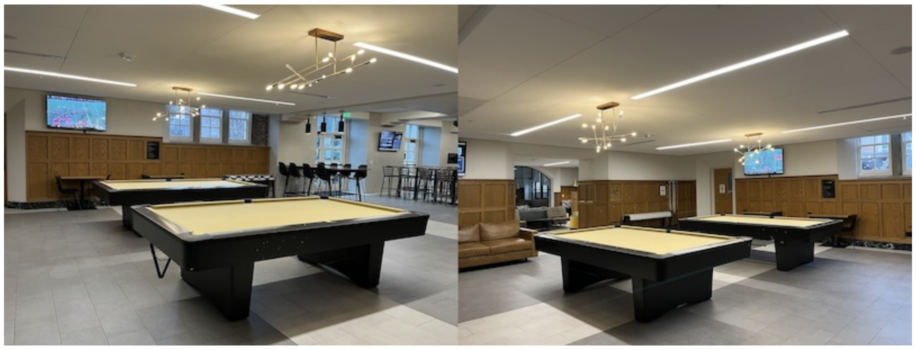

Figure 32: Custom image pair 2

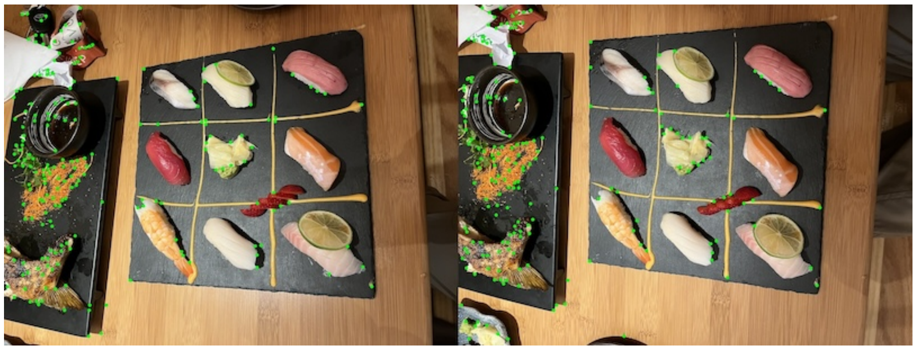
Figure 33: Corners in image pair 1 (with $\sigma = 1$)

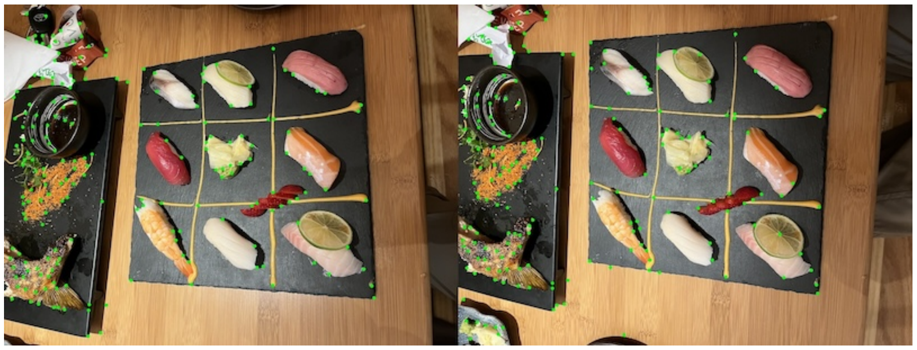
Figure 34: Corners in image pair 1 (with $\sigma = 1.2$)

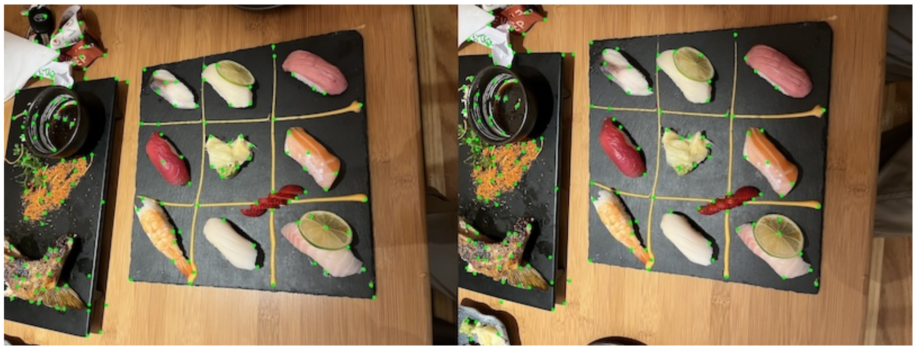
Figure 35: Corners in image pair 1 (with $\sigma = 1.6$)

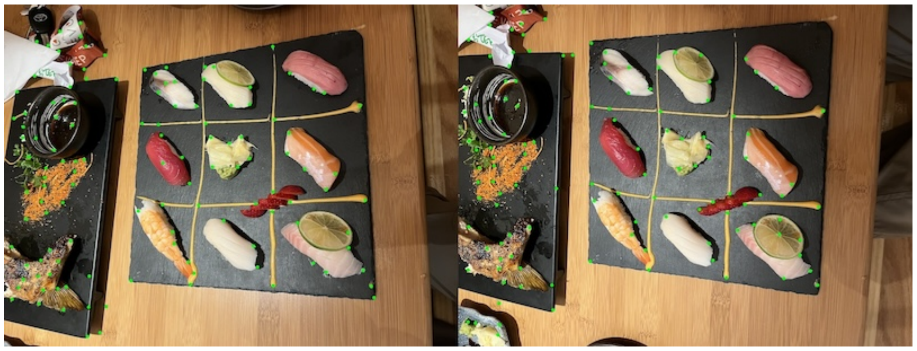
Figure 36: Corners in image pair 1 (with $\sigma = 2$)

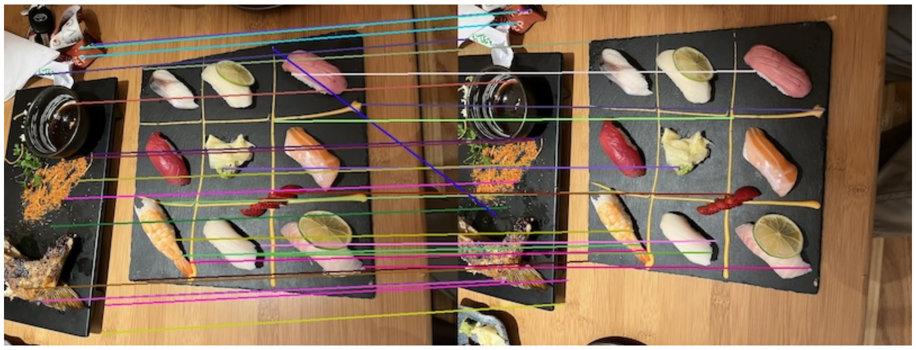
Figure 37: Matching in image pair 1 (using SSD and $\sigma = 1$)

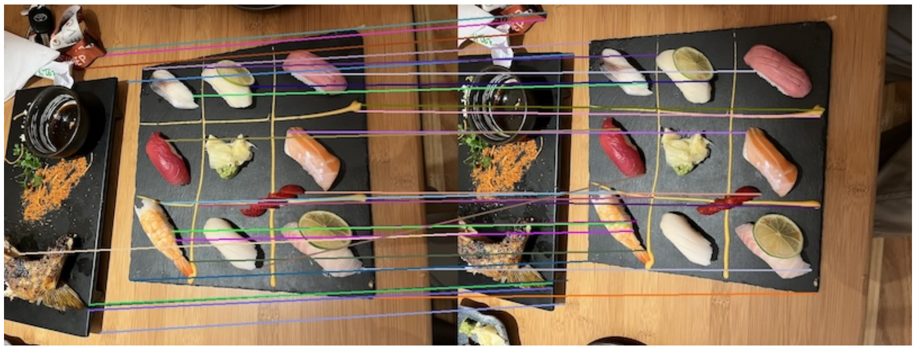
Figure 38: Matching in image pair 1 (using SSD and $\sigma = 1.2$)

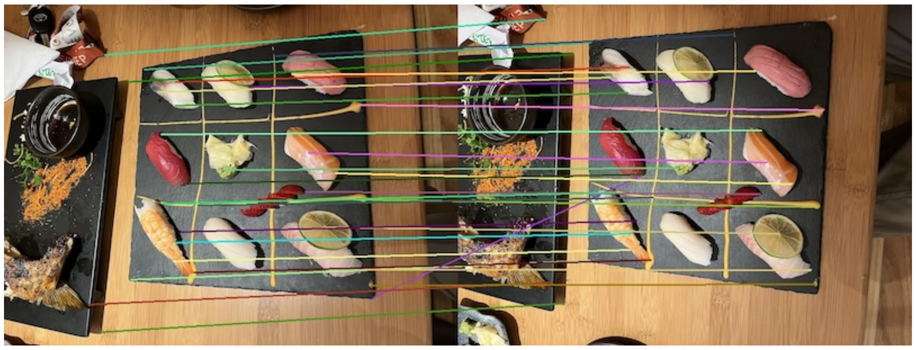
Figure 39: Matching in image pair 1 (using SSD and $\sigma = 1.6$)

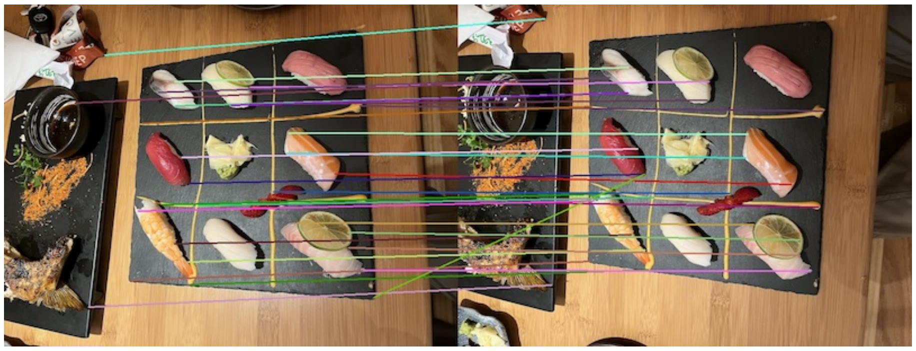
Figure 40: Matching in image pair 1 (using SSD and $\sigma = 2$)

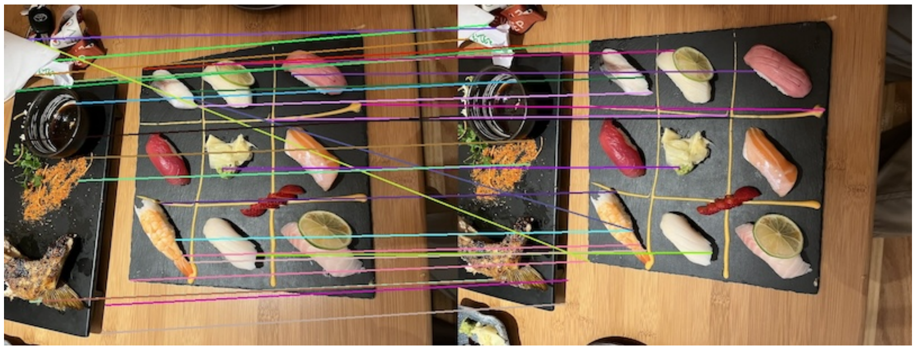
Figure 41: Matching in image pair 1 (using NCC and $\sigma = 1$)

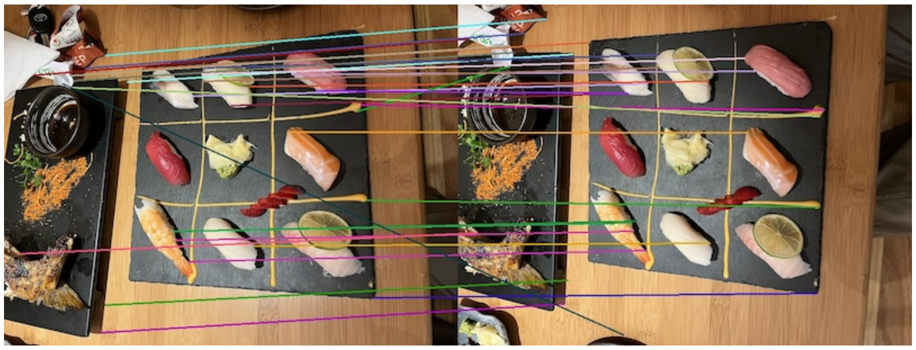
Figure 42: Matching in image pair 1 (using NCC and $\sigma = 1.2$)

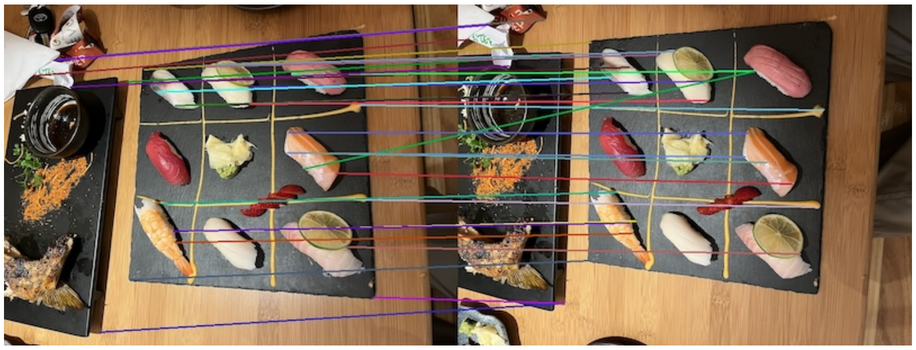
Figure 43: Matching in image pair 2 (using NCC and $\sigma = 1.6$)

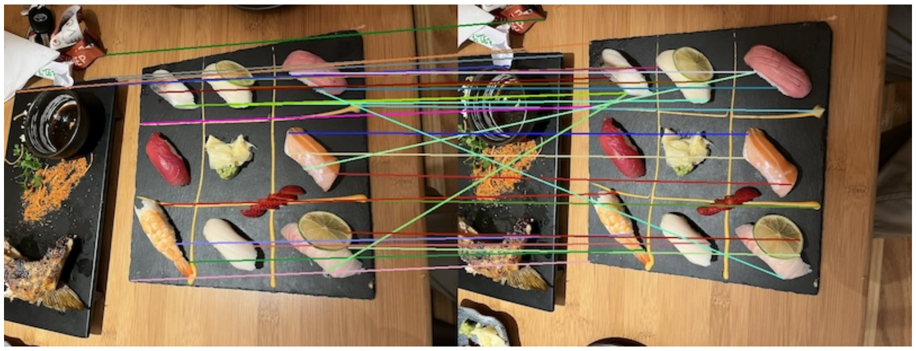
Figure 44: Matching in image pair 2 (using NCC and $\sigma = 2$)

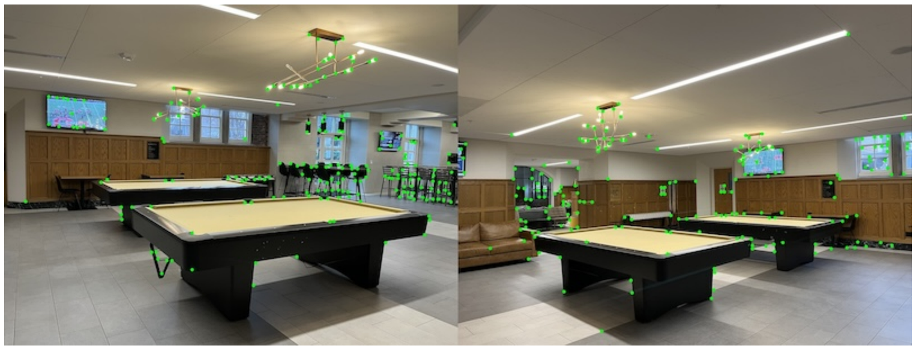
Figure 45: Corners in image pair 2 (with $\sigma = 1$)

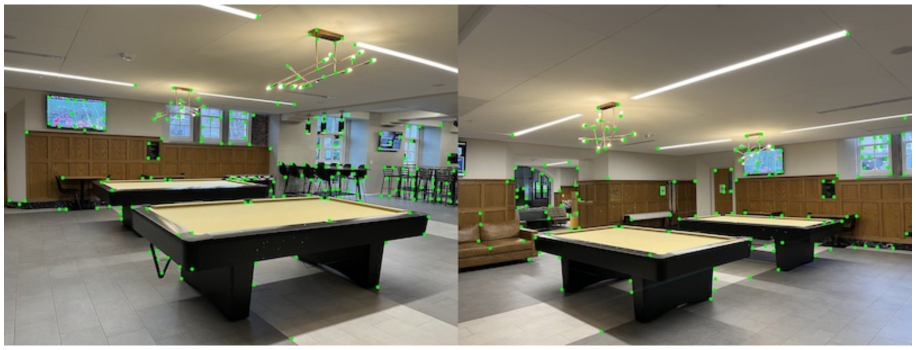
Figure 46: Corners in image pair 2 (with $\sigma = 1.2$)

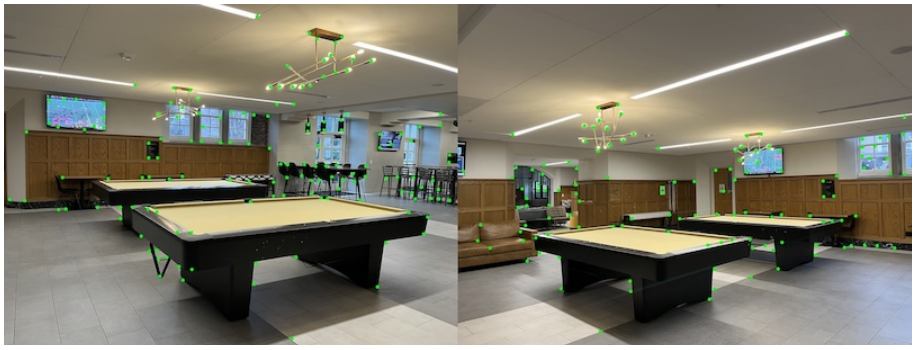
Figure 47: Corners in image pair 2 (with $\sigma = 1.6$)

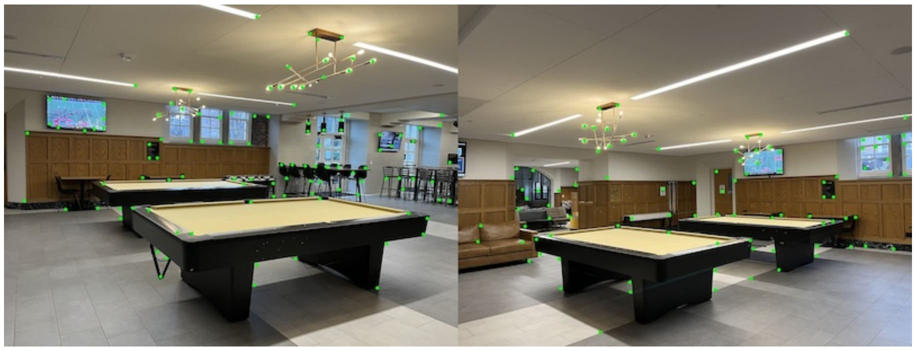
Figure 48: Corners in image pair 2 (with $\sigma = 2$)

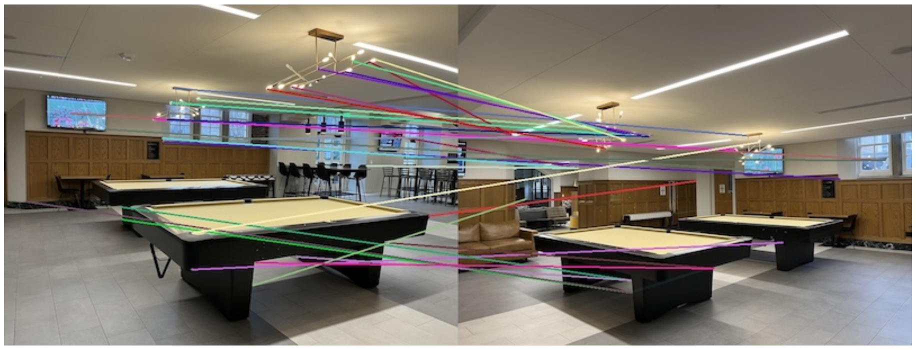
Figure 49: Matching in image pair 2 (using SSD and $\sigma = 1$)

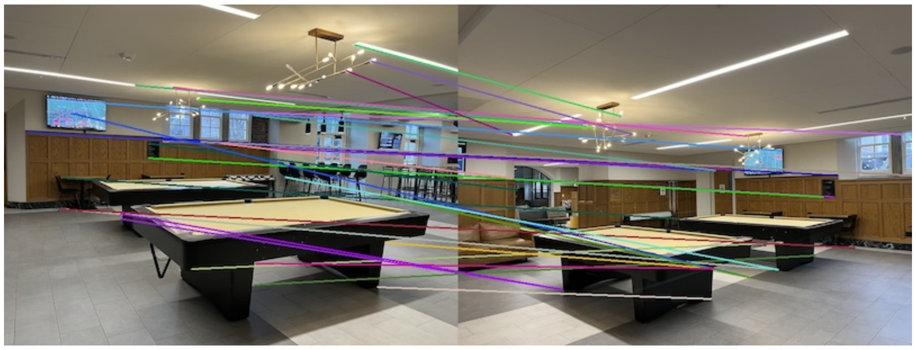
Figure 50: Matching in image pair 2 (using SSD and $\sigma = 1.2$)

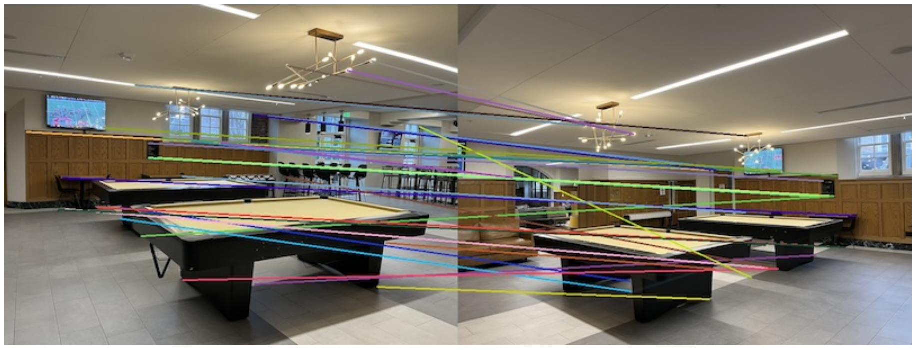
Figure 51: Matching in image pair 2 (using SSD and $\sigma = 1.6$)

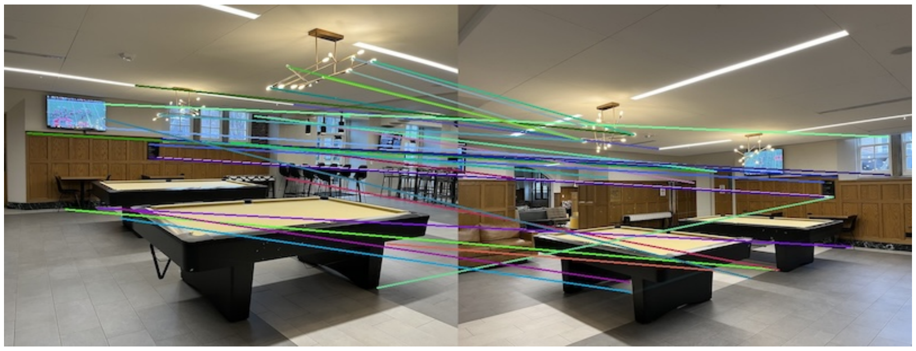
Figure 52: Matching in image pair 2 (using SSD and $\sigma = 2$)

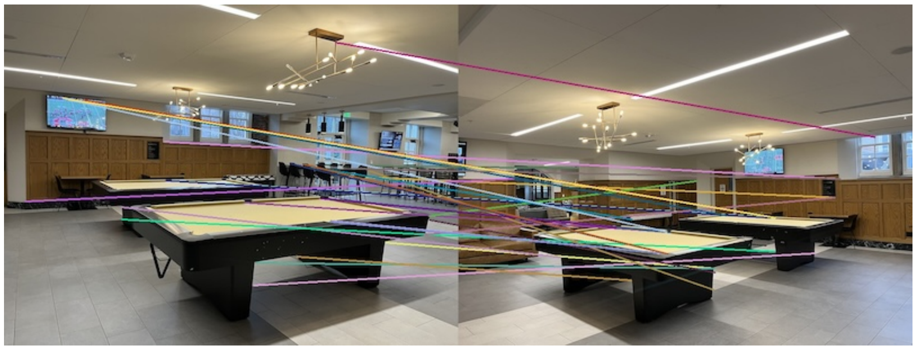
Figure 53: Matching in image pair 2 (using NCC and $\sigma = 1$)

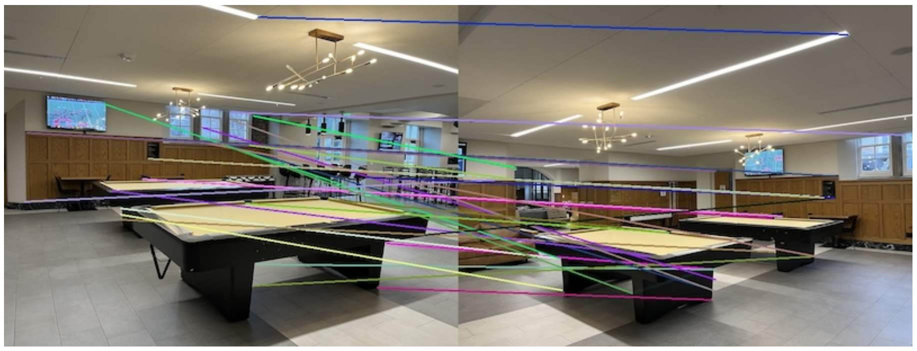
Figure 54: Matching in image pair 2 (using NCC and $\sigma = 1.2$)

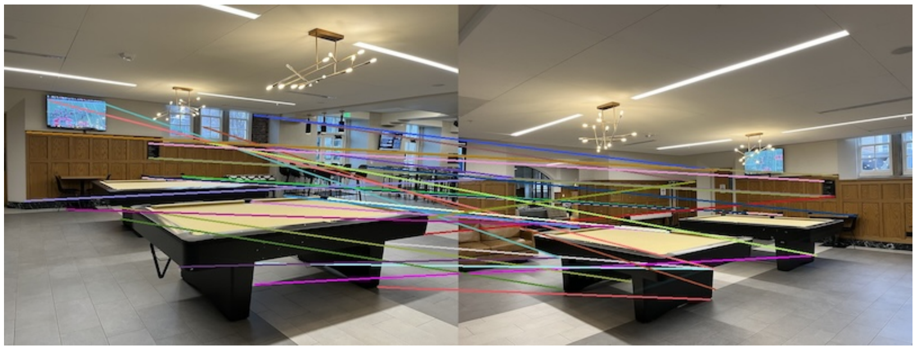
Figure 55: Matching in image pair 2 (using NCC and $\sigma = 1.6$)

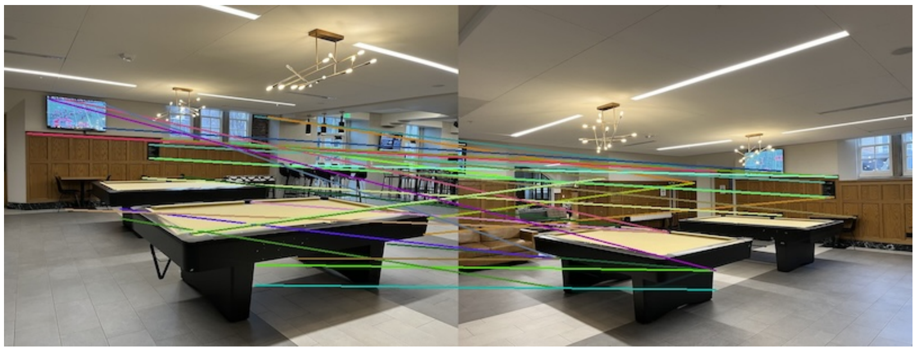
Figure 56: Matching in image pair 2 (using NCC and $\sigma = 2$)

Observations: Similarly, in our custom images, we can observe that larger values of the scale σ lead to less corners detected, and moreover, the detected features tend to be larger. Also, SSD provides slightly better result when matching corners in different views. Interestingly, since the view difference in image pair 1 is smaller than the view difference in image pair 2, the matching algorithm performs relatively better in image pair 1. Thus for large view difference, we may need more advanced algorithms such as deep learning based methods.

3.2 SIFT

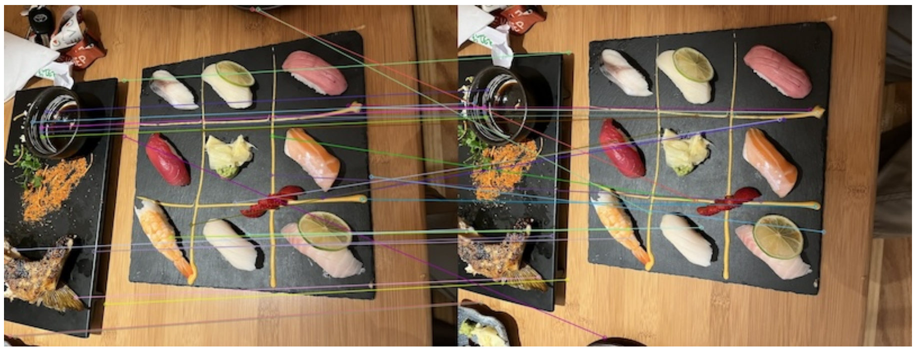

Figure 57: Matching in image pair 1 using SIFT

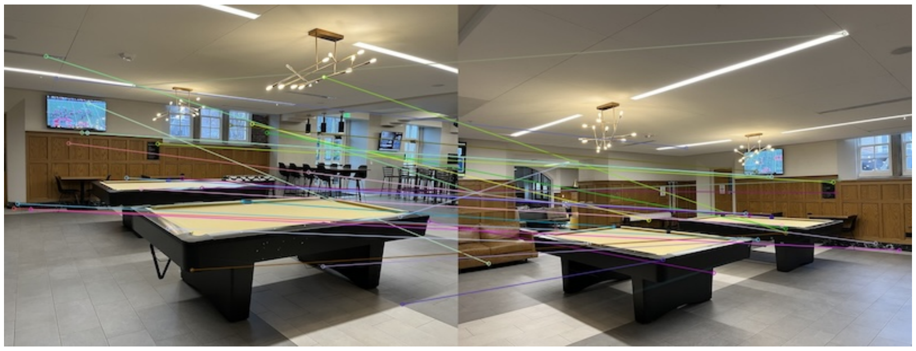

Figure 58: Matching in image pair 2 using SIFT

3.3 SuperPoint and SuperGlue

We observe that in image pair 1, the deep learning based approach matches a large number of interest points. I believe we can reduce the number of interest points detected by adjusting the hyperparameters of the model. Also, we observe that this method achieves high accuracy in image pair 2, and this result is substantially better than SIFT and Harris corner detector.

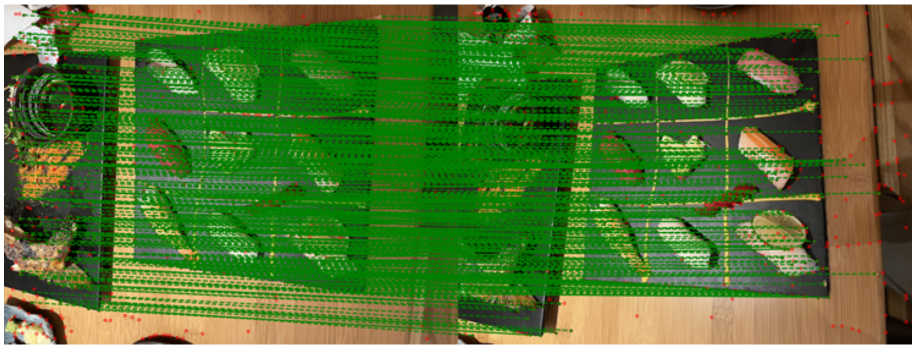
Figure 59: Matching in custom image pair 1

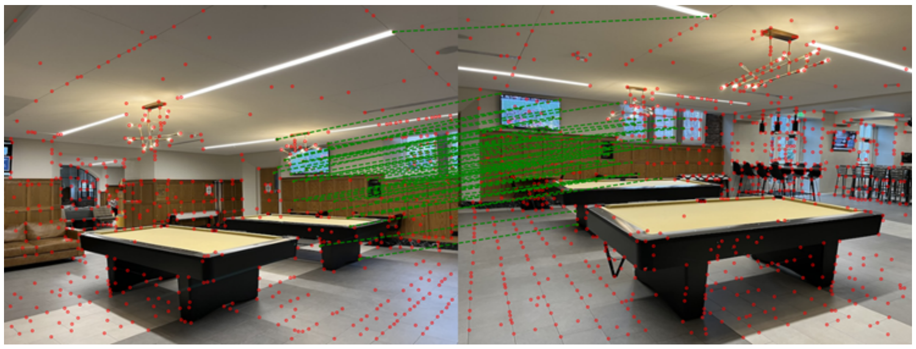
Figure 60: Matching in custom image pair 2

4 Python Code

```
import cv2 as cv
import numpy as np

sigma = 1
source_image_paths = ["HW4_images/hovde_2.jpg", "HW4_images/hovde_3.jpg"]
img_name = "hovde"

def harris_corner_detection(img, sigma):

 # Calculate the smallest even number >= 4 * sigma
 M = np.ceil(4 * sigma).astype(int)
 M += M % 2

 # Haar wavelet filters
 Hx = np.hstack([-np.ones((M, M // 2)), np.ones((M, M // 2))])
 Hy = Hx.T

 # Process the input image and apply the Haar filters
 img0 = cv.cvtColor(img, cv.COLOR_BGR2GRAY) / 255
 dx = cv.filter2D(src=img0, ddepth=-1, kernel=Hx)
 dy = cv.filter2D(src=img0, ddepth=-1, kernel=Hy)

 # Calculate the smallest odd number >= 5 * sigma
 M = np.ceil(5 * sigma).astype(int)
 M += (1 - M % 2)

 # Calculate matrix determinant and trace
 C11 = cv.filter2D(src=dx ** 2, ddepth=-1, kernel=np.ones((M, M)))
 C12 = cv.filter2D(src=dx * dy, ddepth=-1, kernel=np.ones((M, M)))
 C22 = cv.filter2D(src=dy ** 2, ddepth=-1, kernel=np.ones((M, M)))
 det_C = C11 * C22 - C12 ** 2
 tr_C = C11 + C22

 # Calculate the Harris corner response
 k = 0.05  # empirical constant
 R = det_C - k * tr_C ** 2
 t = np.mean(R[R > 0])  # average of the positive entries of R
 threshold = (R >= t)

 # Find local maxima
 local_max = np.zeros_like(R)
 for m in range(M, R.shape[0]-M):
  for n in range(M, R.shape[1]-M):
   if R[m, n] == np.max(R[m-M:m+M+1, n-M:n+M+1]):
    local_max[m, n] = 1

 # Return the top 200 corners based on the values of Harris response
 corners = threshold * local_max

 out = [(x, y, R[y, x]) for y in range(corners.shape[0])
     for x in range(corners.shape[1]) if corners[y, x]]
 out = [(x, y) for x, y, _ in sorted(out, key=lambda t : t[2], reverse=True)]

 img0 = img.copy()
 for x, y in out[:200]:
  cv.circle(img0, (x, y), radius=3, color=(0, 255, 0), thickness=-1)

 return out[:200], img0

def interest_pts_matching(img1, img2, metric="SSD"):
 corners1, img01 = harris_corner_detection(img1, sigma)
 corners2, img02 = harris_corner_detection(img2, sigma)

 cv.imshow("window", np.hstack([img01, img02]))
 cv.imwrite(f"{img_name}_s{sigma}.jpg", np.hstack([img01, img02]))
 cv.waitKey(0)

 # Find the matched interest points and their SSD/NCC
 matching = {}
 M = np.ceil(10 * sigma).astype(int)
 img01 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY) / 255
 img02 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY) / 255
 for x1, y1 in corners1:
  if not (M <= y1 < img1.shape[0] - M and M <= x1 < img1.shape[1] - M):
   continue
  window1 = img01[y1-M:y1+M+1, x1-M:x1+M+1]
  m1 = np.mean(window1)
  min_dist = np.inf
  for x2, y2 in corners2:
   if not (M <= y2 < img2.shape[0] - M and M <= x2 < img2.shape[1] - M):
    continue
   window2 = img02[y2-M:y2+M+1, x2-M:x2+M+1]
   m2 = np.mean(window2)

   dist = np.sum((window1 - window2) ** 2) if metric == "SSD" else \
    - np.sum((window1 - m1) * (window2 - m2)) / \
    np.sqrt(np.sum((window1 - m1) ** 2) * np.sum((window2 - m2) ** 2))

   if dist < min_dist:
    min_dist = dist
    matching[(y1, x1)] = (y2, x2, dist)

 # Sort the matched interes point pairs according to their distance
 matching = sorted(list(matching.items()), key=lambda x : x[-1][-1])

 # Plot the matching for the first 30 pairs
 img0 = np.hstack([img1, img2])
 for t1, t2 in matching[:30]:
  y1, x1, y2, x2 = t1[0], t1[1], t2[0], t2[1]
  c = np.random.randint(0, 256, size=3)
  cv.line(img0, (x1, y1), (x2+img1.shape[1], y2), (int(c[0]), int(c[1]), int(c[2])), 2)
 cv.imshow("window", img0)
 cv.imwrite(f"{img_name}_s{sigma}_{metric}.jpg", img0)
 cv.waitKey(0)

def SIFT(img1, img2):
 sift = cv.SIFT_create()

 # Calculate keypoints and descriptors
 kp1, des1 = sift.detectAndCompute(cv.cvtColor(img1, cv.COLOR_BGR2GRAY), None)
 kp2, des2 = sift.detectAndCompute(cv.cvtColor(img2, cv.COLOR_BGR2GRAY), None)

 # Match descriptors and sort the matched pairs according to distance
 bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)
 matches = sorted(bf.match(des1, des2), key=lambda x : x.distance)
 # Plot the first 30 matched pairs
 img0 = cv.drawMatches(img1, kp1, img2, kp2, matches[:30], None, flags=2)
 cv.imshow("window", img0)
 cv.imwrite(f"{img_name}_sift.jpg", img0)
 cv.waitKey(0)

img1, img2 = cv.imread(source_image_paths[0]), cv.imread(source_image_paths[1])
cv.imshow("window", np.hstack([img1, img2]))
cv.imwrite(f"{img_name}.jpg", np.hstack([img1, img2]))
cv.waitKey(0)
interest_pts_matching(img1, img2, metric="SSD")
interest_pts_matching(img1, img2, metric="NCC")
SIFT(img1, img2)
```