Theory 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?

Given an image f, its σ-smoothed version is $\mathit{ff}(x, y, \sigma)$:

$$\mathit{ff}(x, y, \sigma) = \iint_{-\infty}^{\infty} f(x', y') g_\sigma(x - x', y - y') dx' dy'$$

where the $g_\sigma(x, y)$ is the Gaussian function $g_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$. The scale factor σ controls the scale of smoothing and when taking the partial derivative with respect to σ in scale space we get:

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

Which can be simplified to:

$$\frac{\partial \mathit{ff}(x, y, \sigma)}{\partial \sigma} = \sigma f(x, y) * h(x, y, \sigma)$$

Going back to the original image, if we apply Laplacian-of-Gaussian LoG to the image:

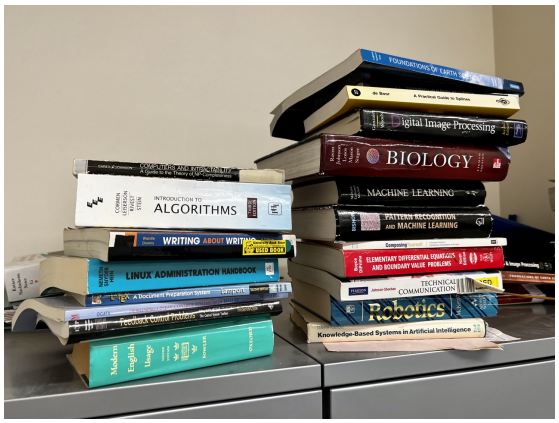
$$LoG(f(x, y)) = \nabla^2 \mathit{ff}(x, y) = f(x, y) * h(x, y, \sigma)$$

This is the same as taking the partial derivative of the Gaussian function with respect to σ. Therefore, the LoG can be estimated by subtracting the $(\sigma + \delta\sigma)$-smoothed version of $f(x, y)$ from the σ-smoothed version of $f(x, y)$, which is the Difference-of-Gaussian DoG.

Using DoG has computational advantages. Since it is separable in x and y, the 2-D smoothing can be done by two 1-D smoothing, first along in one direction and then the result of that can be smoothed in the other direction. LoG is not separable and hence this cannot be done with LoG. Since DoG allows 1-D smoothing, the DoG operator is smaller than the LoG operator which does 2D smoothing. If we take $\sigma = \sqrt{2}$, the width of the 1-D operator for DoG will be 9 pixels, while the width of the 2-D operator for LoG will be 13 pixels.

Task 1

For this task you are given an image pair per scene, first image from each pair is shown in fig.1. Compare the quality of the correspondences obtained from the following three sub-tasks and state your observations.



(a) Image from pair 1

(b) Image from pair 2

Figure 1: Input Images

Task 1.1.1: Harris Corner Detector

Background

Harris Corner Detector is a corner detector that interprets corners as the junction of two edges. It does this by calculating gradients in x and y directions. If the gradients in both directions are above some specific threshold, there exist two edges, which makes that pixel a possible corner.

Brief Summary of the process:

1. A scale σ is selected.
2. First order derivatives d_x and d_y are calculated at each pixel based on the selected σ.
3. For each pixel, a $5\sigma \times 5\sigma$ neighborhood is created
4. In the neighborhood, the existence of corner is determined. If the corner exist, the pixel is marked as a corner or interest points.
5. This is performed on both images of the same scene.
6. SSD and NCC metrics are used to establish correspondences between the two sets of interest points of the image pairs.

Selecting the scale σ: For this homework, we need to select 4 different σ. Hence, the following values are used and the above process is performed for each:

$$\sigma_1 = 0.8$$
$$\sigma_2 = 1.2$$
$$\sigma_3 = 1.4$$
$$\sigma_4 = 1.6$$

Calculating 1^{st} order derivatives d_x and d_y: To calculate the first order derivatives, the Haar Wavelet Filter is used. The Haar wavelet filter has the basic form $\begin{bmatrix}-1 & 1\end{bmatrix}$ along x and $\begin{bmatrix}1 & -1\end{bmatrix}^T$ along y. These forms are scaled up to a M × M operator where M is the smallest even integer greater than 4σ.

For example: for $\sigma = 0.8$, M $= 4 \times 0.8 = 3.2$. Therefore the 4×4 operators for calculating d_x and d_y in x direction and y direction are:

$$k_x = \begin{bmatrix}-1 & -1 & 1 & 1\\ -1 & -1 & 1 & 1\\ -1 & -1 & 1 & 1\\ -1 & -1 & 1 & 1\end{bmatrix} \quad k_y = \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ -1 & -1 & -1 & -1\\ -1 & -1 & -1 & -1\end{bmatrix}$$

Using the Haar wavelet operators calculated based on σ, d_x and d_y are calculated by convolving the operators with the image. For this homework, the OpenCV function "filter2D" is used since it performs vectorized operations.

Creating $5\sigma \times 5\sigma$ neighborhood: For this step, an operator of $5\sigma \times 5\sigma$ is created and for each pixel a matrix C is created based on the $5\sigma \times 5\sigma$ neighborhood. This is achieved by convolving the $5\sigma \times 5\sigma$ operator with the derivatives. For example, for $\sigma = 0.8$,

$$k_{5\sigma} = \begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix} \quad C_{(x,y)} = \begin{bmatrix}\sum d_x^2 & \sum d_x d_y\\ \sum d_x d_y & \sum d_y^2\end{bmatrix}$$

Determining the existence of a corner: The 2×2 C matrix calculated above for each pixel has a rank of 2 if and only if the corner exists at that pixel. Rather than calculating eignenvalues to determine the singularity of each C matrix, a more efficient way is used where a ratio R is calculated from the determinant and the square of the trace of the Matrix C which can then be used to determine a corner.

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

where k $= \frac{det(C)}{Tr(C)^2}$ and the threshold is set based on the largest 5% of all the R calculated. A non-maximum suppression method was used to eliminate excessive points that represented the same corner.

Establishing correspondences using SSD and NCC: Once the corners (interest points) are estimated on both images of the scene, a neighborhood of (M + 1) × (M + 1) is used to

establish correspondence by comparing the gray levels around the corner pixel in one image with the gray levels of the corresponding pixel in the other image. A distance metric is used for comparison and the pair of corner points that yield the lowest distance value are considered to be the same point in the real world scene. The distance metrics used in this homework are:

Sum of Squared Differences (SSD):

$$\text{SDD} = \sum_x \sum_y |f_1(x,y) - f_2(x,y)|^2$$

Normalized cross-correlation (NCC):

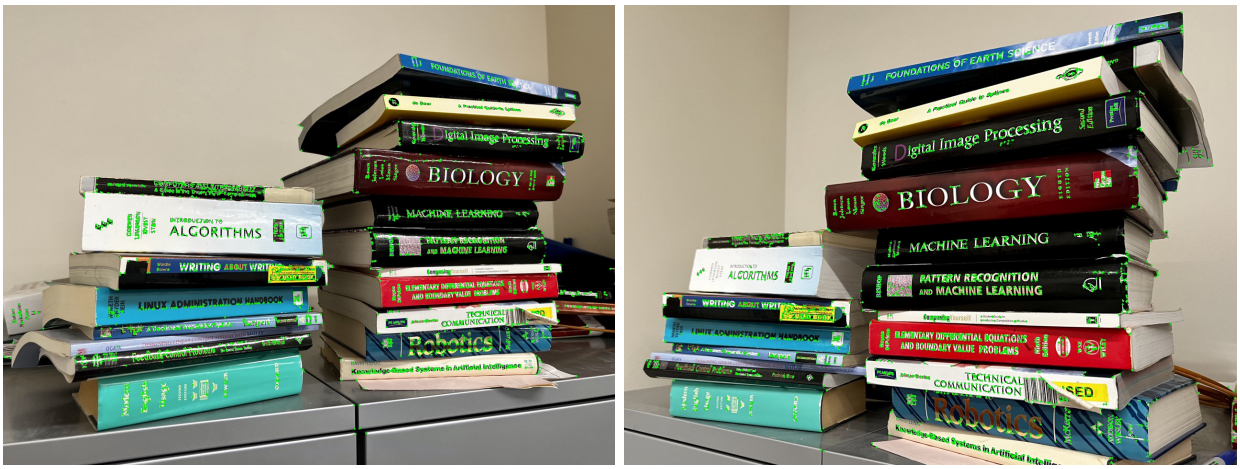
$$\text{NCC} = \frac{\sum_x \sum_y (f_1(x,y) - m_1)(f_2(x,y) - m_2)}{\sqrt{\left[\sum_x \sum_y (f_1(x,y) - m_1)^2\right]\left[\sum_x \sum_y (f_2(x,y) - m_2)^2\right]}}$$

where $f_1(x,y)$ are pixel coordinates from image 1 and $f_2(x,y)$ are pixel coordinates from image 2. m_1 is the mean from the neighborhood in image 1 and m_2 is the mean from the neighborhood in image 2.

Output

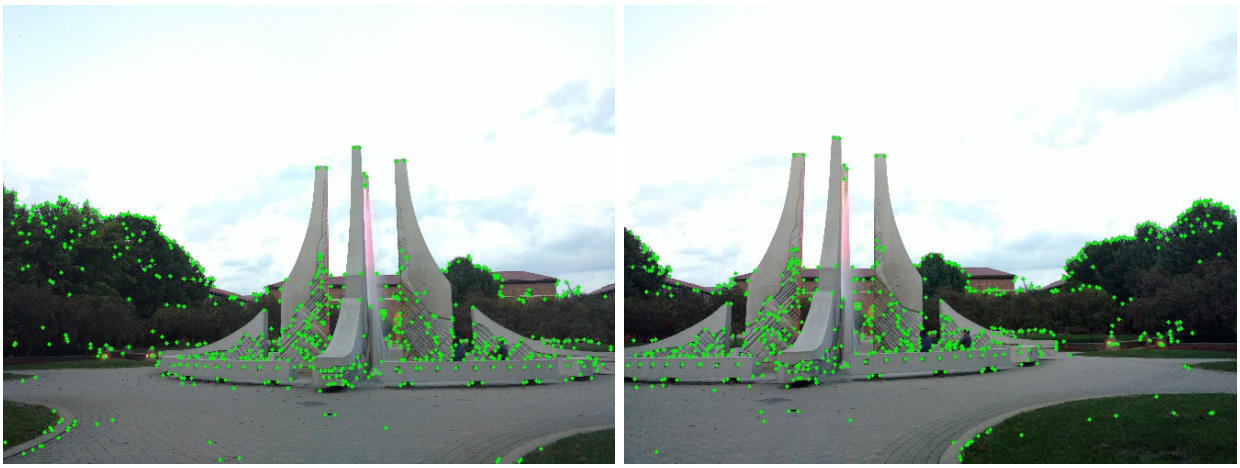
Note: To show better matching between points, only the first 100 correspondences are drawn on the images.

Keypoints for $\sigma = 0.8$:



(a) Book 1 for $\sigma = 0.8$ (b) Book 2 for $\sigma = 0.8$

Figure 2: Interest points for $\sigma = 0.8$



(a) Fountain 1 for $\sigma = 0.8$ (b) Fountain 2 for $\sigma = 0.8$

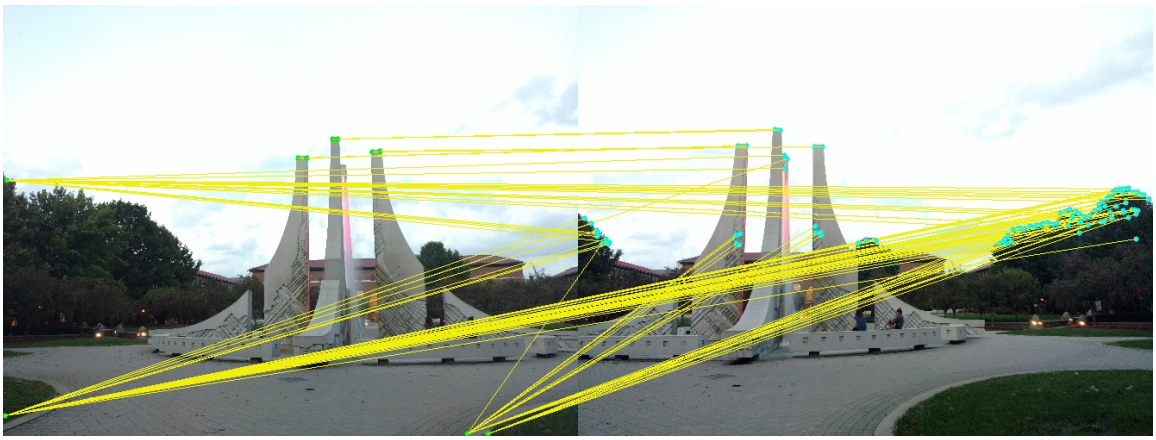
Figure 3: Interest points for $\sigma = 0.8$

Keypoint correspondences for $\sigma = 0.8$:

(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 4: Interest Point Correspondence for $\sigma = 0.8$

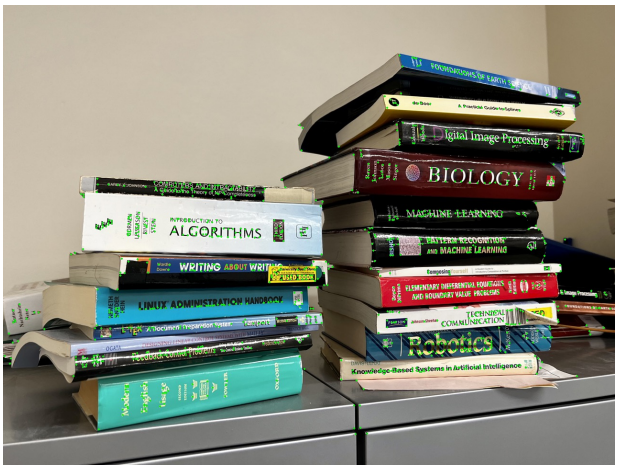


(a) Interest Point Correspondence using SSD

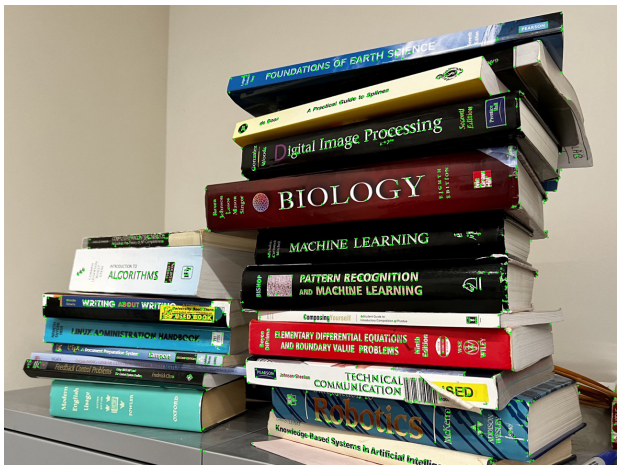
(b) Interest Point Correspondence using NCC

Figure 5: Interest Point Correspondence for $\sigma = 0.8$

Keypoints for $\sigma = 1.2$:

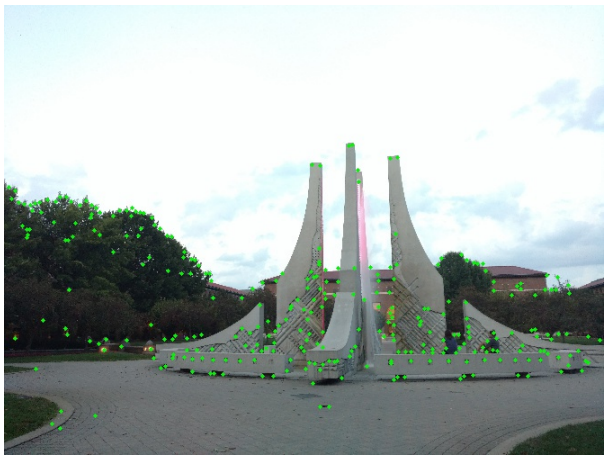


(a) Book 1 for $\sigma = 1.2$

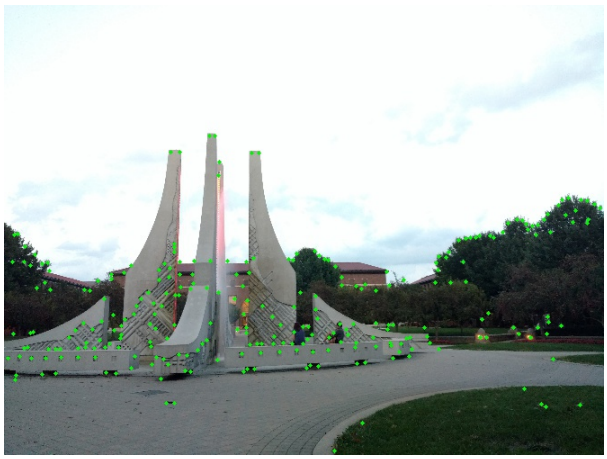


(b) Book 2 for $\sigma = 1.2$

Figure 6: Interest points for $\sigma = 1.2$



(a) Fountain 1 for $\sigma = 1.2$



(b) Fountain 2 for $\sigma = 1.2$

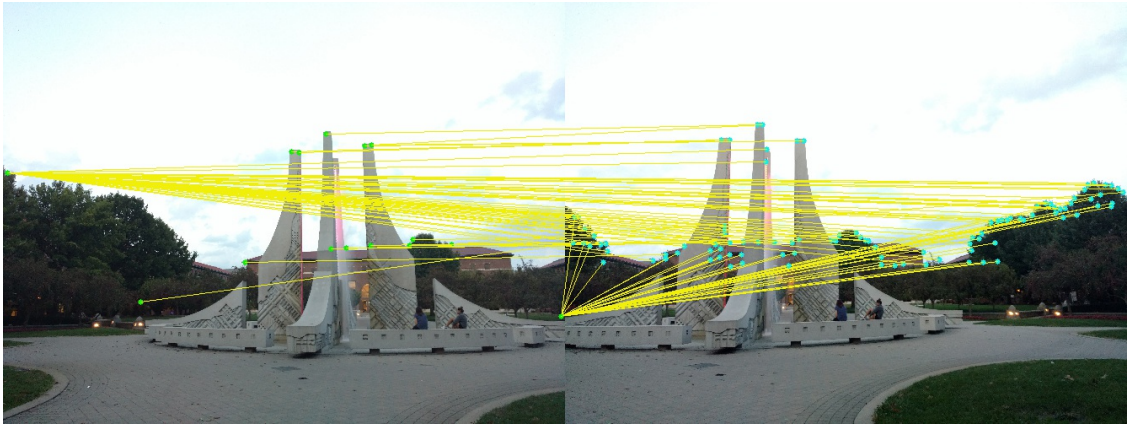
Figure 7: Interest points for $\sigma = 1.2$

Keypoint correspondences for $\sigma = 1.2$:

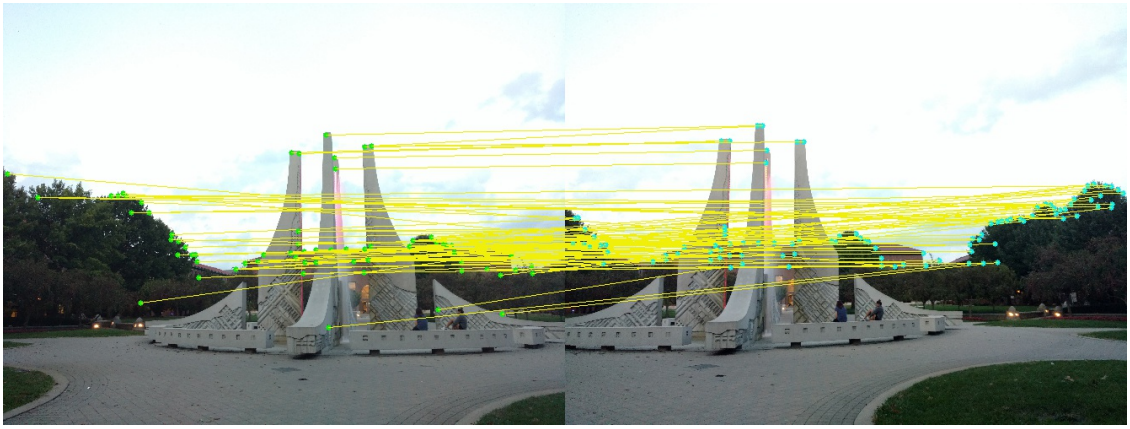
(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 8: Interest Point Correspondence for $\sigma = 1.2$



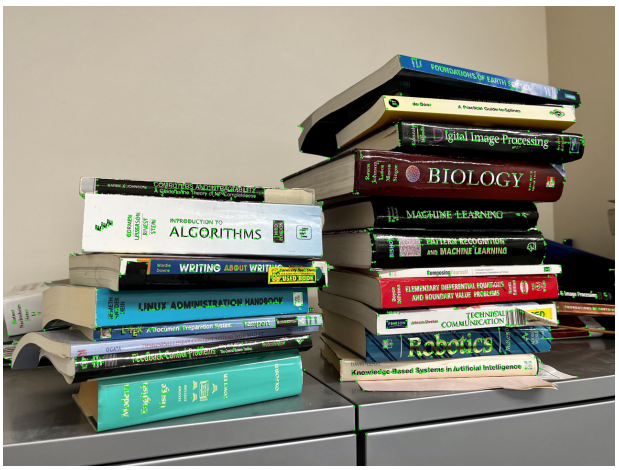
(a) Interest Point Correspondence using SSD



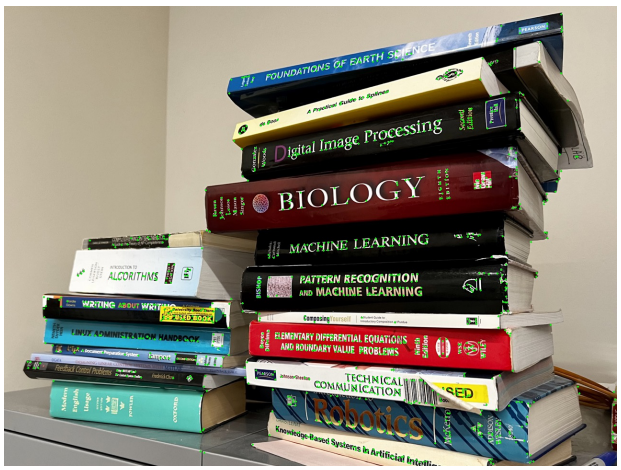
(b) Interest Point Correspondence using NCC

Figure 9: Interest Point Correspondence for $\sigma = 1.2$

Keypoints for $\sigma = 1.4$:

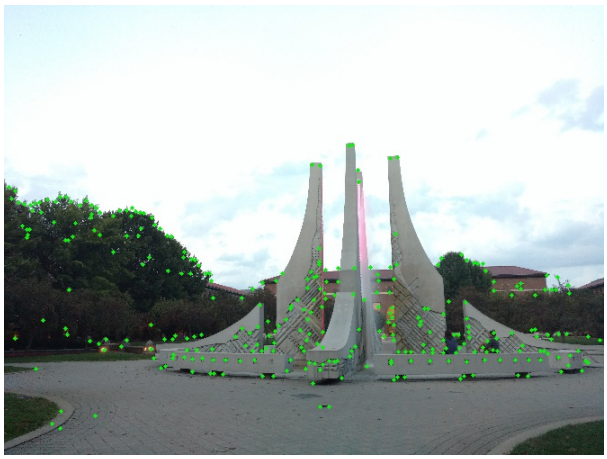


(a) Book 1 for $\sigma = 1.4$

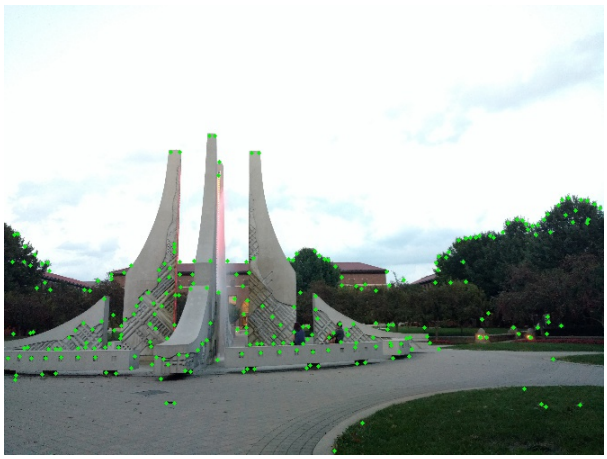


(b) Book 2 for $\sigma = 1.4$

Figure 10: Interest points for $\sigma = 1.4$



(a) Fountain 1 for $\sigma = 1.4$



(b) Fountain 2 for $\sigma = 1.4$

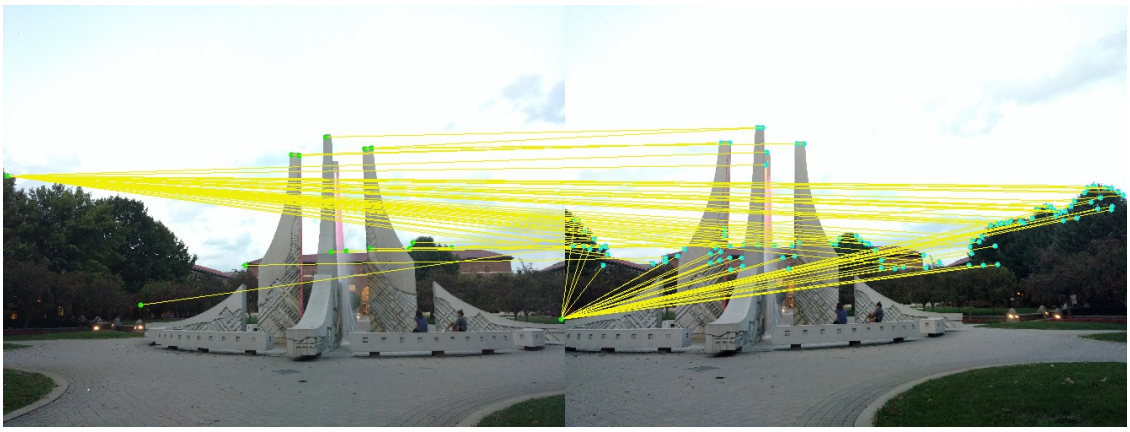
Figure 11: Interest points for $\sigma = 1.4$

Keypoint correspondences for $\sigma = 1.4$:

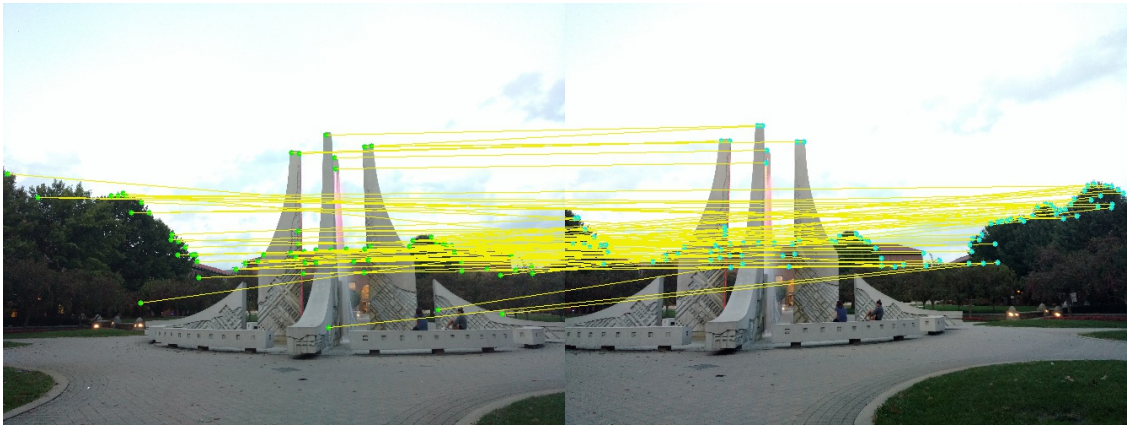
(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 12: Interest Point Correspondence for $\sigma = 1.4$



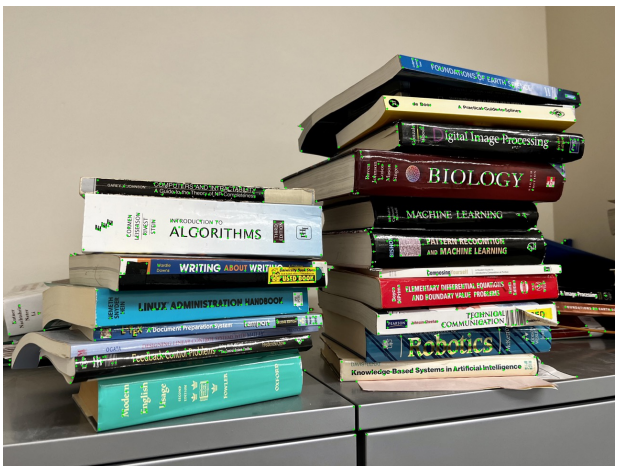
(a) Interest Point Correspondence using SSD



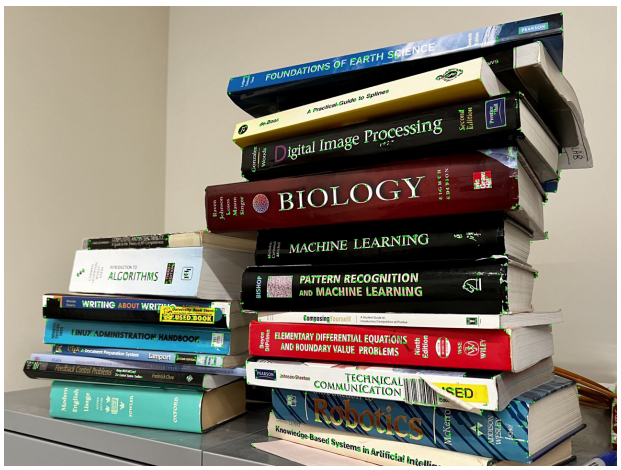
(b) Interest Point Correspondence using NCC

Figure 13: Interest Point Correspondence for $\sigma = 1.4$

Keypoints for $\sigma = 1.6$:

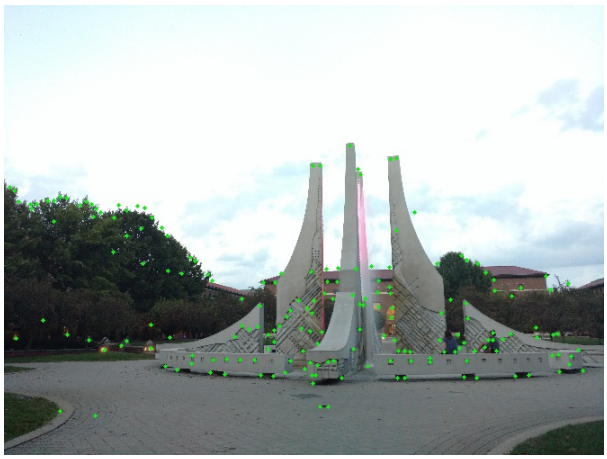


(a) Book 1 for $\sigma = 1.6$



(b) Book 2 for $\sigma = 1.6$

Figure 14: Interest points for $\sigma = 1.6$



(a) Fountain 1 for $\sigma = 1.6$

(b) Fountain 2 for $\sigma = 1.6$

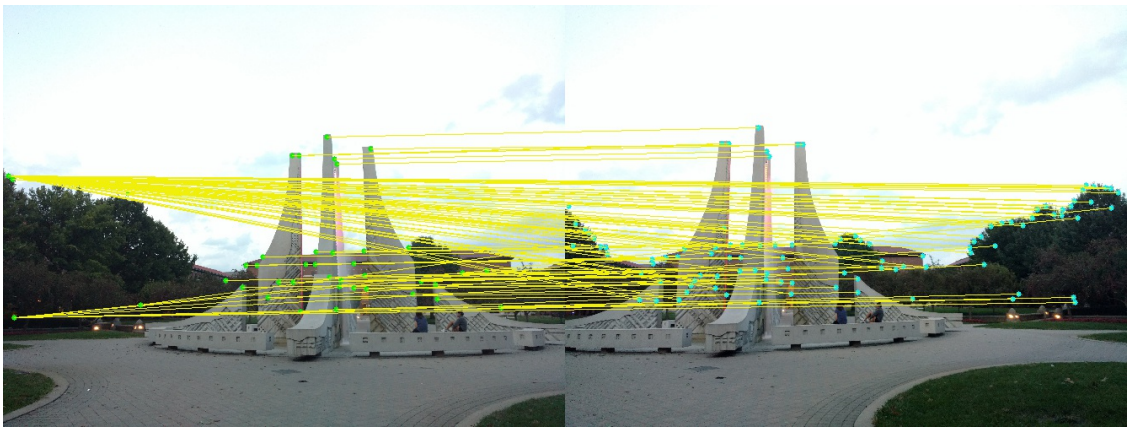
Figure 15: Interest points for $\sigma = 1.6$

Keypoint correspondences for $\sigma = 1.6$:

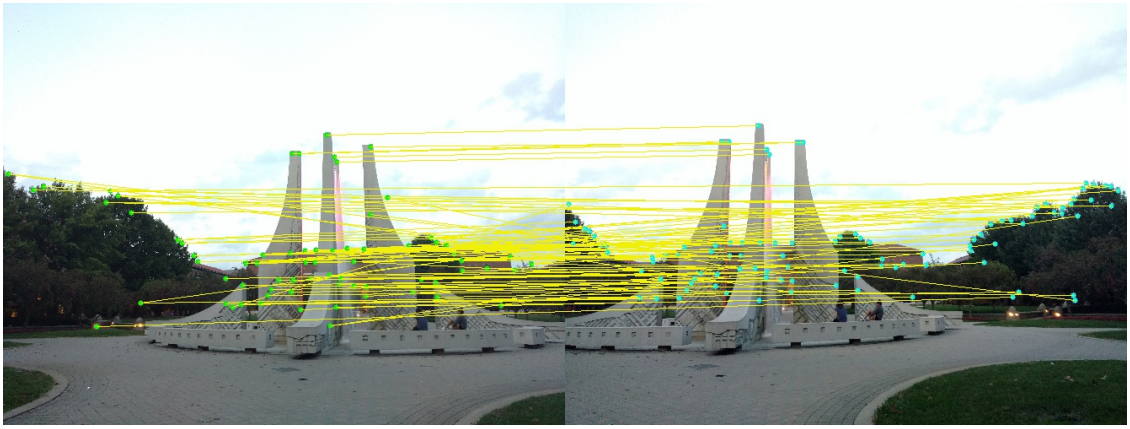
(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 16: Interest Point Correspondence for $\sigma = 1.6$



(a) Interest Point Correspondence using SSD



(b) Interest Point Correspondence using NCC

Figure 17: Interest Point Correspondence for $\sigma = 1.6$

Observations:

As can be seen from the images above, as σ increases, the number of keypoints decreases. While Harris Corner detector worked well on the fountain images, it did not do well on the books images. This may be because the books images have a high density of corner points (corner points packed very close together), it fails at finding the right correspondence. While the keypoints in the fountain images are well spread out and thus get a better correspondence matches. Lastly, for better visual presentation of the outputs, only the first 100 correspondences were drawn on the images. NCC performed better than SSD since it had more correspondences that matched correctly.

Task 1.1.2: SIFT Algorithm

Background

While Harris Corner detector is rotation invariant, it is not scale invariant. SIFT stands for Scale-Invariant Feature Transform and it consists of four main steps:

1. **Scale-space Extrema detection:** The Guassian Pyramid is constructed by calculating the DoG for different octaves of the pyramid. DoG is calculated as the difference of Guassian blurring of the image with two different σ.

   Once the DoG is calculated, the images are searched for a local extrema. This is done by comparing each pixel with its 8 neighbors, 9 neighbors on the scale space above and 9 neighbors on the scale below. If the pixel is the local extrema, it is considered as a potential keypoint.

2. **Keypoint Localization:** As σ increases, the images get more coarse. A keypoint located on a scale that is coarse may not be accurate in pointing to pixel in the original image. To improve the localization of the keypoints, Taylor series expansion of the DoG in the vicinity of the location of the keypoint found in step 1 is used. If the keypoint found in step 1 has a location $\boldsymbol{x_0} = \begin{bmatrix} x_0 & y_0 & \sigma_0 \end{bmatrix}^T$, the true location of the keypoint is given by $\boldsymbol{x} = -\mathrm{H}^{-1}(\boldsymbol{x_0}) \cdot \mathrm{J}(\boldsymbol{x_0})$. $\mathrm{H}(\boldsymbol{x_0})$ is the Hessian calculated at $\boldsymbol{x_0}$ and $\mathrm{J}(\boldsymbol{x_0})$ is the gradient calculated at $\boldsymbol{x_0}$.

   Weak keypoints that have a extrema value less than 0.03 are rejected. Since DoG also marks edges as keypoints, they need to be rejected. A 2×2 Hessian matrix H computes the principal curvature by calculating the ratio of the eigenvalues. For edges, one eigenvalue is greater than the other eigenvalue, and hence if the principal curvature is larger than a threshold (typically 10), then the keypoint is rejected for being a potential edge rather than a corner.

3. **Orientation Assignment:** To make the keypoints invariant to in-plane rotations of the image, a dominant local orientation is assigned to each keypoint. To find this orientation, gradient magnitude and direction are computed in a K×K neighborhood around the keypoint. An orientation histogram of 36 bins covering 360° is created. The highest peak in the histogram gives us the dominant local orientation for that keypoint.

4. **Keypoint Descriptor:** A 16×16 neighborhood around the keypoint is taken and divided into 16 4×4 blocks. For each block, a 8 bin histogram is calculated. Each keypoint therefore has 128 bins, represented as a dimensional vector. This vector is forms the keypoint descriptor.

5. **Keypoint Matching:** The correspondence between keypoints is determined by finding the nearest neighbors. For this homework, OpenCV's BFMatcher algorithm is used.

Output

(a) Interest Point Correspondence

(b) Interest Point Correspondence

Figure 18: Interest Point Correspondence using SIFT

Observation

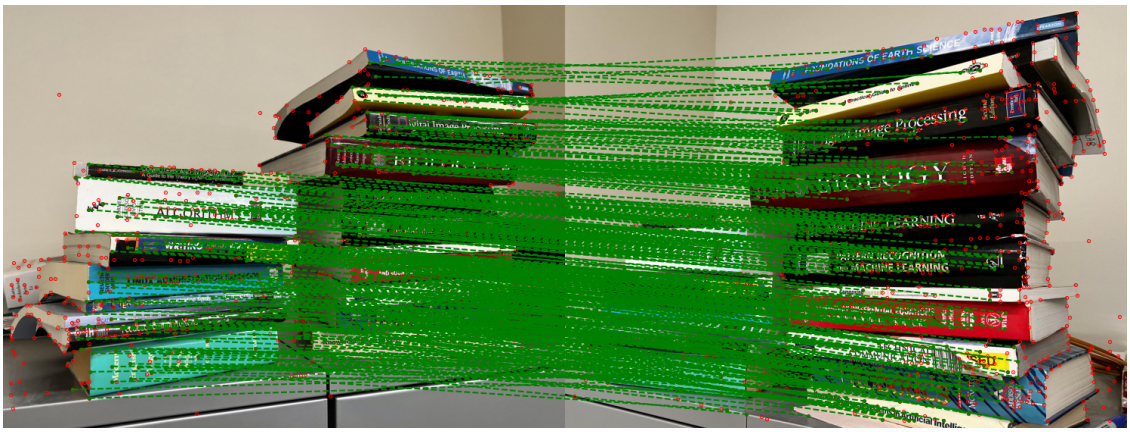
As expected, SIFT gives a much better correspondence between points. This is because of the Brute Force Matcher algorithm that SIFT algorithm in OpenCV uses to find correspondences. Again, for better visual representation of the matching, only the first 100 correspondences were drawn on the images.

Task 1.2:SuperPoint and SuperGlue

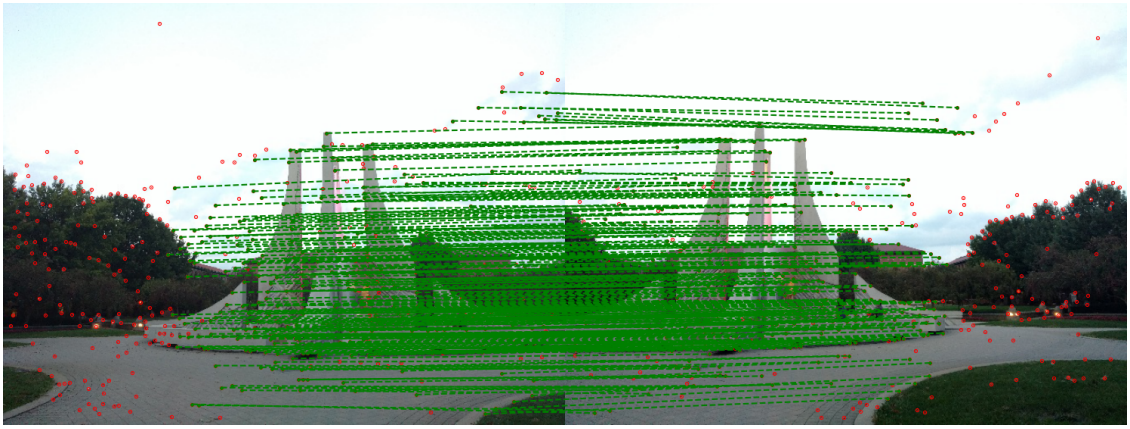
Background

SuperPoint is used for point detection and SuperGlue is used to find the correspondences between the points.

Output



(a) Interest Point Correspondences from SuperGlue



(b) Interest Point Correspondences From SuperGlue

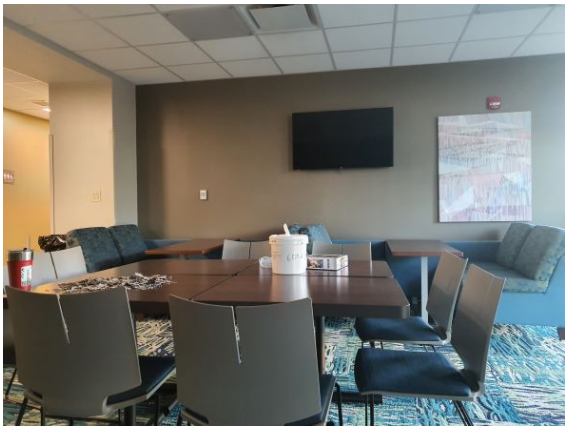
Figure 19: Interest Point Correspondences using SuperPoint and SuperGlue

Observation

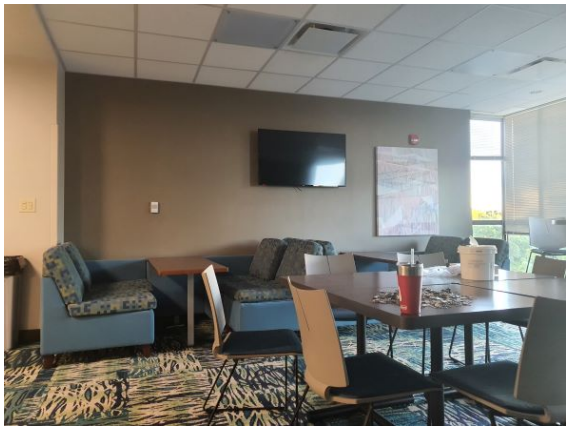
SuperPoint and SuperGlue gave the best results. It worked better than Harris and SIFT.

Task 2

The image pairs that I used for this homework are shown below:



(a) Lounge 1



(b) Lounge 2

Figure 20: Input Image pair

(a) Studio 1

(b) Studio 2

Figure 21: Input Image pair

Task 1.1.1: Harris Corner Detector

Output

Keypoints for $\sigma = 0.8$:

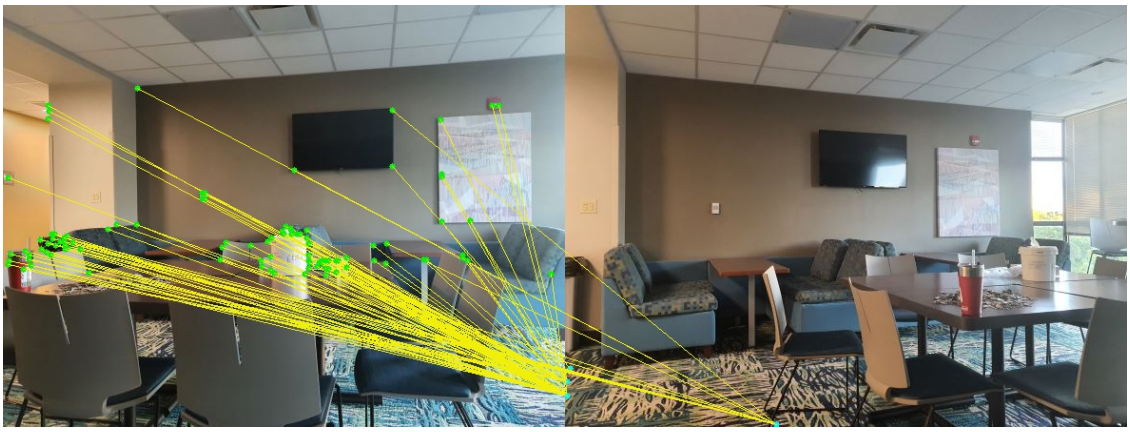
(a) Lounge 1 for $\sigma = 0.8$ (b) Lounge 2 for $\sigma = 0.8$

Figure 22: Interest points for $\sigma = 0.8$

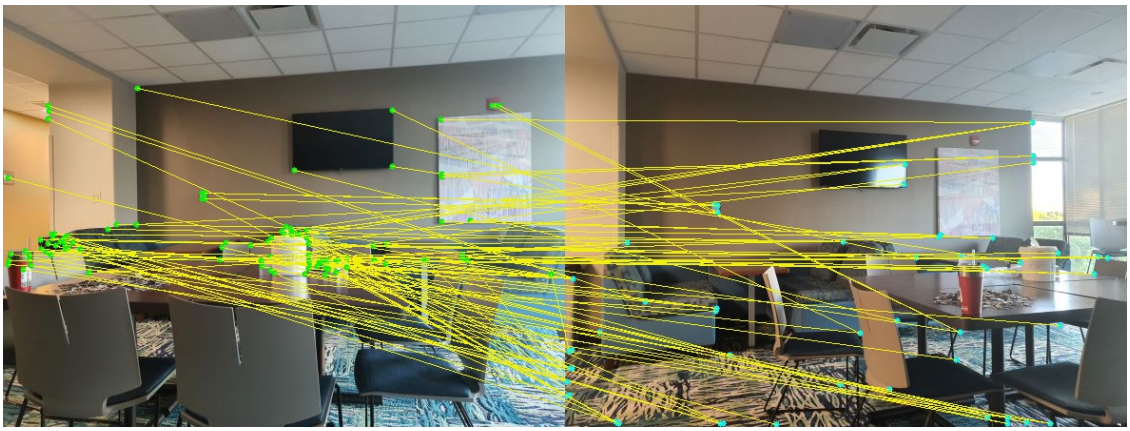
(a) Studio 1 for $\sigma = 0.8$ (b) Studio 2 for $\sigma = 0.8$

Figure 23: Interest points for $\sigma = 0.8$

Keypoint correspondences for $\sigma = 0.8$:

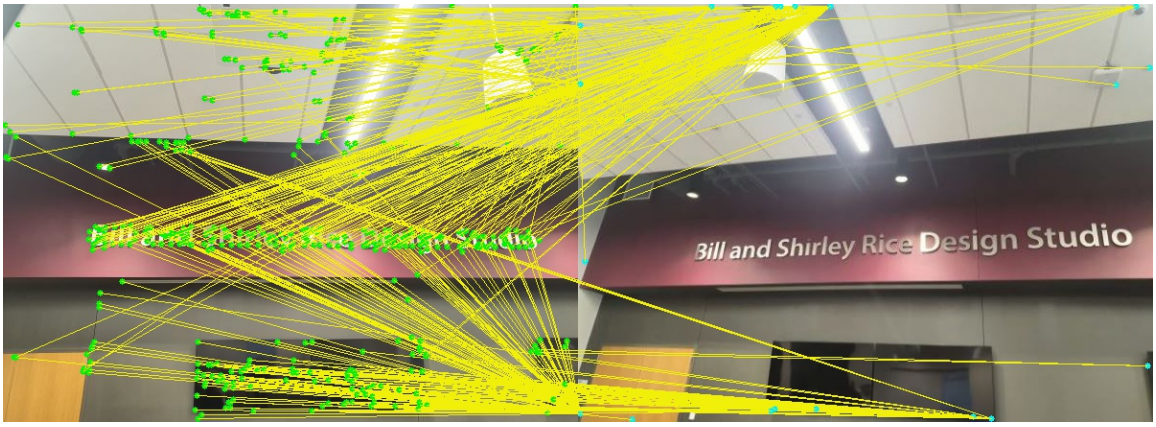


(a) Interest Point Correspondence using SSD



(b) Interest Point Correspondence using NCC

Figure 24: Interest Point Correspondence for $\sigma = 0.8$



(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 25: Interest Point Correspondence for $\sigma = 0.8$

Keypoints for $\sigma = 1.2$:

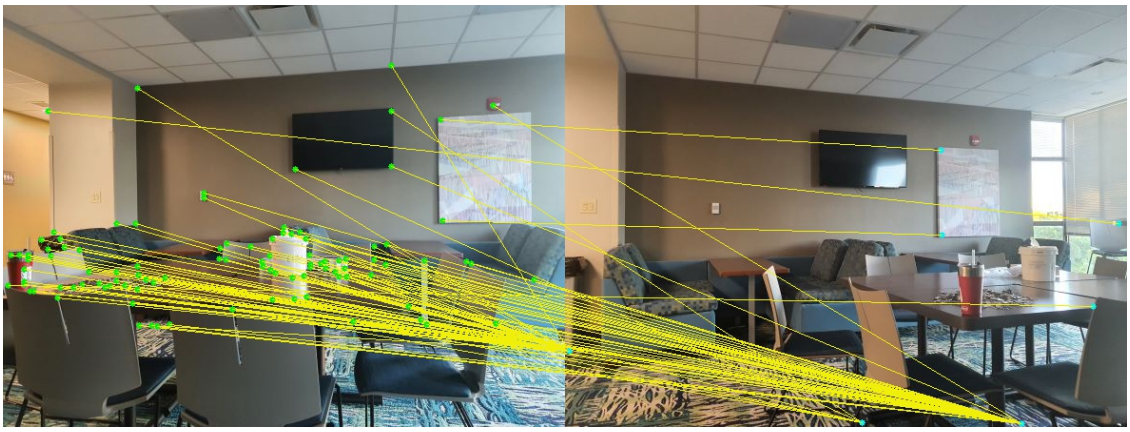
(a) Lounge 1 for $\sigma = 1.2$ (b) Lounge 2 for $\sigma = 1.2$

Figure 26: Interest points for $\sigma = 1.2$

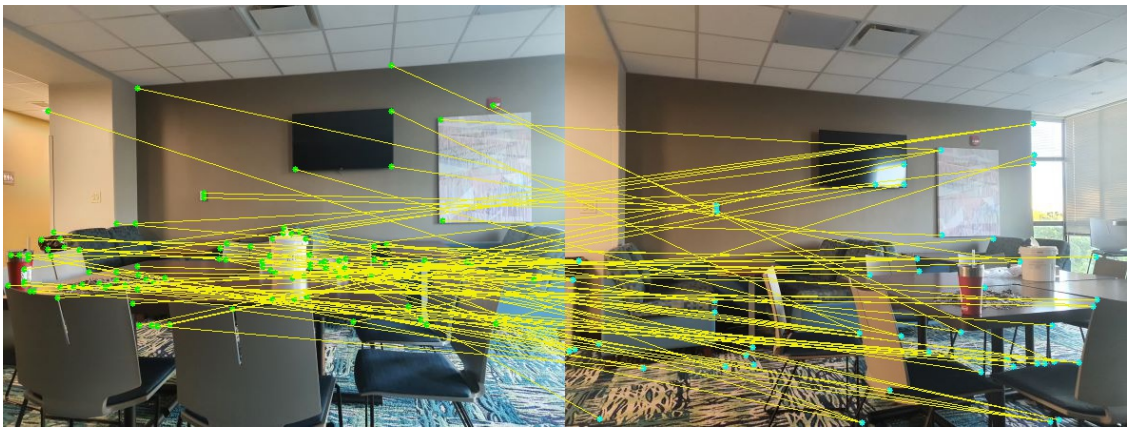
(a) Studio 1 for $\sigma = 1.2$ (b) Studio 2 for $\sigma = 1.2$

Figure 27: Interest points for $\sigma = 1.2$

Keypoint correspondences for $\sigma = 1.2$:

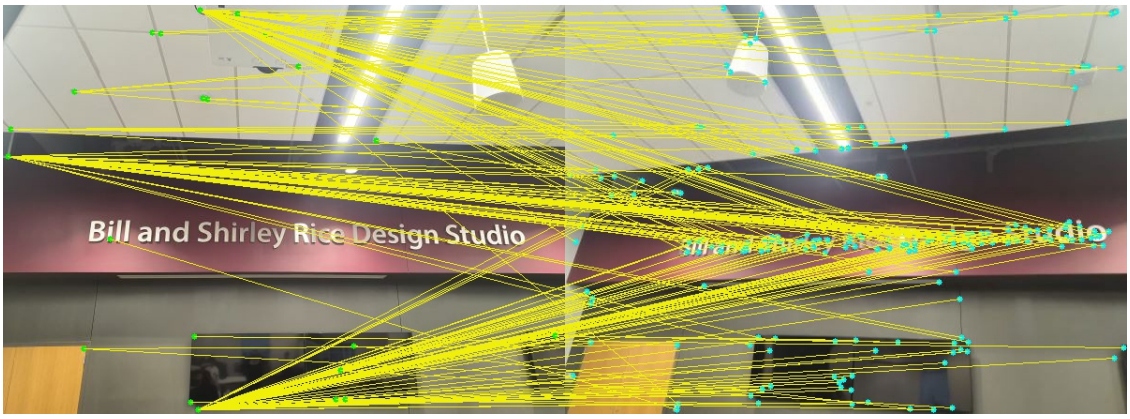


(a) Interest Point Correspondence using SSD

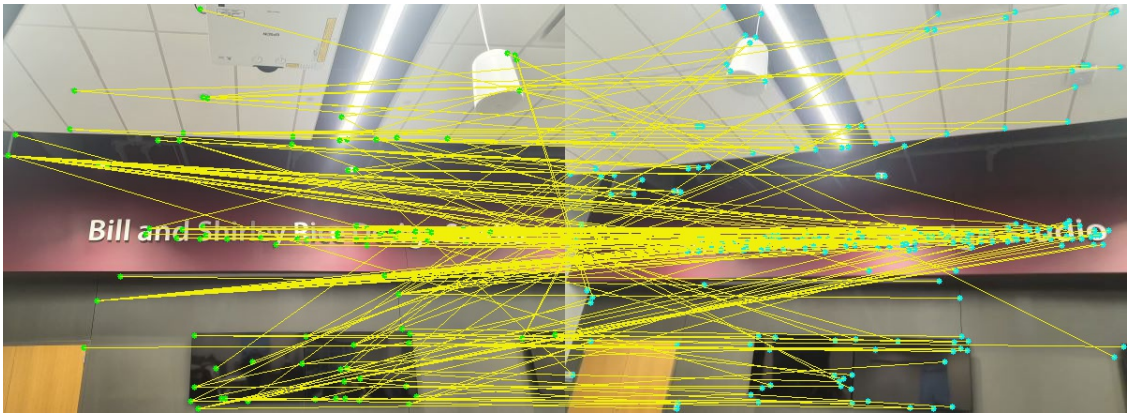


(b) Interest Point Correspondence using NCC

Figure 28: Interest Point Correspondence for $\sigma = 1.2$



(a) Interest Point Correspondence using SSD



(b) Interest Point Correspondence using NCC

Figure 29: Interest Point Correspondence for $\sigma = 1.2$

Keypoints for $\sigma = 1.4$:

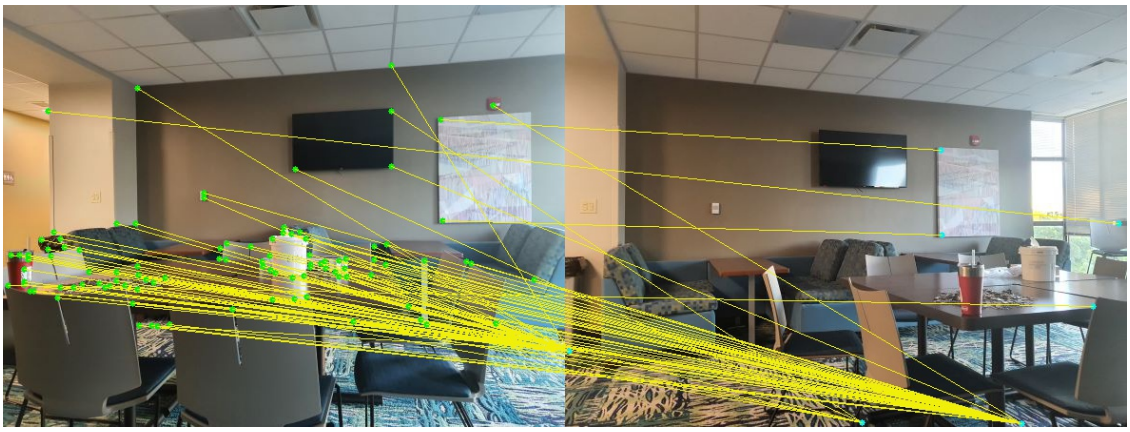
(a) Lounge 1 for $\sigma = 1.4$ (b) Lounge 2 for $\sigma = 1.4$

Figure 30: Interest points for $\sigma = 1.4$

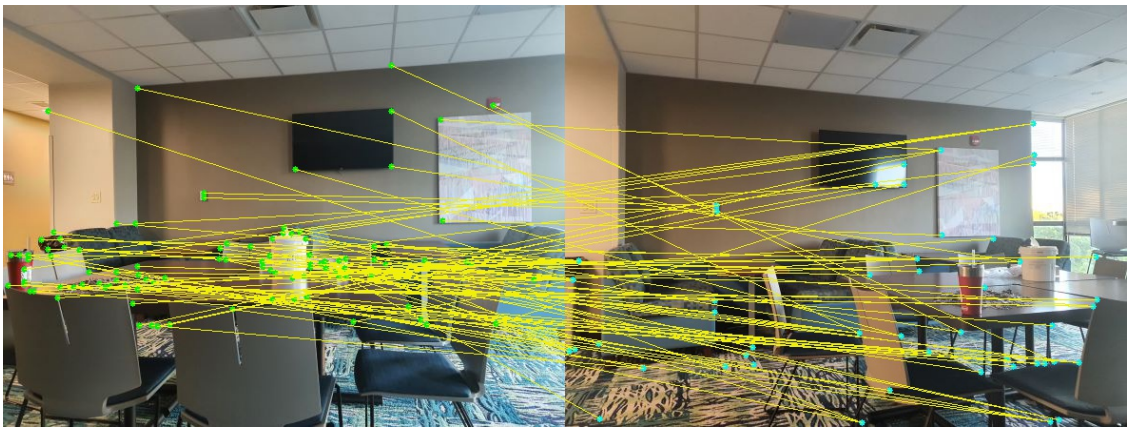
(a) Studio 1 for $\sigma = 1.4$ (b) Studio 2 for $\sigma = 1.4$

Figure 31: Interest points for $\sigma = 1.4$

Keypoint correspondences for $\sigma = 1.4$:

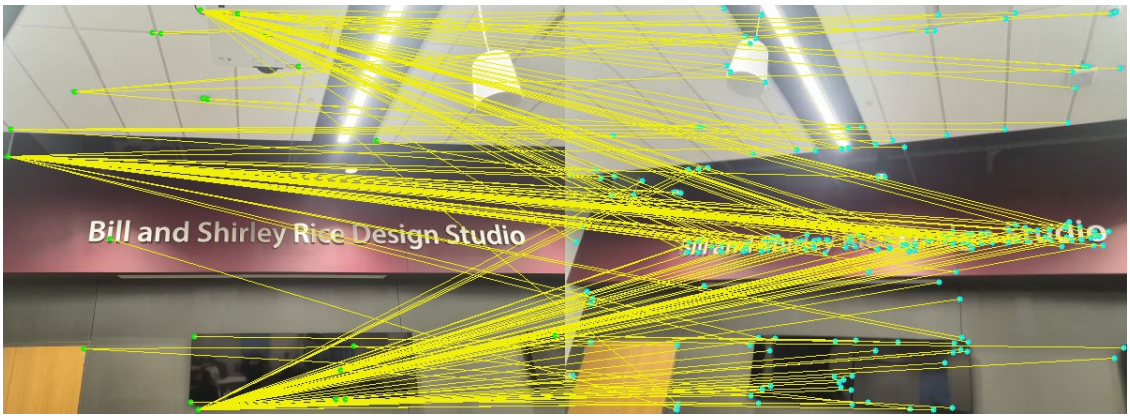


(a) Interest Point Correspondence using SSD

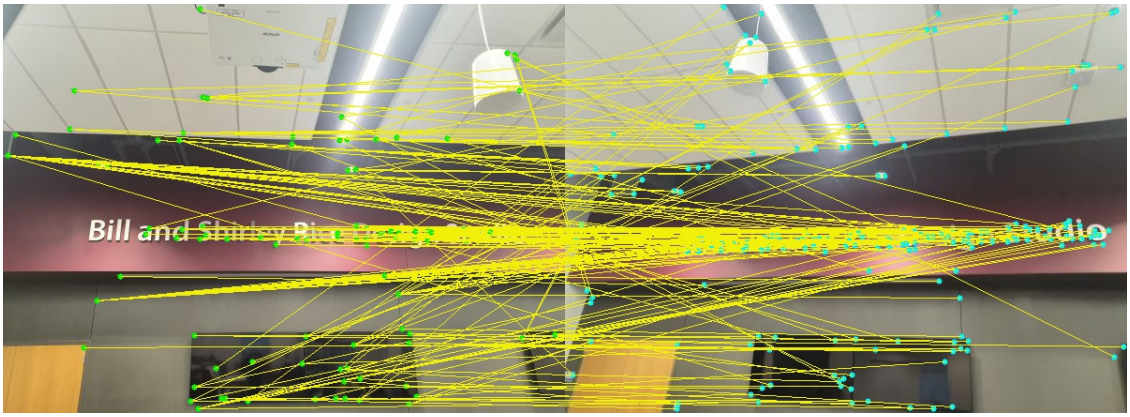


(b) Interest Point Correspondence using NCC

Figure 32: Interest Point Correspondence for $\sigma = 1.4$



(a) Interest Point Correspondence using SSD



(b) Interest Point Correspondence using NCC

Figure 33: Interest Point Correspondence for $\sigma = 1.4$

Keypoints for $\sigma = 1.6$:

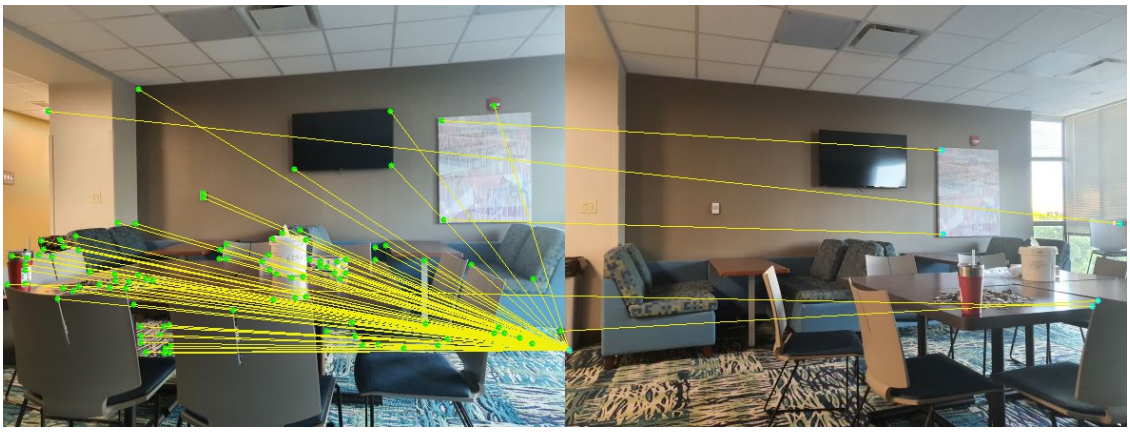
(a) Lounge 1 for $\sigma = 1.6$ (b) Lounge 2 for $\sigma = 1.6$

Figure 34: Interest points for $\sigma = 1.6$

(a) Studio 1 for $\sigma = 1.6$ (b) Studio 2 for $\sigma = 1.6$

Figure 35: Interest points for $\sigma = 1.6$

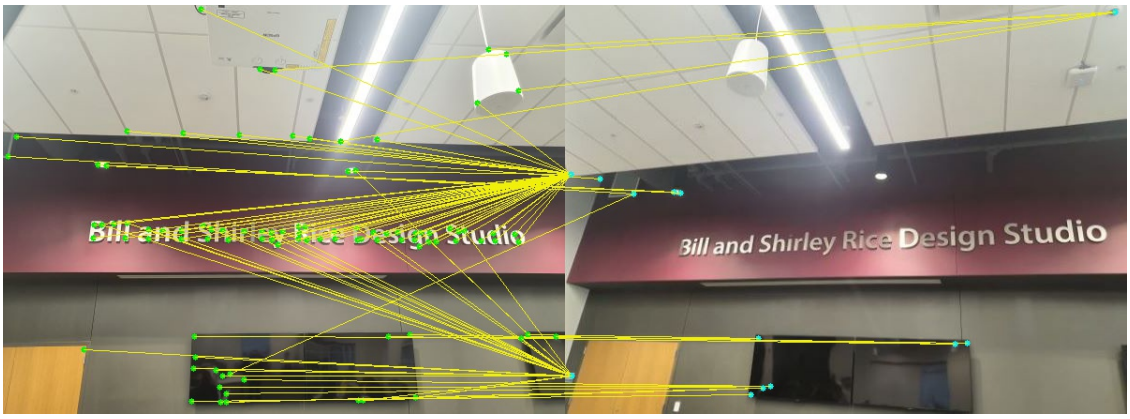
Keypoint correspondences for $\sigma = 1.6$:



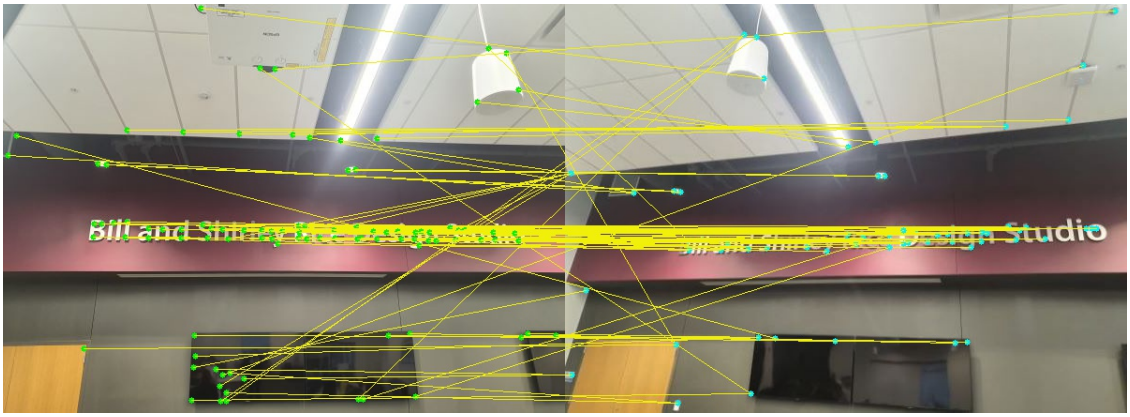
(a) Interest Point Correspondence using SSD

(b) Interest Point Correspondence using NCC

Figure 36: Interest Point Correspondence for $\sigma = 1.6$



(a) Interest Point Correspondence using SSD

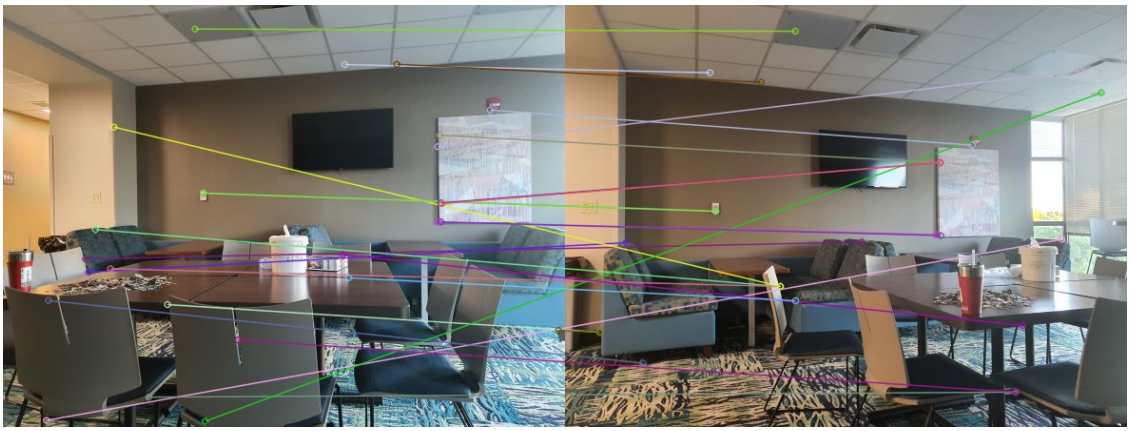


(b) Interest Point Correspondence using NCC

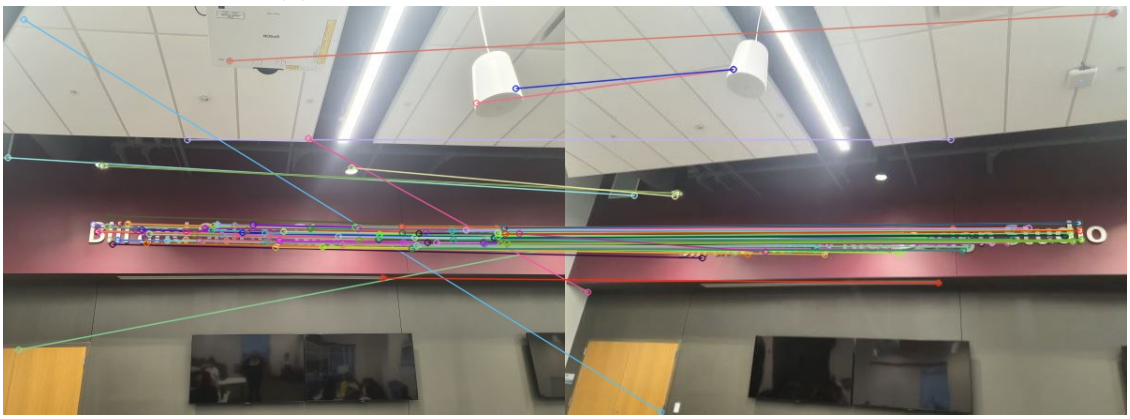
Figure 37: Interest Point Correspondence for $\sigma = 1.6$

Task 1.1.2: SIFT Algorithm

Output



(a) Interest Point Correspondence for Lounge



(b) Interest Point Correspondence for Studio

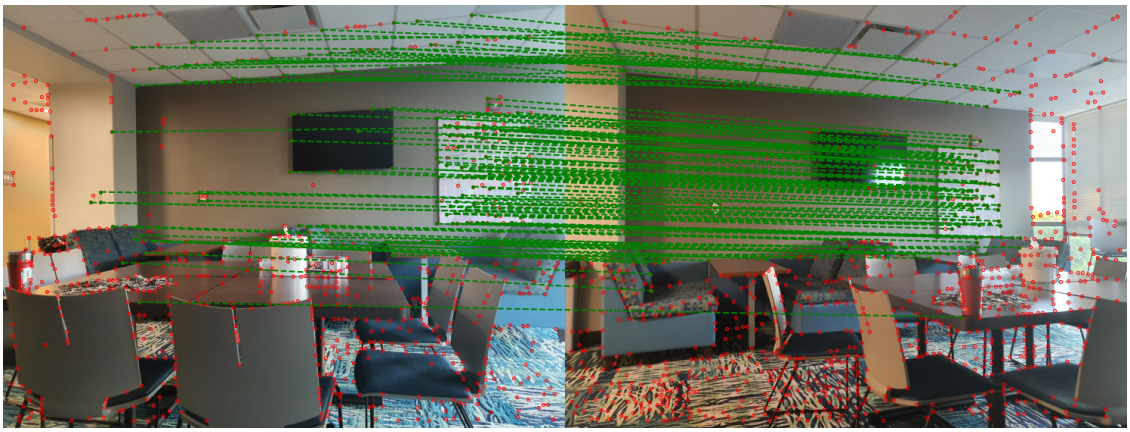
Figure 38: Interest Point Correspondence using SIFT

Observation

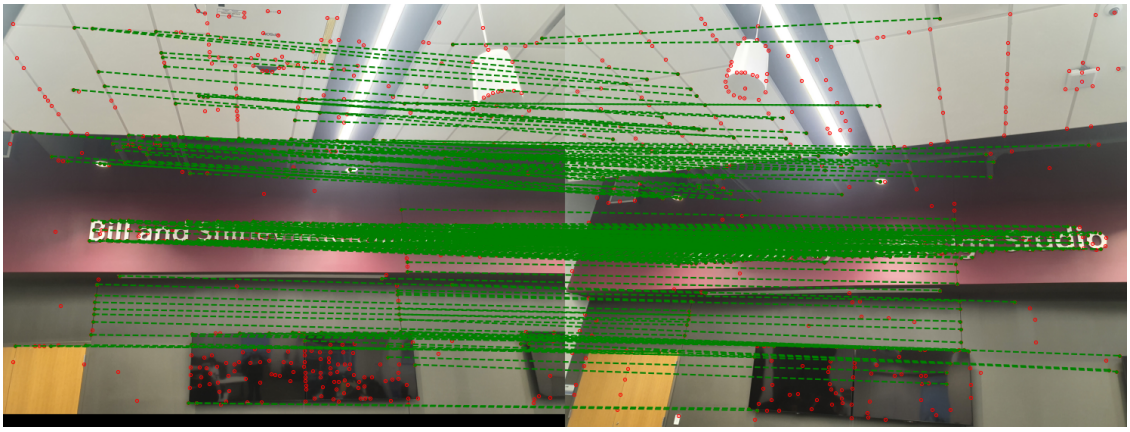
As expected, SIFT gives a much better correspondence between points. This is because of the Brute Force Matcher algorithm that SIFT algorithm in OpenCV uses to find correspondences. Again, for better visual representation of the matching, only the first 100 correspondences were drawn on the images.

Task 1.2: SuperPoint and SuperGlue

Output



(a) Interest Point Correspondences from SuperGlue



(b) Interest Point Correspondences From SuperGlue

Figure 39: Interest Point Correspondences using SuperPoint and SuperGlue

Observation

SuperPoint and SuperGlue gave the best results.

Code

```
import numpy as np
import cv2 as cv
import math
from random import choice

#-------------------------Functions-------------------------------#
def HaarMat(sigma):
    M = math.ceil(4*sigma) + (math.ceil(4*sigma)%2==1)
    kx = np.ones((M,M))
    ky = np.ones((M,M))
    kx[:,:M//2] = -1
    ky[M//2:,:] = -1
    return kx,ky

def Harris(img_raw, sigma):
    if len(img_raw.shape) == 3:
        img = cv.cvtColor(img_raw,cv.COLOR_BGR2GRAY)
    else:
        img = img_raw
    img = img.astype(np.float64)
    img -= np.min(img)
    img /= np.max(img)

    # Get kernel kx and ky to estimate dx and dy
    kx, ky = HaarMat(sigma)
    dx = cv.filter2D(img, -1, kernel = kx)
    dy = cv.filter2D(img, -1, kernel = ky)

    # Create the 5sigma neighborhood
    N = 2*int((5*sigma)//2) + 1
    k_N = np.ones((N,N))
    # Create the C matrix
    dx_dx = dx * dx
    dy_dy = dy * dy
    dx_dy = dx * dy

    s_dxdx = cv.filter2D(dx_dx, -1, kernel = k_N)
    s_dydy = cv.filter2D(dy_dy, -1, kernel = k_N)
    s_dxdy = cv.filter2D(dx_dy, -1, kernel = k_N)

    # find det(C) and Tr(C)
    det = s_dxdx*s_dydy - (s_dxdy*s_dxdy)
    tr = s_dxdx + s_dydy

    k_vals = det / (tr**2 + 0.001)
    k = np.sum(k_vals) / (img.shape[0]*img.shape[1])

    R = det - k * (tr**2)
    thres = np.sort(R, axis=None)[-int(0.05*len(R.flatten()))]

    # Non-maximum suppression to remove nonrelevant coords
```

```
    r = int(N/2)
    coords = []
    for y in range(r, img.shape[0]-r):
        for x in range(r, img.shape[1]-r):
            roi = R[y-r : y+r , x-r : x+r]
            if R[y,x] == np.amax(roi) and np.amax(roi) >= thres and R[y,x]
                                                    >0:
                coords.append([x,y])
    return coords

def drawCircles(img, corners):
    img_copy = img.copy()
    for corner in corners:
        x,y = corner
        cv.circle(img_copy, (x,y), 2, (10,240,10), -1)
    return img_copy

def getDistance(img1, img2, coord1, coord2, metric):
    M = math.ceil(4*sigma) + (math.ceil(4*sigma)%2==1)
    x1,y1 = coord1
    x2,y2 = coord2

    r = min(x1, y1, img1.shape[1] - x1, img1.shape[0]-y1, x2, y2, img2.
                                        shape[1] - x2, img2.shape[0]-y2,
                                        M+1)
    reg1 = img1[y1-r : y1+r, x1-r : x1+r]
    reg2 = img2[y2-r : y2+r, x2-r : x2+r]

    if metric == 'SSD':
        d = np.sum(np.square(reg1-reg2))
    elif metric == 'NCC':
        m1 = np.mean(reg1)
        m2 = np.mean(reg2)
        d = 1 - np.sum((reg1-m1)*(reg2-m2))/(np.sqrt(np.sum(np.square(reg1
                                            -m1))*np.sum(np.square(reg2-
                                            m2))))
    return d

def getCorresp(img1_raw, img2_raw, corner_coords1, corner_coords2, sigma,
                                        metric):
    if len(img1_raw.shape) == 3:
        img1 = cv.cvtColor(img1_raw,cv.COLOR_BGR2GRAY)
    else:
        img1 = img1_raw
    img1 = img1.astype(np.float64)
    img1 -= np.min(img1)
    img1 /= np.max(img1)

    if len(img2_raw.shape) == 3:
        img2 = cv.cvtColor(img2_raw,cv.COLOR_BGR2GRAY)
    else:
        img2 = img2_raw
    img2 = img2.astype(np.float64)
    img2 -= np.min(img2)
```

```
    img2 /= np.max(img2)

    corrs = []
    if len(corner_coords1) <= len(corner_coords2):
        for coord1 in corner_coords1:
            bestP = 0
            bestD = float('inf')
            for coord2 in corner_coords2:
                d = getDistance(img1, img2, coord1, coord2, metric)
                if d < bestD:
                    bestD = d
                    bestP = coord2
            corrs.append((coord1, bestP, bestD))
    else:
        for coord2 in corner_coords2:
            bestP = 0
            bestD = float('inf')
            for coord1 in corner_coords1:
                d = getDistance(img1, img2, coord1, coord2, metric)
                if d < bestD:
                    bestD = d
                    bestP = coord1
            corrs.append((bestP, coord2, bestD))

    return corrs

def drawCorrs(img1_raw, img2_raw, corrs, N):
    # resize images:
    h1 = img1_raw.shape[0]
    w1 = img1_raw.shape[1]
    h2 = img2_raw.shape[0]
    w2 = img2_raw.shape[1]
    if h1>h2:
        img1_raw = cv.resize(img1_raw, (w2,h2), cv.INTER_AREA)
    else:
        img2_raw = cv.resize(img2_raw, (w1,h1), cv.INTER_AREA)

    # Join the images next to each other to show correspondence
    comb_img = np.concatenate((img1_raw, img2_raw), axis=1)
    width = img1_raw.shape[1]

    # Show N first correspondences
    if N == 0:
        N = len(corrs)
    for i in range(N):
        x1,y1 = corrs[i][0]
        x2,y2 = corrs[i][1]
        x2 = x2 + width
        cv.circle(comb_img, (x1,y1), 3, (10, 240, 10), -1)
        cv.circle(comb_img, (x2,y2), 3, (240, 240,10), -1)
        cv.line(comb_img, (x1,y1), (x2,y2), (10, 240,240), 1)
    return comb_img

def sift(img1_raw, img2_raw):
```

```
    if len(img1_raw.shape) == 3:
        img1 = cv.cvtColor(img1_raw,cv.COLOR_BGR2GRAY)
    else:
        img1 = img1_raw

    if len(img2_raw.shape) == 3:
        img2 = cv.cvtColor(img2_raw,cv.COLOR_BGR2GRAY)
    else:
        img2 = img2_raw

    corrs = []

    # Create sift detector
    sift_detector = cv.SIFT_create()

    # find key points and descriptors with Sift
    kp1, des1 = sift_detector.detectAndCompute(img1,None)
    kp2, des2 = sift_detector.detectAndCompute(img2,None)

    # BFMatcher
    bf = cv.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    # Apply ratio test
    corrs = []
    for m,n in matches:
        if m.distance < 0.75*n.distance:
            corrs.append([m])
    comb_img = np.concatenate((img1_raw, img2_raw), axis=1)
    cv.drawMatchesKnn(img1_raw, kp1, img2_raw, kp2, corrs[0:150], comb_img
                                        ,flags=2)

    return comb_img


#-----------------------------------Inputs
                                    ---------------------------------#

# Input Images
book1_img = cv.imread('HW4/books_1.jpeg')
book2_img = cv.imread('HW4/books_2.jpeg')
fountain1_img = cv.imread('HW4/fountain_1.jpg')
fountain2_img = cv.imread('HW4/fountain_2.jpg')
lounge1_img = cv.imread('HW4/lounge1.jpg')
lounge2_img = cv.imread('HW4/lounge2.jpg')
studio1_img = cv.imread('HW4/studio1.jpg')
studio2_img = cv.imread('HW4/studio2.jpg')


# Input scale sigma
sigmas = [0.8, 1.2, 1.4, 1.6]

#--------------------Task 1.1: Harris Corner Detection
                                    --------------------#
```

```
for sigma in sigmas:
    print("Sigma:", sigma)

    #----------------1st Pair--------------#
    # Find Interest Points
    book1_corners = Harris(book1_img, sigma)
    print("Interest points for books1:",len(book1_corners))
    book1_corners_img = drawCircles(book1_img, book1_corners)

    book2_corners = Harris(book2_img, sigma)
    print("Interest points for books2:",len(book2_corners))
    book2_corners_img = drawCircles(book2_img, book2_corners)

    # Get Correspondence using SSD
    book_corrs_SSD = getCorresp(book1_img, book2_img, book1_corners,
                                book2_corners, sigma, 'SSD')
    book_corrs_SSD_img = drawCorrs(book1_img, book2_img, book_corrs_SSD,
                                100)

    # Get Correspondence using NCC
    book_corrs_NCC = getCorresp(book1_img, book2_img, book1_corners,
                                book2_corners, sigma, 'NCC')
    book_corrs_NCC_img = drawCorrs(book1_img, book2_img, book_corrs_NCC,
                                100)

    # Save images
    cv.imwrite('HW4/book1_'+str(sigma)+'.jpeg', book1_corners_img)
    cv.imwrite('HW4/book2_'+str(sigma)+'.jpeg', book2_corners_img)
    cv.imwrite('HW4/book_corrs_SSD'+str(sigma)+'.jpeg', book_corrs_SSD_img
                                )
    cv.imwrite('HW4/book_corrs_NCC'+str(sigma)+'.jpeg', book_corrs_NCC_img
                                )

    #----------------2nd Pair--------------#
    # Find Interest Points
    fountain1_corners = Harris(fountain1_img, sigma)
    print("Interest Points for Fountain:",len(fountain1_corners))
    fountain1_corners_img = drawCircles(fountain1_img, fountain1_corners)

    fountain2_corners = Harris(fountain2_img, sigma)
    print("Interest Points for Fountain2:",len(fountain2_corners))
    fountain2_corners_img = drawCircles(fountain2_img, fountain2_corners)

    # Get Correspondence using SSD
    fountain_corrs_SSD = getCorresp(fountain1_img, fountain2_img,
                                fountain1_corners,
                                fountain2_corners, sigma, 'SSD')
    fountain_corrs_SSD_img = drawCorrs(fountain1_img, fountain2_img,
                                fountain_corrs_SSD, 100)

    # Get Correspondence using NCC
    fountain_corrs_NCC = getCorresp(fountain1_img, fountain2_img,
                                fountain1_corners,
```

```
                                        fountain2_corners, sigma, 'NCC')
fountain_corrs_NCC_img = drawCorrs(fountain1_img, fountain2_img,
                                        fountain_corrs_NCC, 100)

# Save Images
cv.imwrite('HW4/fountain1_'+str(sigma)+'.jpeg', fountain1_corners_img)
cv.imwrite('HW4/fountain2_'+str(sigma)+'.jpeg', fountain2_corners_img)
cv.imwrite('HW4/fountain_corrs_SSD'+str(sigma)+'.jpeg',
                                        fountain_corrs_SSD_img)
cv.imwrite('HW4/fountain_corrs_NCC'+str(sigma)+'.jpeg',
                                        fountain_corrs_NCC_img)

#----------------3rd Pair--------------#
# Find Interest Points
lounge1_corners = Harris(lounge1_img, sigma)
print("Interest points for lounge1:",len(lounge1_corners))
lounge1_corners_img = drawCircles(lounge1_img, lounge1_corners)

lounge2_corners = Harris(lounge2_img, sigma)
print("Interest points for lounge2:",len(lounge2_corners))
lounge2_corners_img = drawCircles(lounge2_img, lounge2_corners)

# Get Correspondence using SSD
lounge_corrs_SSD = getCorresp(lounge1_img, lounge2_img,
                                        lounge1_corners, lounge2_corners,
                                         sigma, 'SSD')
lounge_corrs_SSD_img = drawCorrs(lounge1_img, lounge2_img,
                                        lounge_corrs_SSD, 100)

# Get Correspondence using NCC
lounge_corrs_NCC = getCorresp(lounge1_img, lounge2_img,
                                        lounge1_corners, lounge2_corners,
                                         sigma, 'NCC')
lounge_corrs_NCC_img = drawCorrs(lounge1_img, lounge2_img,
                                        lounge_corrs_NCC, 100)

# Save images
cv.imwrite('HW4/lounge1_'+str(sigma)+'.jpeg', lounge1_corners_img)
cv.imwrite('HW4/lounge2_'+str(sigma)+'.jpeg', lounge2_corners_img)
cv.imwrite('HW4/lounge_corrs_SSD'+str(sigma)+'.jpeg',
                                        lounge_corrs_SSD_img)
cv.imwrite('HW4/lounge_corrs_NCC'+str(sigma)+'.jpeg',
                                        lounge_corrs_NCC_img)

#----------------4th Pair--------------#
# Find Interest Points
studio1_corners = Harris(studio1_img, sigma)
print("Interest points for studio1:",len(studio1_corners))
studio1_corners_img = drawCircles(studio1_img, studio1_corners)

studio2_corners = Harris(studio2_img, sigma)
print("Interest points for studio2:",len(studio2_corners))
studio2_corners_img = drawCircles(studio2_img, studio2_corners)
```

```
    # Get Correspondence using SSD
    studio_corrs_SSD = getCorresp(studio1_img, studio2_img,
                                  studio1_corners, studio2_corners,
                                   sigma, 'SSD')
    studio_corrs_SSD_img = drawCorrs(studio1_img, studio2_img,
                                  studio_corrs_SSD, 0)

    # Get Correspondence using NCC
    studio_corrs_NCC = getCorresp(studio1_img, studio2_img,
                                  studio1_corners, studio2_corners,
                                   sigma, 'NCC')
    studio_corrs_NCC_img = drawCorrs(studio1_img, studio2_img,
                                  studio_corrs_NCC, 0)

    # Save images
    cv.imwrite('HW4/studio1_'+str(sigma)+'.jpeg', studio1_corners_img)
    cv.imwrite('HW4/studio2_'+str(sigma)+'.jpeg', studio2_corners_img)
    cv.imwrite('HW4/studio_corrs_SSD'+str(sigma)+'.jpeg',
                                  studio_corrs_SSD_img)
    cv.imwrite('HW4/studio_corrs_NCC'+str(sigma)+'.jpeg',
                                  studio_corrs_NCC_img)

# Get Interest points using SIFT
book_corrs_sift_img = sift(book1_img, book2_img)
cv.imwrite('HW4/book_corrs_Sift.jpeg', book_corrs_sift_img)

# Get Interest points using SIFT
fountain_corrs_sift_img = sift(fountain1_img, fountain2_img)
cv.imwrite('HW4/fountain_corrs_sift.jpeg', fountain_corrs_sift_img)

# Get Interest points using SIFT
lounge_corrs_sift_img = sift(lounge1_img, lounge2_img)
cv.imwrite('HW4/lounge_corrs_sift.jpeg', lounge_corrs_sift_img)

# Get Interest points using SIFT
studio_corrs_sift_img = sift(studio1_img, studio2_img)
cv.imwrite('HW4/studio_corrs_sift.jpeg', studio_corrs_sift_img)
```