FALL 22 ECE 661 – COMPUTER VISION

Homework 4

Sahithi Kodali – 34789866
kodali1@purdue.edu

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.

The Laplacian of Gaussian (LoG) of an image f(x,y) is given as ∇^2 ff (x, y, σ). According to the scale space theory for images, ff (x, y, σ) represents the σ smoothed version of an image f(x,y). This is given as,

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

In the above equation, it can be observed that the LoG of an image f(x,y) can be obtained by subtracting the $(\sigma + \Delta\sigma)$ smoothened version of image from (σ) smoothened image, which is nothing but the definition of Difference of Gaussian (DoG) of an image.

Further expanding the above theory,

$$\sigma\,\nabla^2\,\text{ff}\,(x, y, \sigma) = \frac{\partial}{\partial\sigma}\text{ff}\,(x, y, \sigma) \sim \frac{1}{\Delta\sigma}\,\text{ff}\,(\sigma + \Delta\sigma) - \text{ff}\,(x, y, \sigma)$$

$$\sim \frac{1}{k\sigma - \sigma}\,\text{ff}\,(\sigma + \Delta\sigma) - \text{ff}\,(x, y, \sigma)$$

$$(k - 1)\sigma^2\,\nabla^2\,\text{ff}\,(x, y, \sigma) \sim \text{ff}\,(\sigma + \Delta\sigma) - \text{ff}\,(x, y, \sigma)$$

Since k-1 is a constant and hence does not affect the local extrema, the scale-normalized LoG can be computed using the Difference of Gaussian (DoG) as above.

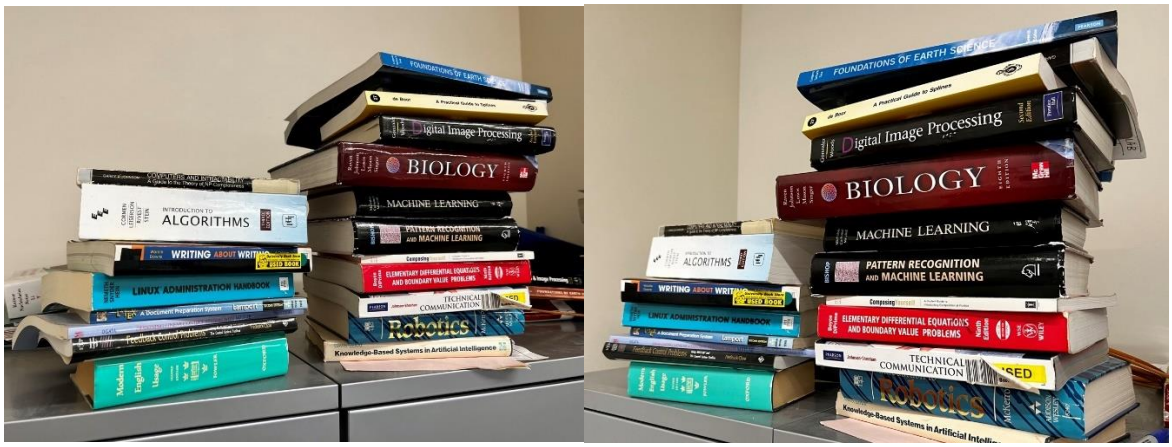
Why LoG of an image as a DoG is computationally more efficient?

This method of calculating the LoG of an image as a DoG is much more efficient than computing LoG. This is because, for estimating LoG we need larger kernels to perform convolution to find the second order partials for the image and can be carried out only as 2D convolution without a choice of separating them along axis directions.

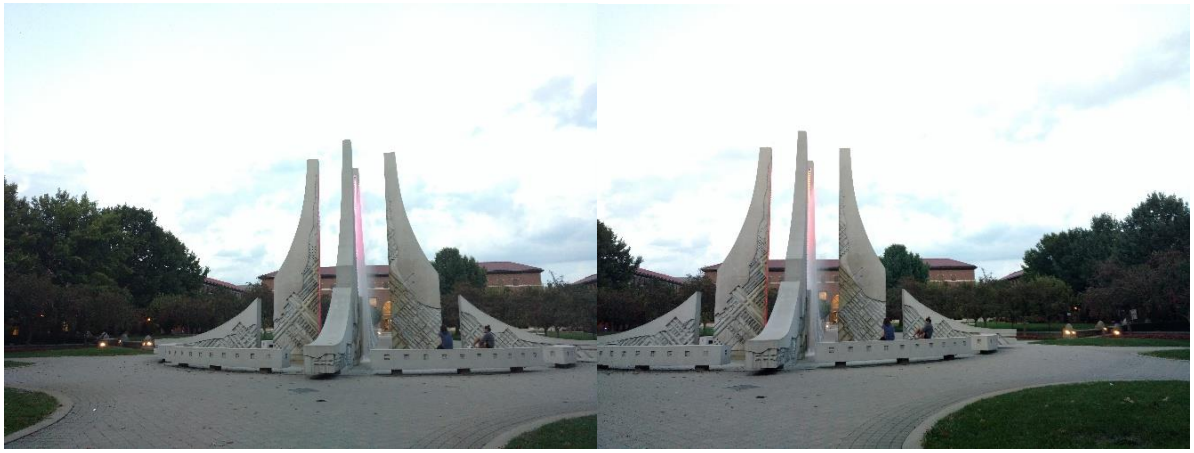
However, DoG requires smaller kernels for performing similar convolution at a same scale for the same image and hence requires much less computational power requirements. Adding to that, DoG allows the calculation of 1D convolution along x and y directions i.e., separable calculations. Since a 2D convolution for an image can be calculated using two 1D convolutions as well, it can be said that estimating the LoG of an image as a DoG is computationally much more efficient.

TASK – 1:

A pair of two images shown below are provided to perform interest points detection using Harris Corner Detection algorithm and SIFT/SURF implementation, followed by finding the point-to-point correspondences between the pair of images using SSD, NCC similarity metrics. Further, a GNN-based Superglue matching network is provided to test on these images.



(a) Input image 1 pairs



(b) Input image 2 pairs

1.1 Harris Corner Detector

The Harris Corner Detection algorithm is one of the popular interest point detector before SIFT and SURF were introduced. It is based on calculation of image intensity gradients along x and y directions to detect a corner. These gradients d_x and d_y can be calculated using Sobel filter or Haar wavelets at a scale σ. If N x N is said to be a Haar filter, N is calculated by determining the smaller Even integer larger than 4σ value, with a structure of $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ along y-direction and $[-1 \quad 1]$ along y-direction. The value of σ is chosen based on the extent to which the image needs to be smoothened to avoid the noise.

The gradients of the image tell a lot about the corner, say a single gradient being high indicates the presence of a single edge and if both the gradient are high and independent it indicates the presence of two non-parallel edges which can be a potential corner in the image. Once the

gradients are determined, a 5σ x 5σ window neighbourhood is computed at each pixel to find the correlation of gradients at a pixel's neighbourhood using the matrix C,

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

A threshold using the det and trace of matric C is computed to determine if the pixel has a corner known as response R,

$$R = \det(C) - k\,(\text{trace}(C))^2 , \quad \text{where the ratio } k = \det(C)/\text{trace}(C)^2$$

Since, there is a possibility of having multiple pixels at the same corner, a non-maximal suppression window is used to eliminate some pixels based on few threshold values on Rand the window. This window is an adaptive window based on the ratio of k, which changes according to the σ value chosen. The pixels detected as corner after the suppression are the interest points of the respective image.

Point-to-point correspondences matching using SSD and NCC

Once the interest points in the both the images pair are detected using the Harris algorithm above, these points need to be matched with the respective corresponding points. Since there are no feature descriptors for these points, a 21 x 21 window around the pixel is chosen as the feature descriptor. The distance using SSD and NCC distance metrics are calculated, where every point in the first image is compared to every point in the second and the points with the smallest distance are chosen as the corresponding interest points pair.

Squared Sum of Differences (SSD)

The SSD distance metric is computed as follows,

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

Normalized Cross-Correlation (NCC)

The NCC distance metric is computed as follows,

$$NCC = \frac{\sum_i \sum_j \left((f_1\,(i,j) - mean1)\,(f_2\,(i,j) - mean2)\right)}{\sqrt{[\,\sum_i \sum_j (f_1\,(i,j) - mean1)\text{^}2\,]\,[\,\sum_i \sum_j\ (f_2\,(i,j) - mean2)\text{^}2\,]}}$$

In the above stated metrics, f_1 relates to first image and f_2 related to second image and mean1, mean2 are the means of interest points in first and second image respectively. In both the metrics the smallest distance is considered to map the corresponding points and a distance threshold is also set to consider the matching validity between points which can be seen in the implementation in source code attached.

Results:

Outputs for the pair of images in (a):

At $\sigma = 0.3$:

Figure 1 Output showing corners using Harris at $\sigma = 0.3$

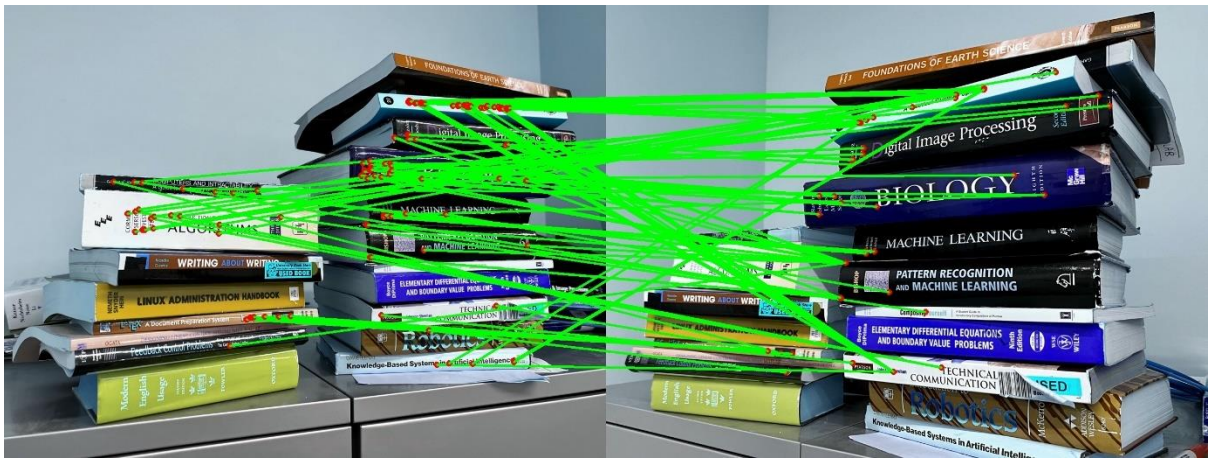


Figure 2 Output correspondences using Harris and SSD metric at $\sigma = 0.3$

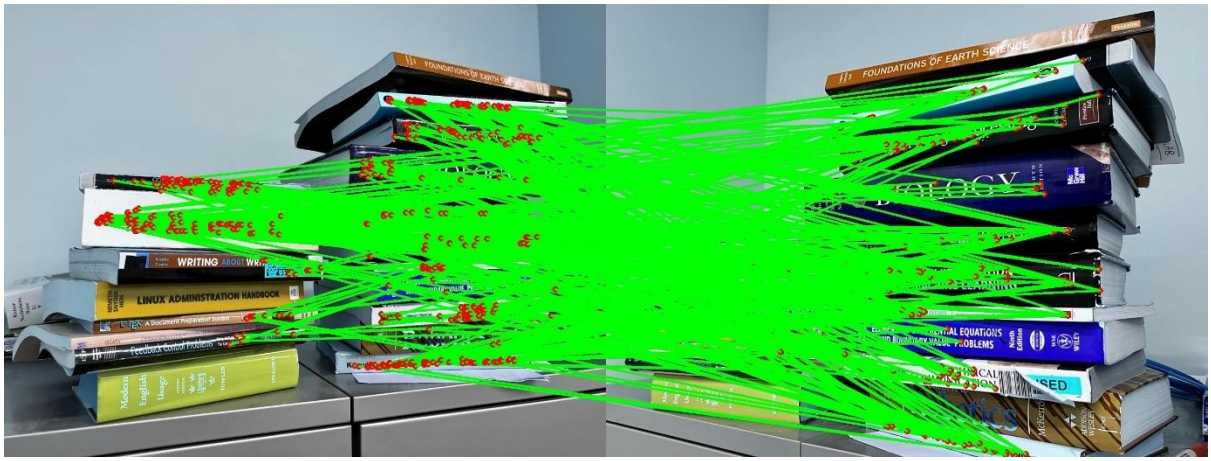


Figure 3 Output correspondences using Harris and NCC metric at $\sigma = 0.3$

At $\sigma = 0.6$*:*

Figure 4 Output showing corners using Harris at $\sigma = 0.6$

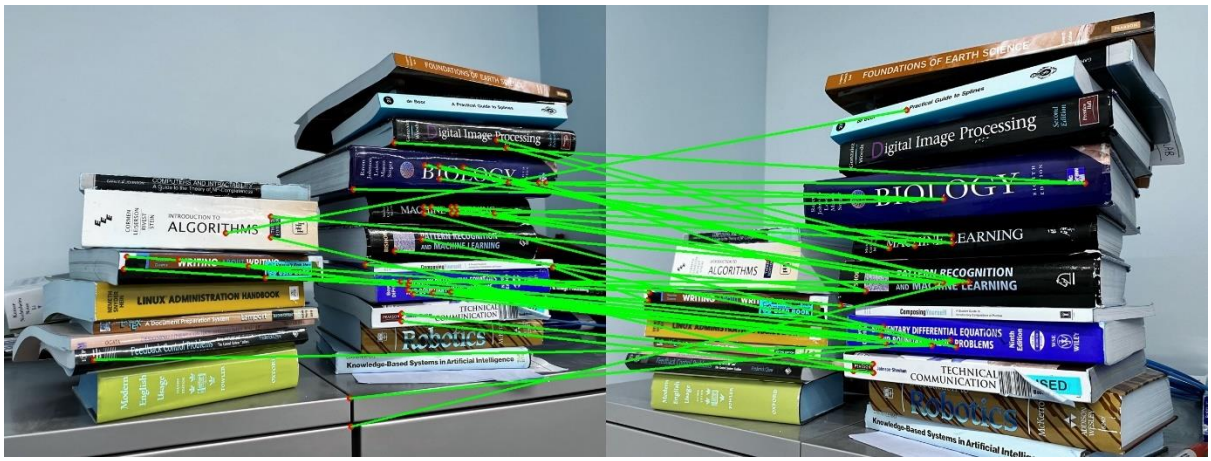


Figure 5 Output correspondences using Harris and SSD metric at $\sigma = 0.6$

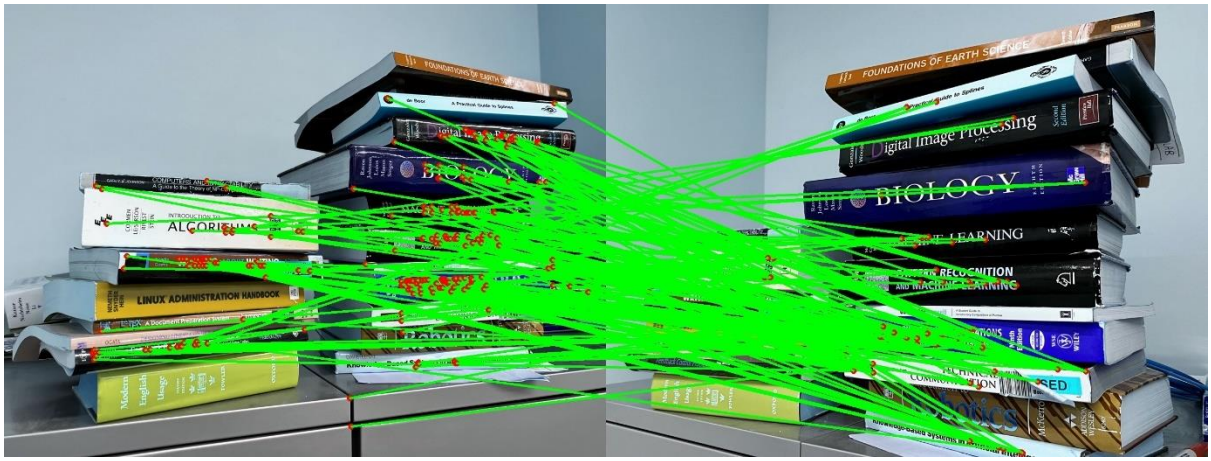


Figure 6 Output correspondences using Harris and NCC metric at $\sigma = 0.6$

<u>*At $\sigma = 1.2$:*</u>

Figure 7 Output showing corners using Harris at $\sigma = 1.2$

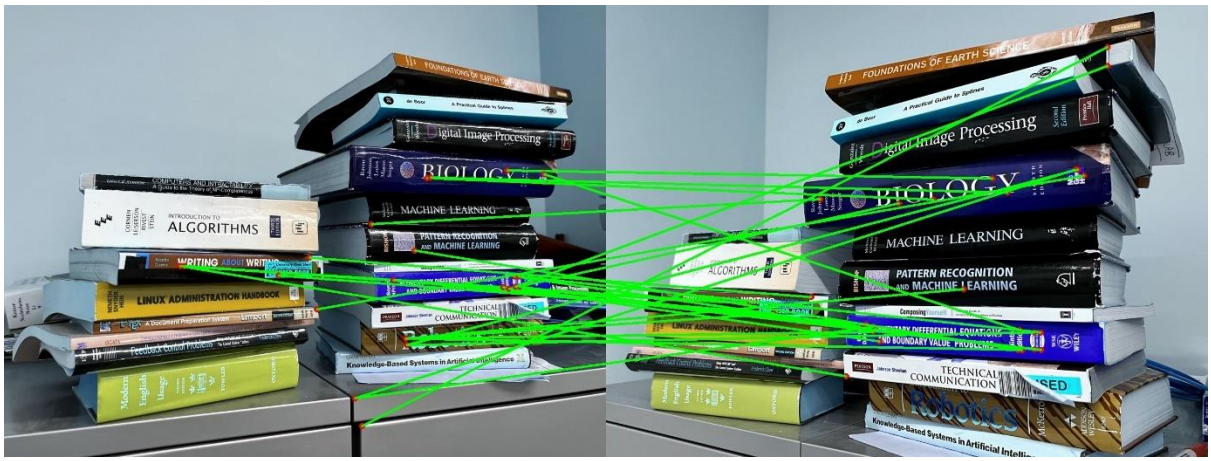


Figure 8 Output correspondences using Harris and SSD metric at $\sigma = 1.2$

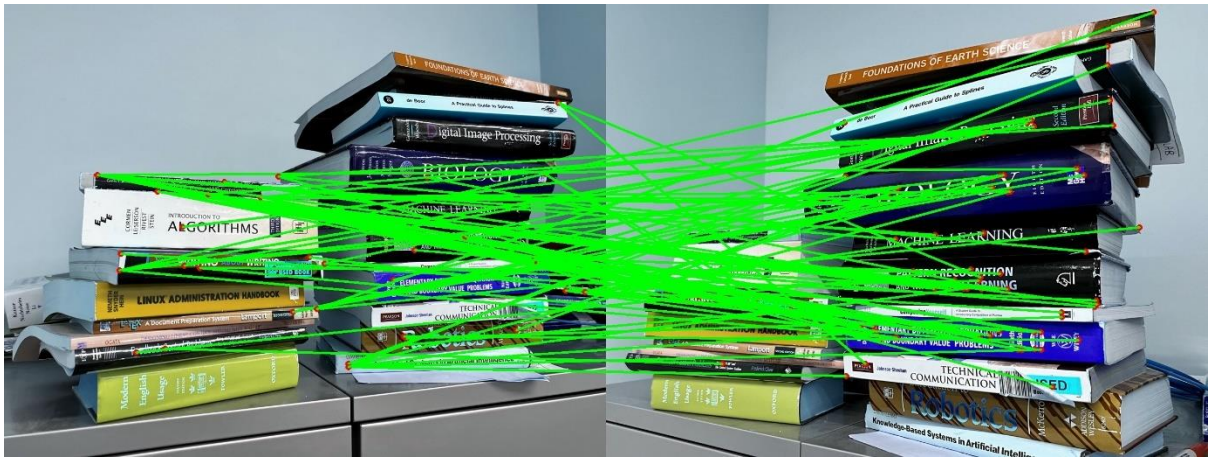


Figure 9 Output correspondences using Harris and NCC metric at $\sigma = 1.2$

At $\sigma = 2.4$:

Figure 10 Output showing corners using Harris at $\sigma = 2.4$

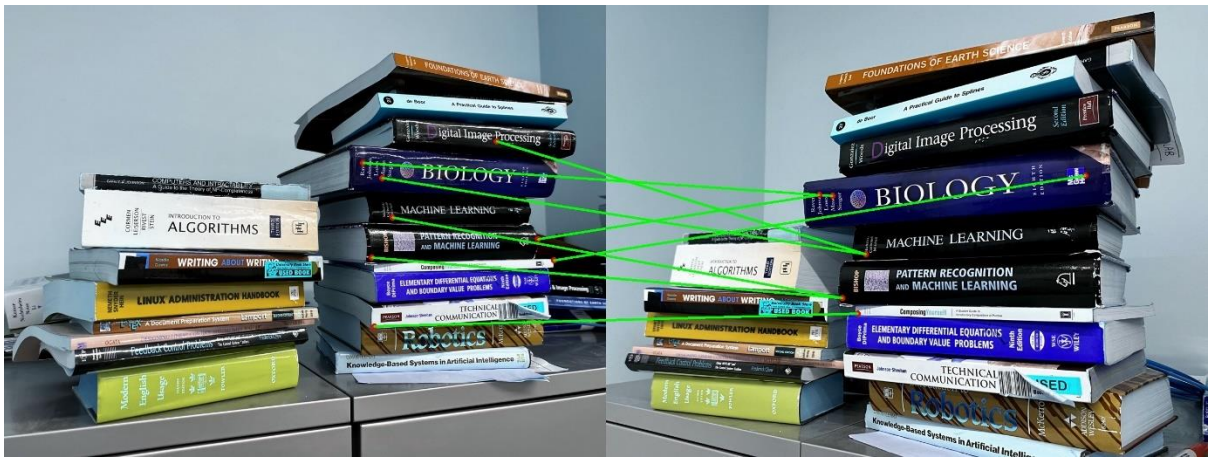


Figure 11 Output correspondences using Harris and SSD metric at $\sigma = 2.4$

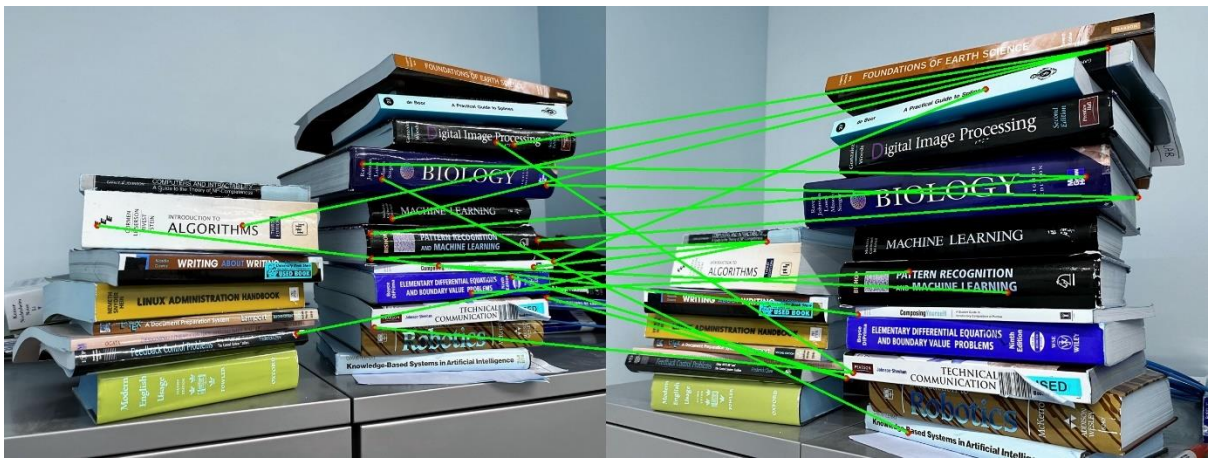


Figure 12 Output correspondences using Harris and NCC metric at $\sigma = 2.4$

Outputs for the pair of images in (b):

At $\sigma = 0.3$:

Figure 13 Output showing corners using Harris at $\sigma = 0.3$

Figure 14 Output correspondences using Harris and SSD metric at $\sigma = 0.3$

Figure 15 Output correspondences using Harris and NCC metric at $\sigma = 0.3$

At $\sigma = 0.6$:

Figure 16 Output showing corners using Harris at $\sigma = 0.6$

Figure 17 Output correspondences using Harris and SSD metric at $\sigma = 0.6$

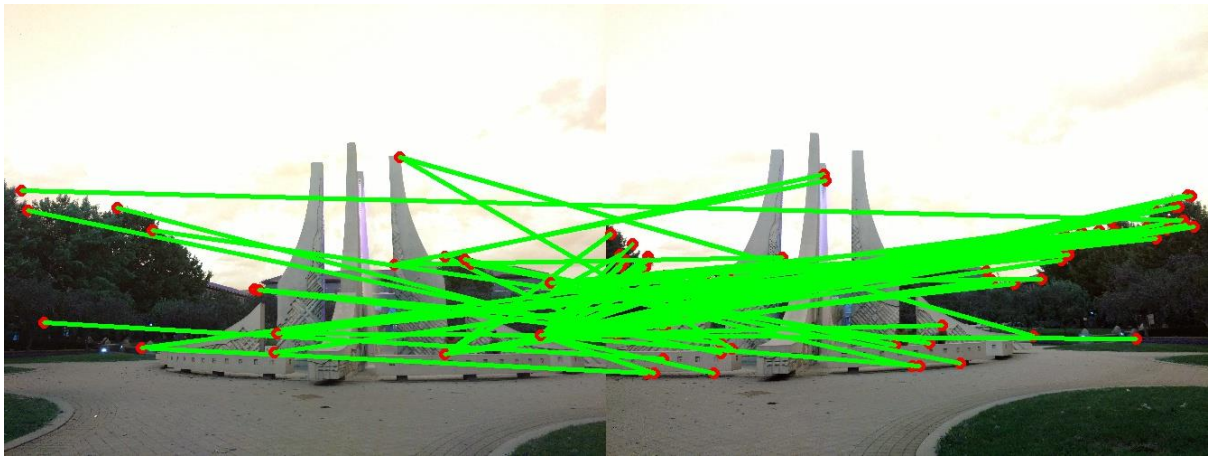


Figure 18 Output correspondences using Harris and NCC metric at $\sigma = 0.6$

At $\sigma = 1.2$:

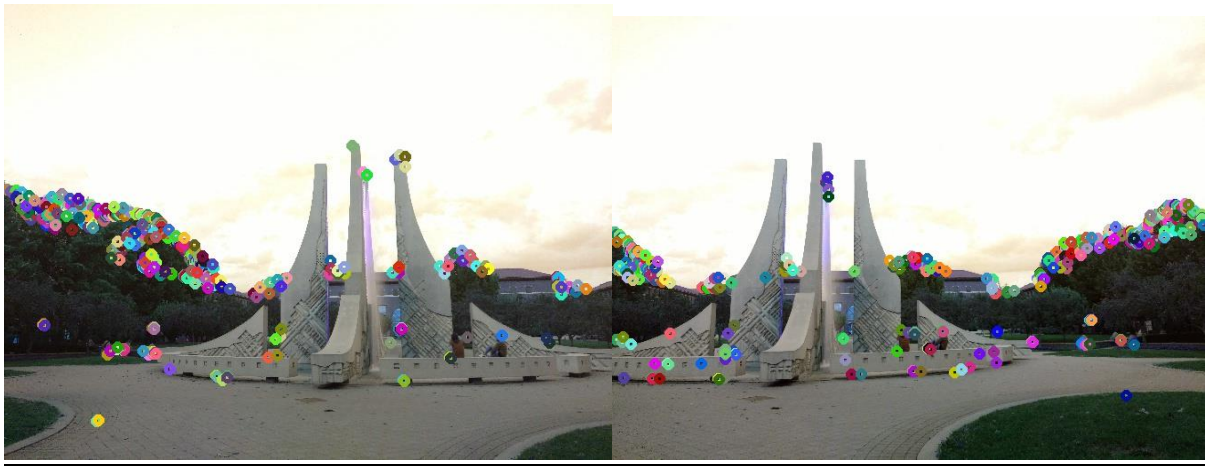


Figure 19 Output showing corners using Harris at $\sigma = 1.2$

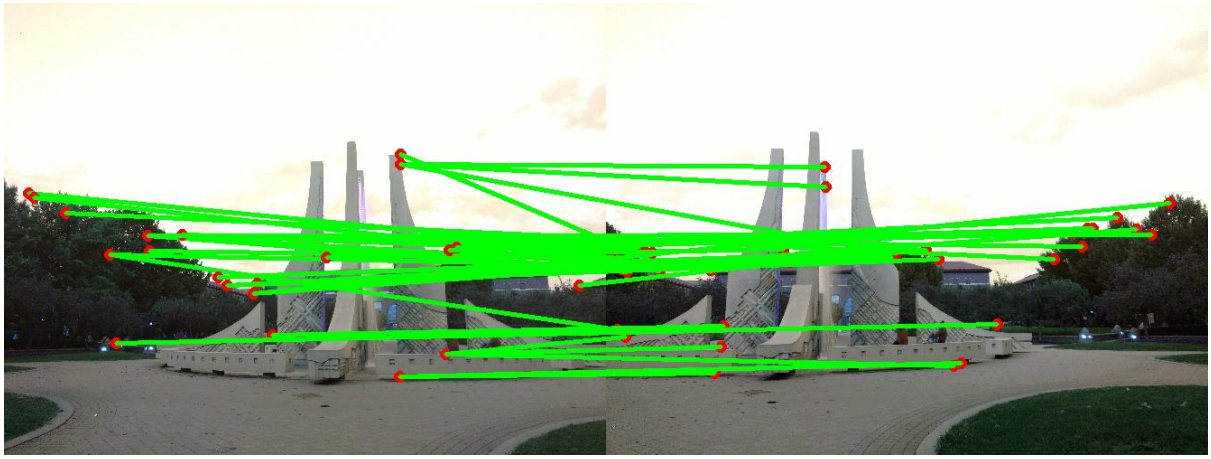


Figure 20 Output correspondences using Harris and SSD metric at $\sigma = 1.2$

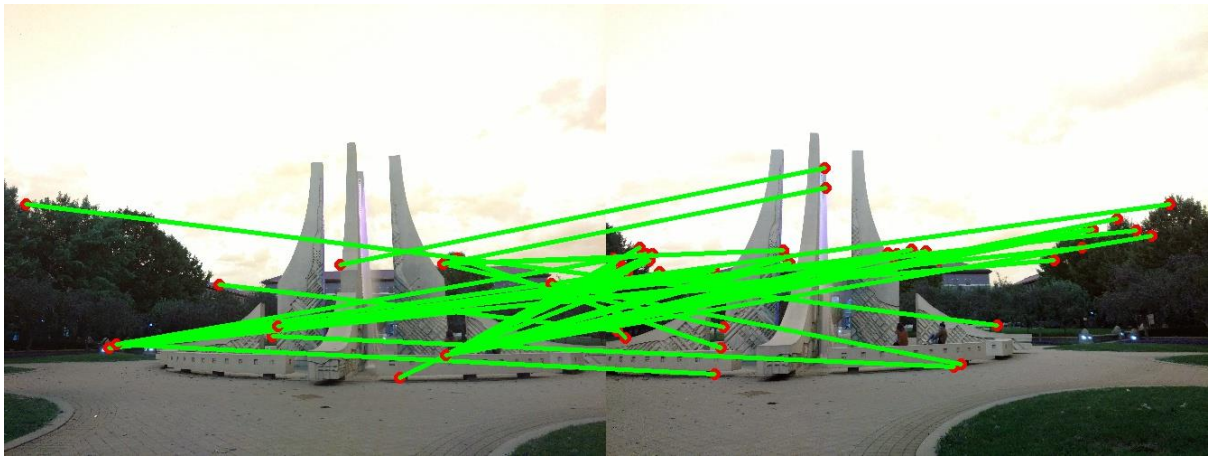


Figure 21 Output correspondences using Harris and NCC metric at $\sigma = 1.2$

At $\sigma = 2.4$*:*

Figure 22 Output showing corners using Harris at $\sigma = 2.4$

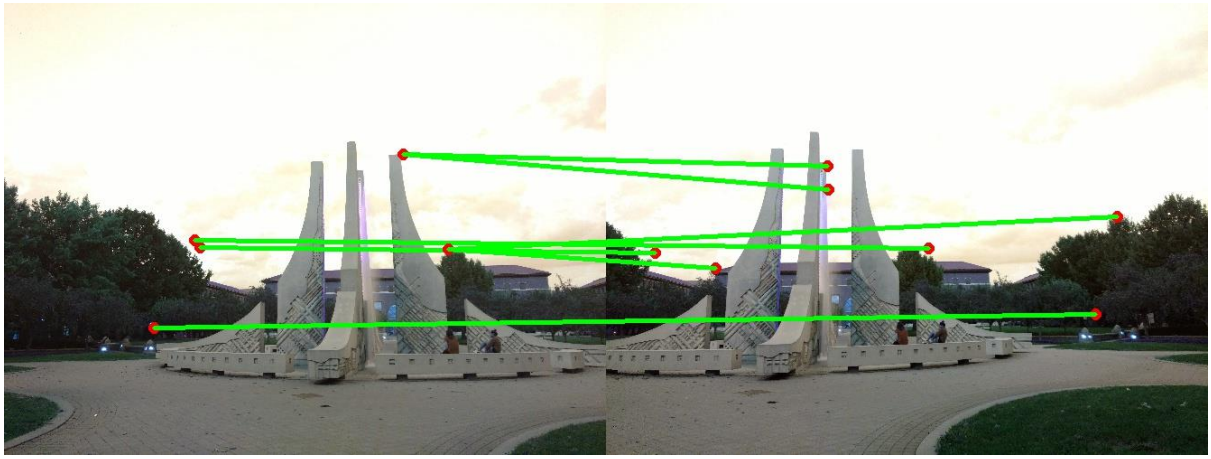


Figure 23 Output correspondences using Harris and SSD metric at $\sigma = 2.4$

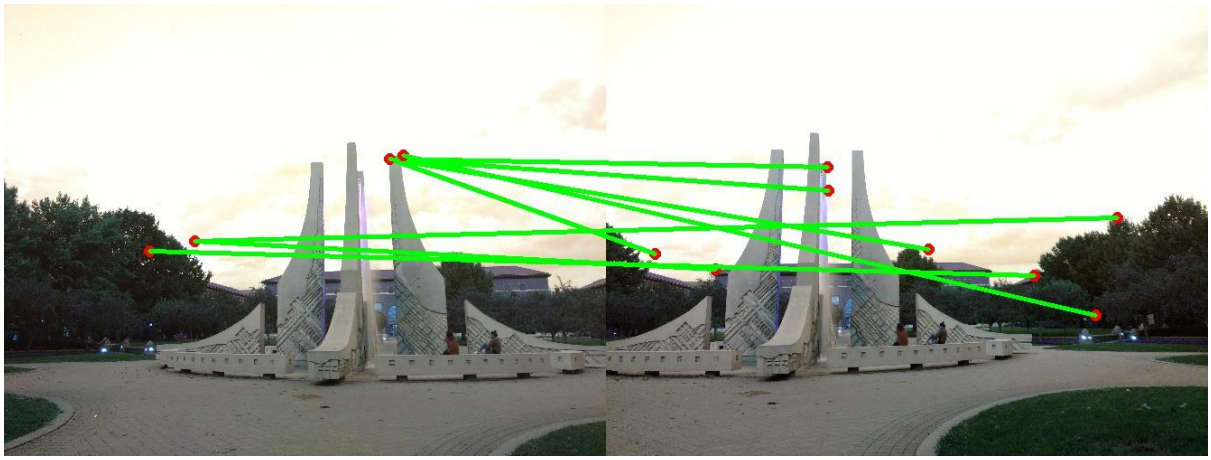


Figure 24 Output correspondences using Harris and NCC metric at $\sigma = 2.4$

Observations:

We can observe that as the sigma scale increases, the matching pair of points decrease. This is due to the smoothening of image as discussed above to reduce the noise in the image. We can

also observe that the bias in the first input image (books) reduces the capacity of Harris detector to match points accurately compared to the fountain image which has less variations.

1.2 SIFT Algorithm

Scale-Invariant Feature Transform (SIFT) algorithm is one of the most popular algorithm used for interest pair matching. In this homework it is implemented using OpenCV directly, but the algorithm and steps beneath in briefly discussed below.

1.) DoG Pyramid

A Difference of Gaussian (DoG) pyramid is constructed based on the σ value and the Gaussian kernel, by computing the DoG values D (x,y, σ) at (x,y).

2.) Local extrema

All the local extrema in the DoG pyramid are computed. Each pixel is compared to (a) 8 points in the immediate 3x3 neighbourhood (b) 9 points in the 3x3 neighbourhood in the next level up scale scape (c) 9 points in the 3x3 neighbourhood in the level just below.

3.) Accuracy of extreme points

As the scale σ increases, it is said that the extrema are not accurate due to coarseness being increased in the image. The true/accurate extrema are determined using Taylor series expansion of D(x, y, σ) or D($x0$) involving Hessian (H(x0)) and gradients (J(x0)), according to which the true location of the extremum is given as $x = -H^{-1}(xo).J(xo)$.

4.) Weak extrema Thresholding

Once the extrema are determined, the weak extrema are weeded out by thresholding $|D(x)| < 0.03$ typically to remove the weak interest points.

5.) Dominant local orientation and SIFT descriptor

Finally, a dominant local orientation is associated with each extremum and a descriptor is associated to each extremum. SIFT descriptor is a 128- dimensional vector that is assigned to every extremum in the DoG pyramid which indicates the consideration of an interest point and matching the interest point pairs from these descriptors.

The SIFT algorithm is implemented using OpenCV SIFT techniques and the matching of the pairs are performed using BF Matcher (Brute-force matcher) that is based on the Euclidean distance measure between the feature descriptors of the images.

https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/

Results:

Figure 25 Output correspondences of input pair of images in (a) using SIFT algorithm

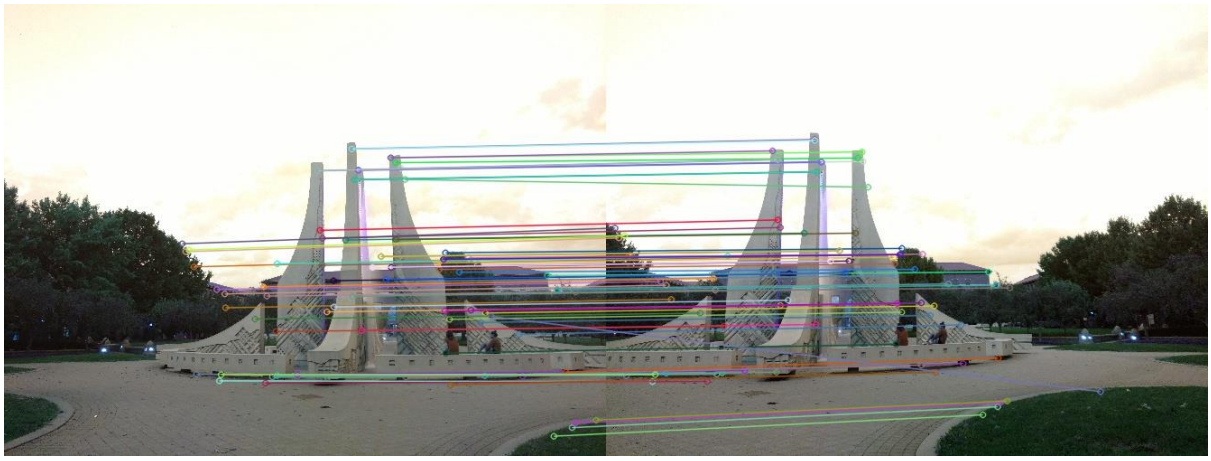


Figure 26 Output correspondences of input pair of images in (b) using SIFT algorithm

Observations:

We can observe that the SIFT algorithm performed very efficiently compared to the Harris detector showing its dominance in finding better matching points. However, we can still observe that the first input image books do not have as efficient matching points as the second image fountains with less variations.

1.3 SuperPoint and SuperGlue

This is a Deep learning based i.e., a neural network-based interest point matching algorithm that is provided in the homework to test its working and running on the images to compare with the above classical approaches.

Running the command: *python superglue_ece661.py <img1> <img2> <output_folder>* in the appropriate environment (according to the given execution instructions), initiates the model and returns the outputs for each pair of images as shown below.

Results:

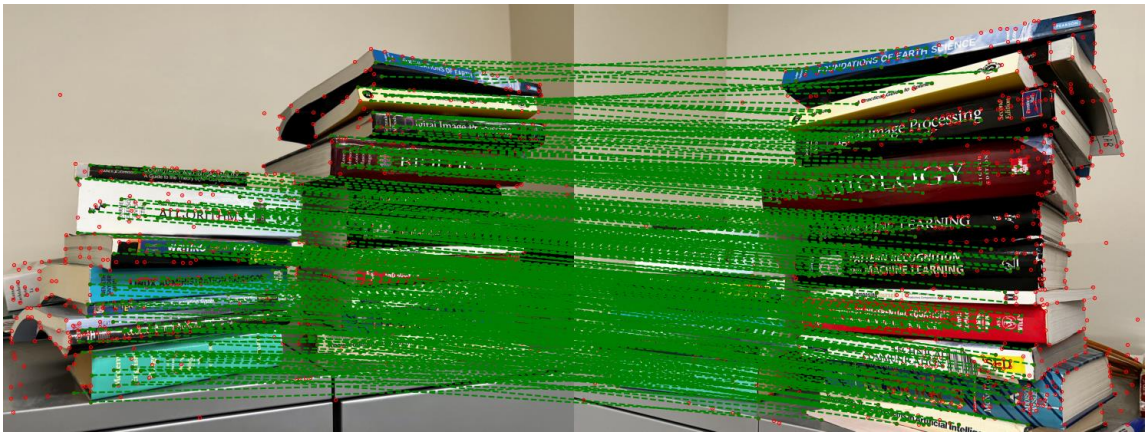


Figure 27 Output correspondences of input pair of images in (a) using SuperPoint + SuperGlue

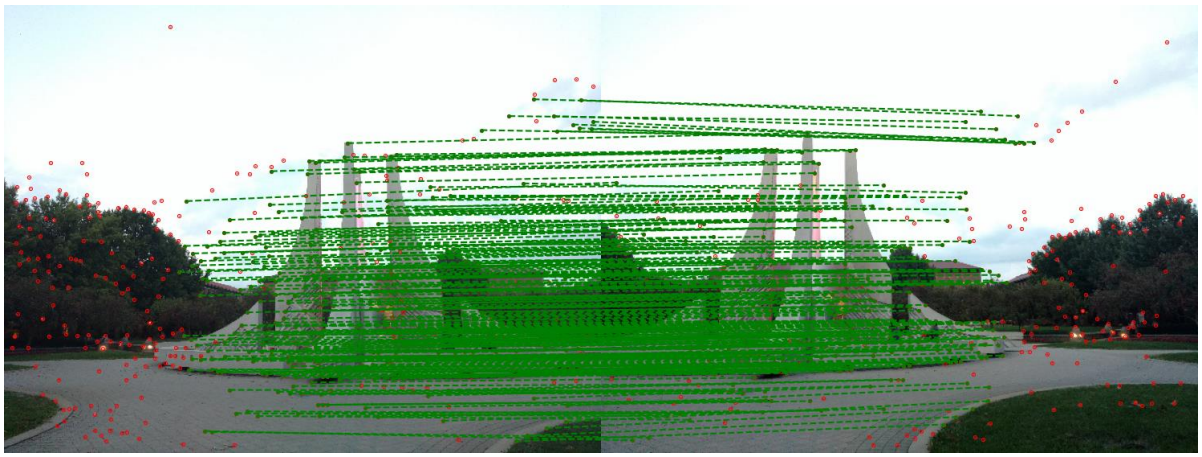


Figure 28 Output correspondences of input pair of images in (b) using SuperPoint + SuperGlue

Observations:

Comparing this super point neural network algorithm with the above two classic algorithms, we can definitely say that the neural network output is very accurate. The points are matched in a much better fashion showing the power of neural networks in finding accurate matching points.

TASK – 2:

This task involves implementing the same methods performed in Task 1 with two of our own images shown below.

(c) Own image 1 pairs

(d) Own image 2 pairs

2.1 Harris Corner Detector

Outputs for the pair of images in (c):

At $\sigma = 0.3$:

Figure 29 Output showing corners using Harris at $\sigma = 0.3$

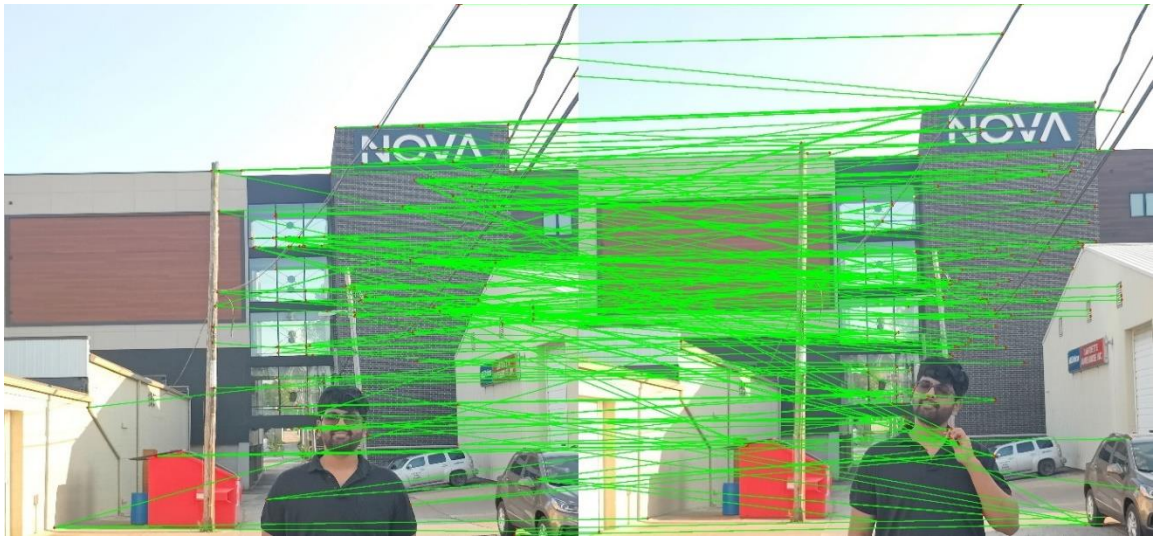


Figure 30 Output correspondences using Harris and SSD metric at $\sigma = 0.3$

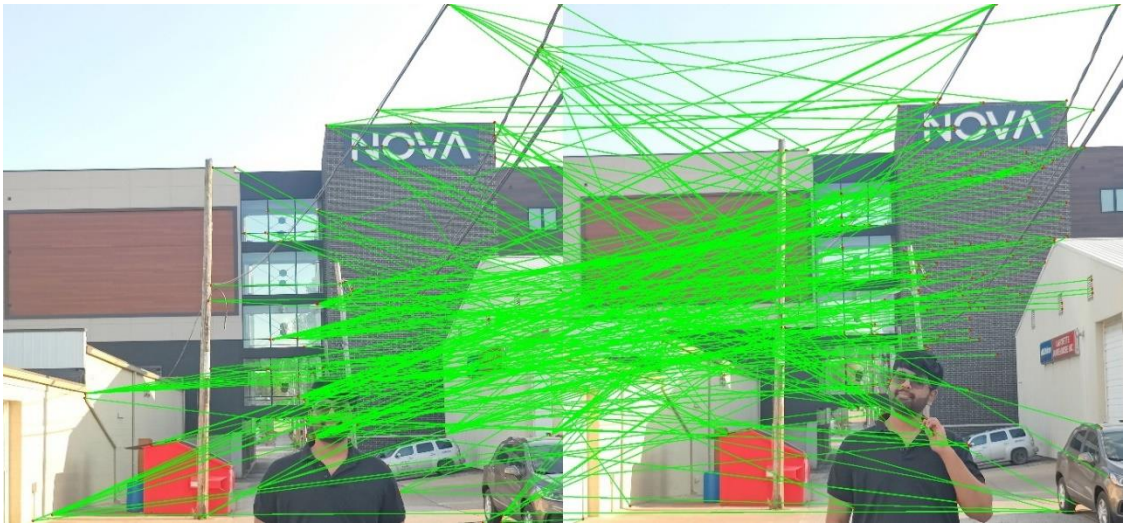


Figure 31 Output correspondences using Harris and NCC metric at $\sigma = 0.3$

At $\sigma = 0.6$:

Figure 32 Output showing corners using Harris at $\sigma = 0.6$

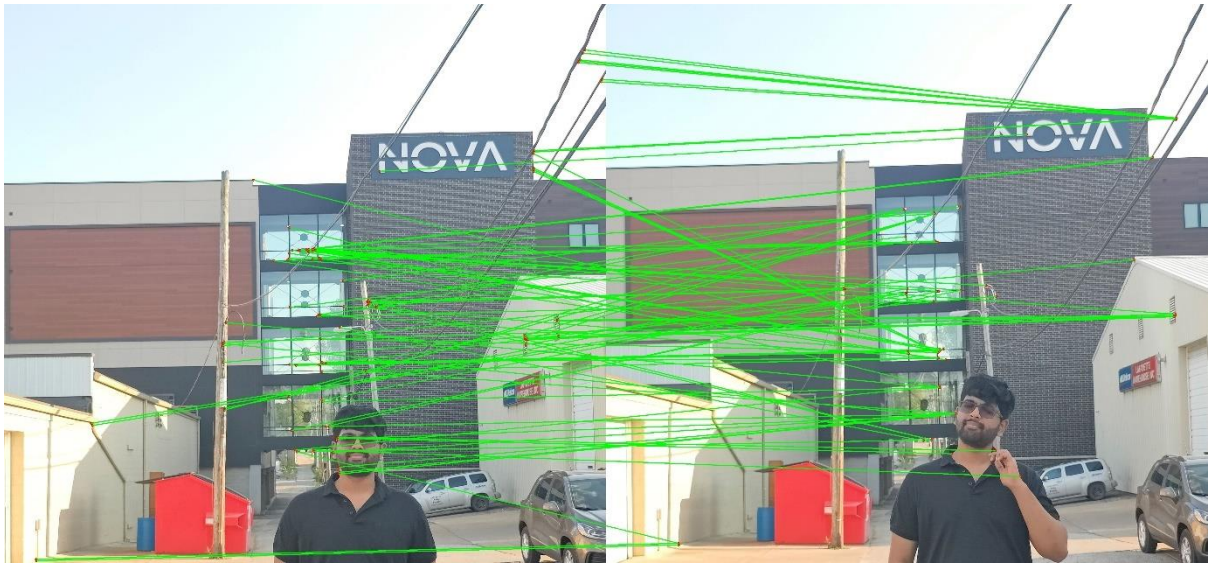


Figure 33 Output correspondences using Harris and SSD metric at $\sigma = 0.6$

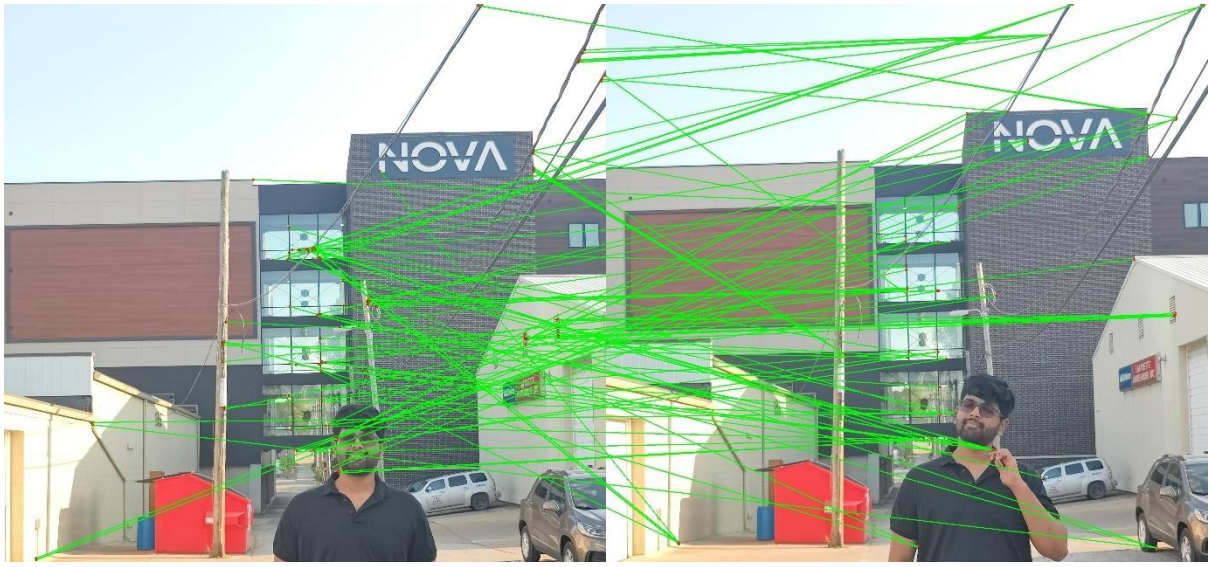


Figure 34 Output correspondences using Harris and NCC metric at $\sigma = 0.6$

At $\sigma = 1.2$*:*

Figure 35 Output showing corners using Harris at $\sigma = 1.2$

Figure 36 Output correspondences using Harris and SSD metric at $\sigma = 1.2$

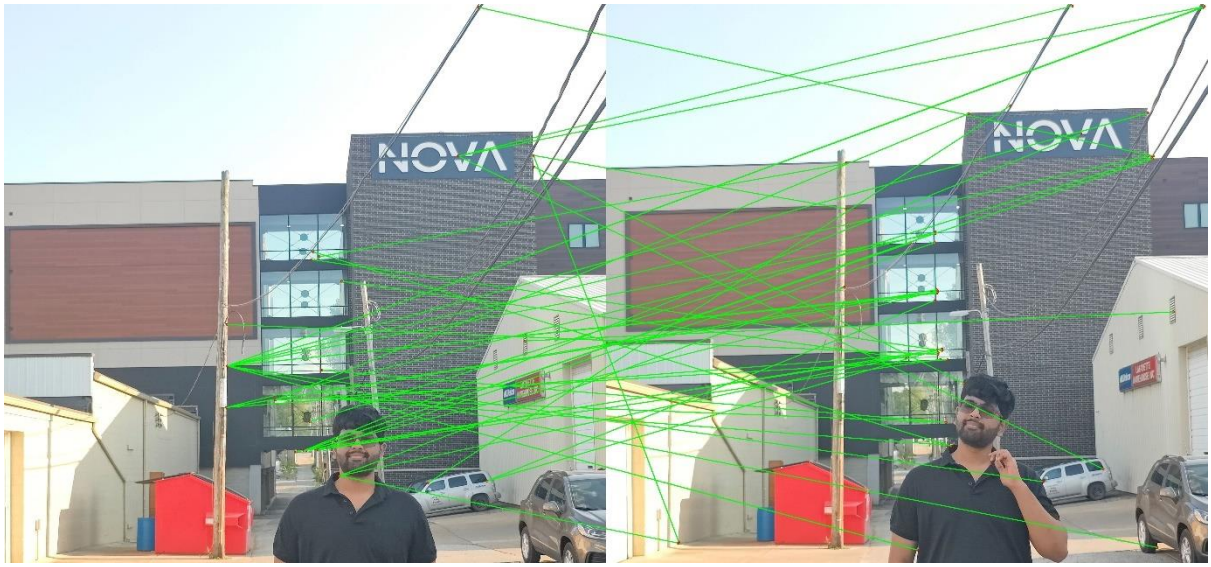


Figure 37 Output correspondences using Harris and NCC metric at $\sigma = 1.2$

At $\sigma = 2.4$*:*

Figure 38 Output showing corners using Harris at $\sigma = 2.4$

Figure 39 Output correspondences using Harris and SSD metric at $\sigma = 2.4$

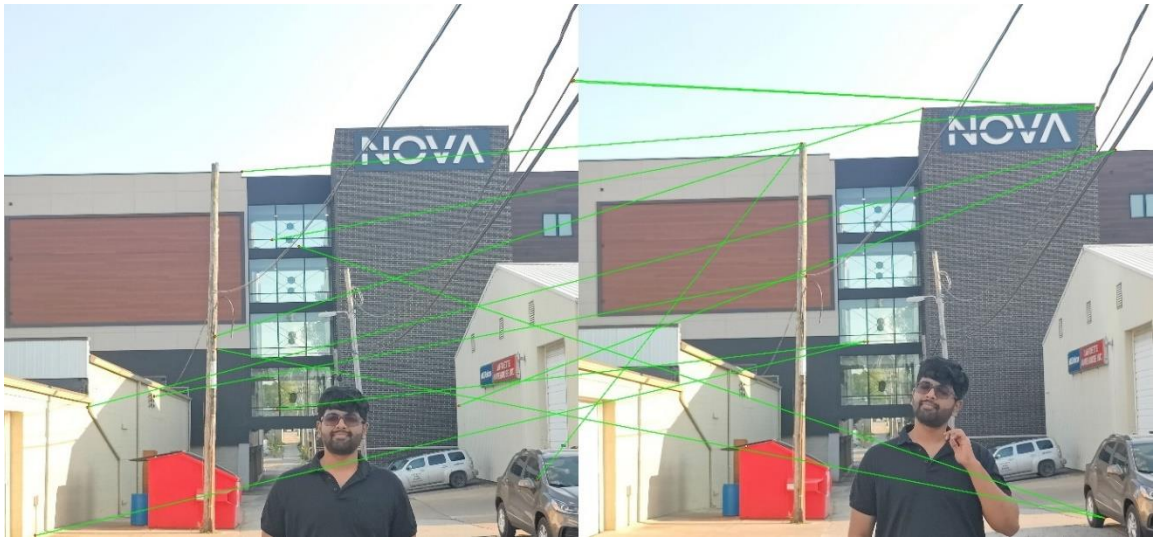


Figure 40 Output correspondences using Harris and NCC metric at $\sigma = 2.4$

Outputs for the pair of images in (d):

At $\sigma = 0.3$:

Figure 41 Output showing corners using Harris at $\sigma = 0.3$

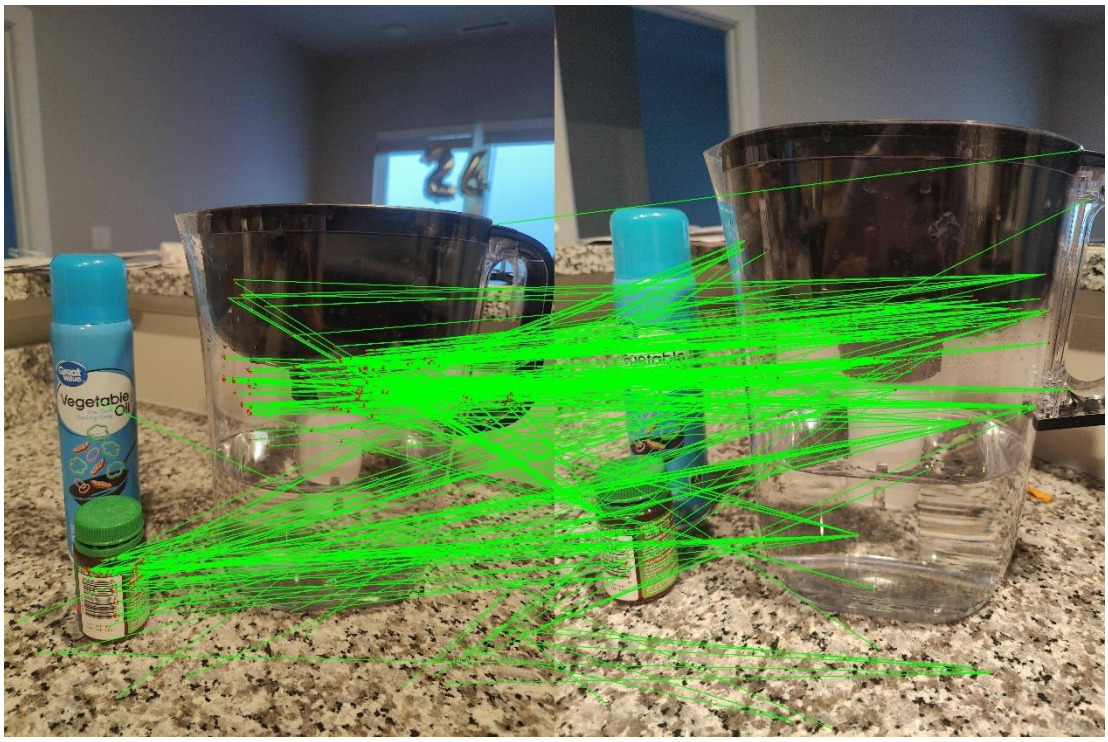


Figure 42 Output correspondences using Harris and SSD metric at $\sigma = 0.3$

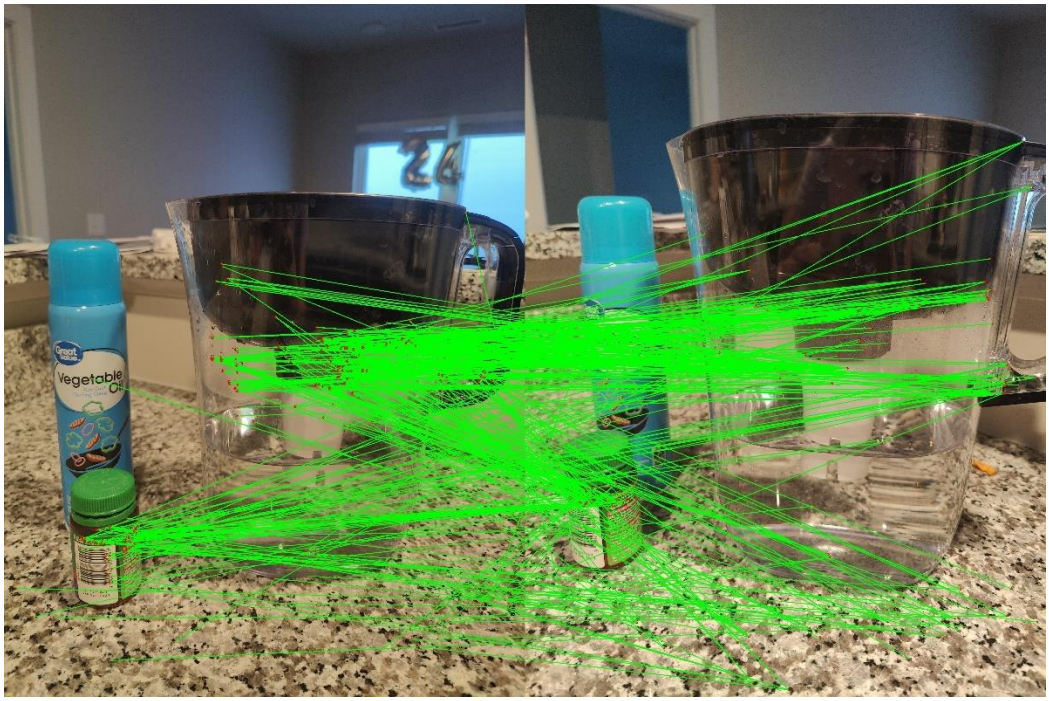


Figure 43 Output correspondences using Harris and NCC metric at $\sigma = 0.3$

At $\sigma = 0.6$:

Figure 44 Output showing corners using Harris at $\sigma = 0.6$

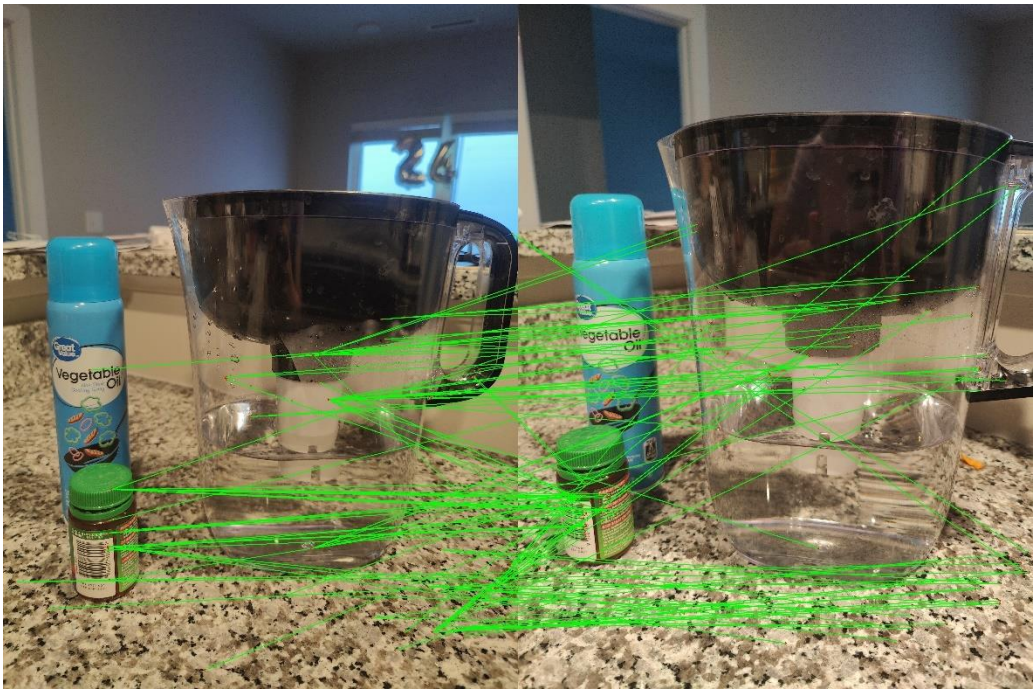


Figure 45 Output correspondences using Harris and SSD metric at $\sigma = 0.6$

Figure 46 Output correspondences using Harris and NCC metric at $\sigma = 0.6$

At $\sigma = 1.2$:

Figure 47 Output showing corners using Harris at $\sigma = 1.2$

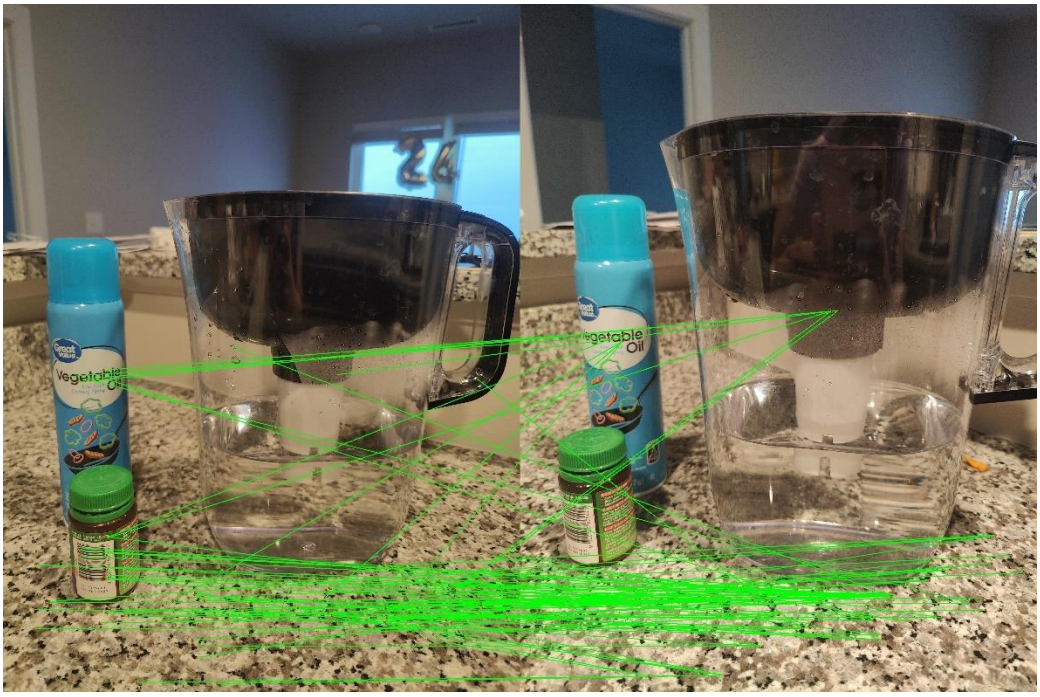


Figure 48 Output correspondences using Harris and SSD metric at $\sigma = 1.2$

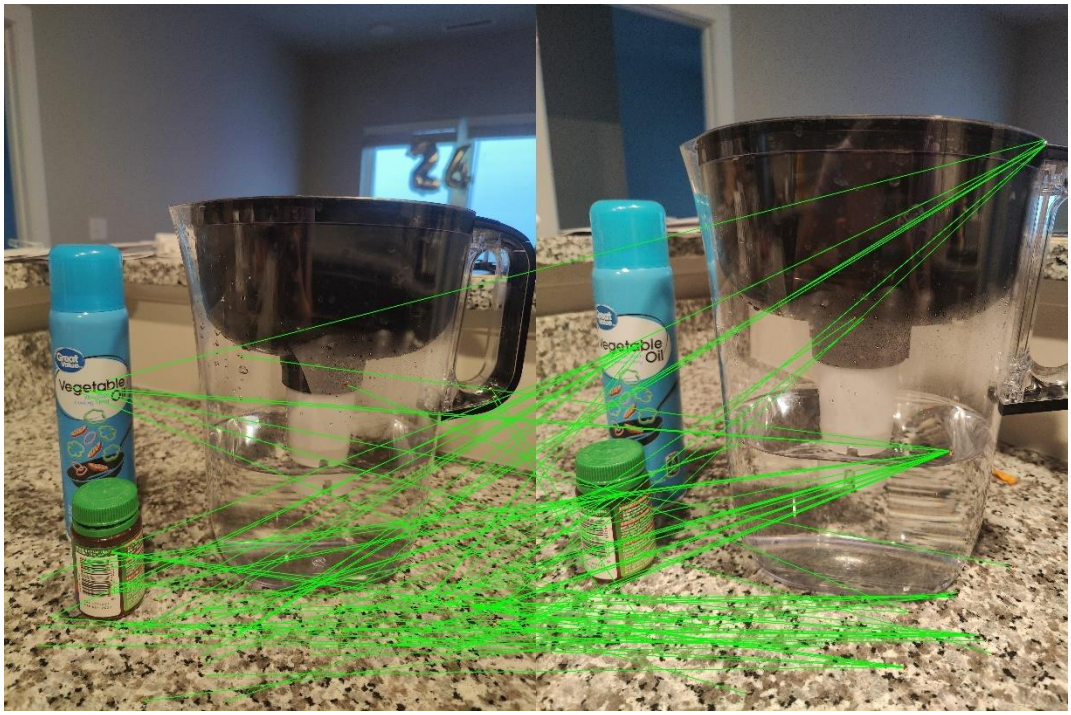


Figure 49 Output correspondences using Harris and NCC metric at $\sigma = 1.2$

At $\sigma = 2.4$*:*

Figure 50 Output showing corners using Harris at $\sigma = 2.4$

Figure 51 Output correspondences using Harris and SSD metric at $\sigma = 2.4$

Figure 52 Output correspondences using Harris and NCC metric at $\sigma = 2.4$

2.2 SIFT Algorithm

Figure 53 Output correspondences of input pair of images in (c) using SIFT algorithm

Figure 54 Output correspondences of input pair of images in (d) using SIFT algorithm

2.3 SuperPoint and SuperGlue

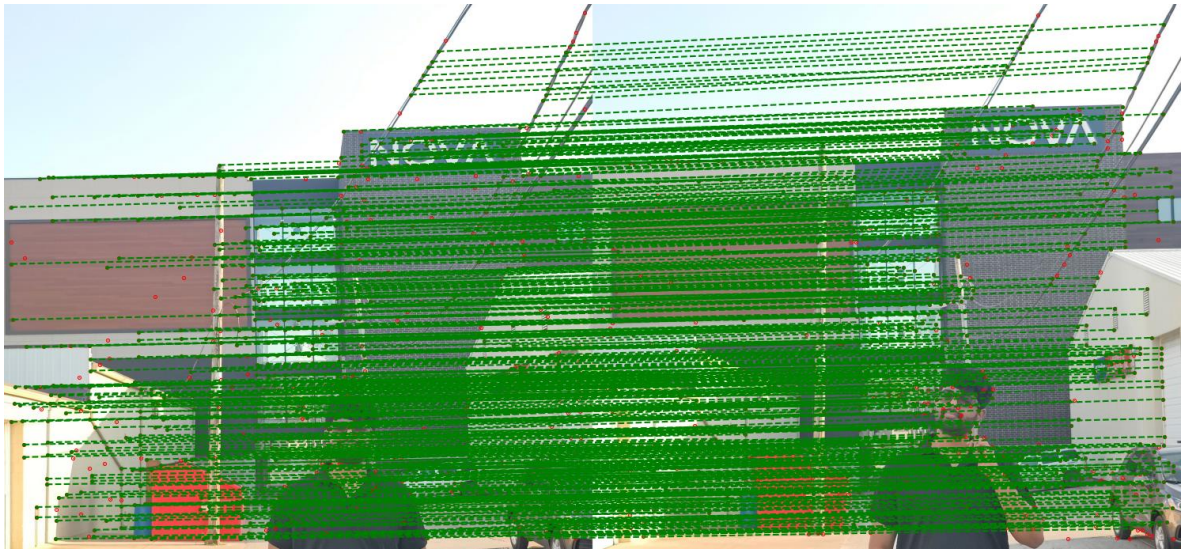


Figure 55 Output correspondences of input pair of images in (c) using SuperPoint + SuperGlue

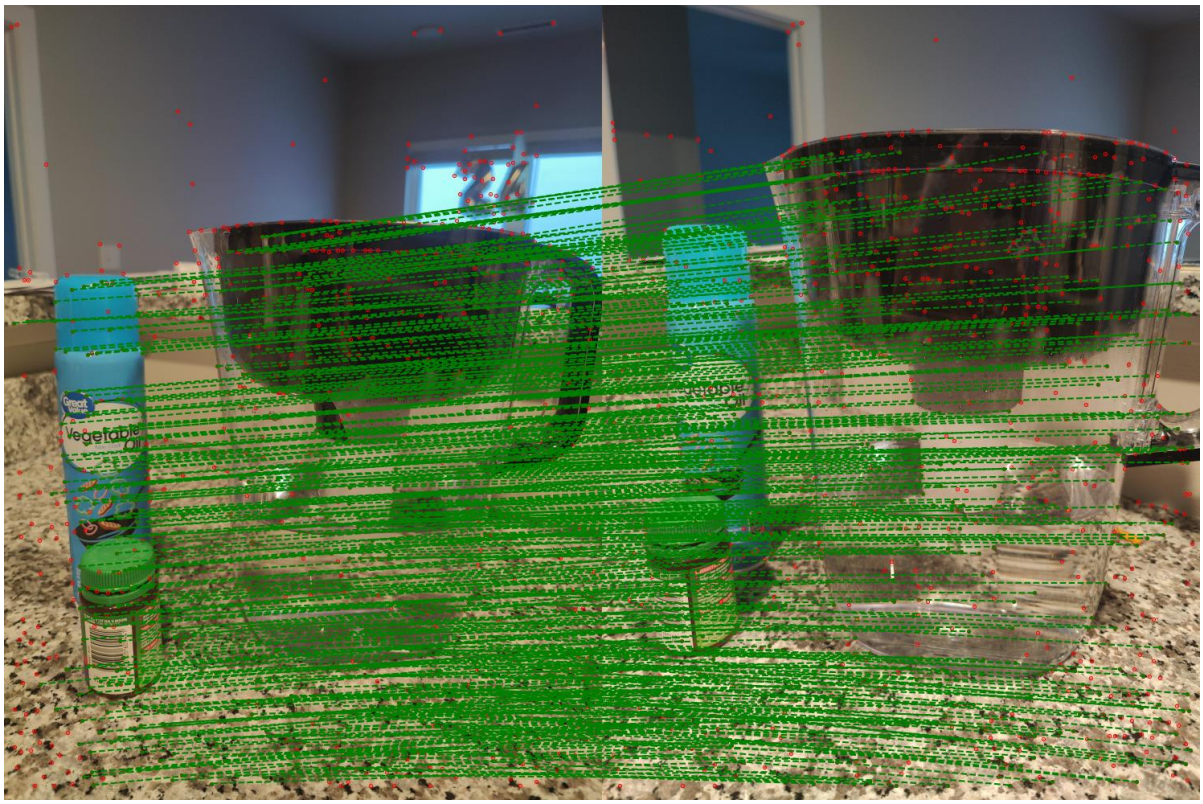


Figure 56 Output correspondences of input pair of images in (d) using SuperPoint + SuperGlue

Observations:

The observations on the own images are similar to those in Task 1 images, the neural network outweighed the outputs of both SIFT and Harris, while SIFT performed better than Harris as discussed in Task1. Further we can observe that the algorithm struggled a bit due to similar blobs on either side of the images in image pairs, but still performed well.

SOURCE CODE:

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

# <h2><center>ECE661 COMPUTER VISION</center></h2>
# <h3><center>Homework - 4</center></h3>
# <h3><center>Sahithi Kodali - 34789866</center></h3>
# <h3><center>kodali1@purdue.edu</center></h3>

# ## Task 1
# ### 1.1 Harris Corner Detector

# In[1]:


# Import libraries needed

import numpy as np
import matplotlib.pyplot as plt
import skimage.io as sk
import cv2
import PIL
import math


# In[2]:


# Read the images from uplaoded files/directly by giving path in the computer

books1_img = sk.imread('books_1.jpeg')
books2_img = sk.imread('books_2.jpeg')
fountain1_img = sk.imread('fountain_1.jpg')
fountain2_img = sk.imread('fountain_2.jpg')


# In[3]:


#find harr wavelets filters which are based on teh smallest even number above
4 * sigma based on teh scale value
def haar_wavelets (sigma):

    #determine kernel size
    k_size = int(math.ceil(sigma * 4))
    k_size = k_size + k_size%2
```

```
    #initiate kernels
    haar_x = np.ones((k_size, k_size))
    haar_y = np.ones((k_size, k_size))

    #modify as haar wavelets
    haar_x[:, : int(k_size//2)] = -1
    haar_y[int(k_size//2) :, :] = -1

    return haar_x, haar_y


# In[4]:


#normalize RGB image by converting to grey scale image i.e channel of 1 and
dividing with 255.

def normalize_img(image):
    if len (image.shape) == 3:
        norm_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        norm_img = image
    return norm_img/255


# In[5]:


def harris(image, sigma):

    #normalize image
    img = normalize_img(image)

    #obtain haar wavelets
    hx, hy = haar_wavelets(sigma)

    #define width and height of the image
    h, w = img.shape

    #Find the gradients
    Ix = cv2.filter2D(img, -1, hx)
    Iy = cv2.filter2D(img, -1, hy)

    #determine the window which is (5*sigma, 5*sigma) size kernel to find the
correlation at each pixel
    window_size = int(math.ceil(5 * sigma))
    window_size = window_size + window_size%2
    sum_window = np.ones((window_size, window_size))
```

```
    #compute the matrix C by finding the respective matrix values
    sum_dx2 = cv2.filter2D( Ix * Ix, -1, sum_window)
    sum_dy2 = cv2.filter2D( Iy * Iy, -1, sum_window)
    sum_dxy = cv2.filter2D( Ix * Iy, -1, sum_window)

    det_c = (sum_dx2 * sum_dy2) - (sum_dxy * sum_dxy)
    trace_c = sum_dx2 + sum_dy2

    #find the ratio k value
    k = det_c / (trace_c ** 2 + 0.000001)
    k = np.sum(k)/ (h*w)

    #compute response and give a threshold fo first 500 values
    response = det_c - (k * trace_c ** 2)
    response_thresholded = np.sort(response.flatten())[-500]

    #perform non-max supression on window aroudn each pixel
    nms_corner_points = []
    nms_window = int(window_size//2)
    print(nms_window)

    for x in range (nms_window, w - nms_window, 1):
        for y in range (nms_window, h - nms_window, 1):

            window = response[ y - nms_window : y + nms_window, x - nms_window
: x + nms_window]

            if response[y,x] == window.max() and window.max() >=
response_thresholded:
                nms_corner_points.append([x,y])

                #mark circles using random colors at each corner point
determined
                rand_color = (list(np.random.choice(range(256), size=3)))
                color =[int(rand_color[0]), int(rand_color[1]),
int(rand_color[2])]
                opt_img = cv2.circle (image, (x,y), 4, color , 3)


    return nms_corner_points, opt_img, sigma


# In[6]:


#initiate sigma values
sigma_vals = [0.3, 0.6, 1.2, 2.4]
```

```
#make copy of images to not override the original images required for further
tasks
books1_copy = books1_img.copy()
books2_copy = books2_img.copy()
fountain1_copy = fountain1_img.copy()
fountain2_copy = fountain2_img.copy()


# In[7]:


#Compute the distance metrics SSD and NCC to compute distance required to
match corresponsing interest points in the image pair

def distance_metric(img1, points1, img2, points2, metric = '', k = 21):
#     k_win = int(k_win//2)
#     print(k_win)

    #give the 21 x 21 window for neighbourhood around each pixel
    w1 = img1[ points1[1] : points1[1] + k, points1[0] : points1[0] + k
].flatten()
    w2 = img2[ points2[1] : points2[1] + k, points2[0] : points2[0] + k
].flatten()

    #compute SSD and NCC accoording to the formula in the theory concepts
    if metric == 'SSD':
        distance = np.sum( (w1 - w2) ** 2)
    elif metric == 'NCC':
        mean1 = w1.mean()
        mean2 = w2.mean()

        ncc_num = np.sum ((w1 - mean1) * (w2 - mean2))
        ncc_den = np.sqrt (np.sum((w1 - mean1) ** 2) * np.sum((w2 - mean2) **
2))

        distance = ncc_num/ (ncc_den + 0.000001)

    return distance


# In[8]:


#The corresponding interest point pairs are matched by computing the distance
accoording to chosen metric
#The points with shortest distacne are chosen as the pair

def harris_correspondences(image1, points1, image2, points2, metric, pad):
```

```
    #normalize images
    img1 = normalize_img(image1)
    img2 = normalize_img(image2)

    #pad the images
    img1 = cv2.copyMakeBorder(img1, pad, pad, pad, pad, cv2.BORDER_CONSTANT,
value = 0)
    img2 = cv2.copyMakeBorder(img2, pad, pad, pad, pad, cv2.BORDER_CONSTANT,
value = 0)

    #compute distance between every point in an image with every oterh point
    #sort distances acocordingly and return the points with best matching
pairs
    #a circle for the interest points deduced and the corresponding matching
lines are drawn
    if len(points1) <= len (points2):
        w = image1.shape[1]
        images_concat = np.concatenate((image1, image2), axis = 1)

        for p1 in points1:
            dist_tmp = []
            for p2 in points2:
                distance = distance_metric(img1, p1, img2, p2, metric =
metric, k = int(2*pad + 1))
                dist_tmp.append(distance)

            dist_sorted = points2[np.argsort(dist_tmp)[0]]

            if np.min(dist_tmp) < 30:
                pp1 = tuple(p1)
                pp2 = (dist_sorted[0] + w, dist_sorted[1])

                cv2.circle (images_concat, pp1 , 4, (0,0,255) , 3)
                cv2.circle (images_concat, pp2 , 4, (0,0,255) , 3)
                cv2.line (images_concat, pp1 , pp2, (0,255,0) , 3)


    else:
        w = image2.shape[1]
        images_concat = np.concatenate((image1, image2), axis = 1)

        for p2 in points2:
            dist_tmp = []
            for p1 in points1:
                distance = distance_metric(img1, p1, img2, p2, metric =
metric, k = int(2*pad + 1))
                dist_tmp.append(distance)
```

```
            dist_sorted = points1[np.argsort(dist_tmp)[0]]

            if np.min(dist_tmp) < 30:
                pp1 = (dist_sorted[0], dist_sorted[1])
                pp2 = (p2[0] + w, p2[1])

                cv2.circle (images_concat, pp1 , 4, (0,0,255) , 3)
                cv2.circle (images_concat, pp2 , 4, (0,0,255) , 3)
                cv2.line (images_concat, pp1 , pp2, (0,255,0) , 3)

    return images_concat


# In[9]:


points_b1, opt_img_b1, sigma = harris(books1_copy, 0.3)
points_b2, opt_img_b2, sigma = harris(books2_copy, 0.3)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b1'
+'.jpg'
cv2.imwrite(filename1, opt_img_b1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b2'
+'.jpg'
cv2.imwrite(filename2, opt_img_b2)

SSDbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
books' +'.jpg'
cv2.imwrite(filename3, SSDbooks_12)

NCCbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
books' +'.jpg'
cv2.imwrite(filename4, NCCbooks_12)


# In[10]:


points_b1, opt_img_b1, sigma = harris(books1_copy, 0.6)
points_b2, opt_img_b2, sigma = harris(books2_copy, 0.6)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b1'
+'.jpg'
cv2.imwrite(filename1, opt_img_b1)
```

```
filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b2'
+'.jpg'
cv2.imwrite(filename2, opt_img_b2)

SSDbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
books' +'.jpg'
cv2.imwrite(filename3, SSDbooks_12)

NCCbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
books' +'.jpg'
cv2.imwrite(filename4, NCCbooks_12)


# In[11]:


points_b1, opt_img_b1, sigma = harris(books1_copy, 1.2)
points_b2, opt_img_b2, sigma = harris(books2_copy, 1.2)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b1'
+'.jpg'
cv2.imwrite(filename1, opt_img_b1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b2'
+'.jpg'
cv2.imwrite(filename2, opt_img_b2)

SSDbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
books' +'.jpg'
cv2.imwrite(filename3, SSDbooks_12)

NCCbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
books' +'.jpg'
cv2.imwrite(filename4, NCCbooks_12)


# In[12]:


points_b1, opt_img_b1, sigma = harris(books1_copy, 2.4)
```

```
points_b2, opt_img_b2, sigma = harris(books2_copy, 2.4)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b1'
+'.jpg'
cv2.imwrite(filename1, opt_img_b1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'b2'
+'.jpg'
cv2.imwrite(filename2, opt_img_b2)

SSDbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
books' +'.jpg'
cv2.imwrite(filename3, SSDbooks_12)

NCCbooks_12 = harris_correspondences(books1_img, points_b1, books2_img,
points_b2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
books' +'.jpg'
cv2.imwrite(filename4, NCCbooks_12)


# ##### Fountain imgs - Harris

# In[13]:


points_f1, opt_img_f1, sigma = harris(fountain1_copy, 0.3)
points_f2, opt_img_f2, sigma = harris(fountain2_copy, 0.3)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f1'
+'.jpg'
cv2.imwrite(filename1, opt_img_f1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f2'
+'.jpg'
cv2.imwrite(filename2, opt_img_f2)

SSDfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
fountain' +'.jpg'
cv2.imwrite(filename3, SSDfountain_12)

NCCfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
fountain' +'.jpg'
```

```
cv2.imwrite(filename4, NCCfountain_12)


# In[14]:


points_f1, opt_img_f1, sigma = harris(fountain1_copy, 0.6)
points_f2, opt_img_f2, sigma = harris(fountain2_copy, 0.6)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f1'
+'.jpg'
cv2.imwrite(filename1, opt_img_f1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f2'
+'.jpg'
cv2.imwrite(filename2, opt_img_f2)

SSDfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
fountain' +'.jpg'
cv2.imwrite(filename3, SSDfountain_12)

NCCfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
fountain' +'.jpg'
cv2.imwrite(filename4, NCCfountain_12)


# In[15]:


points_f1, opt_img_f1, sigma = harris(fountain1_copy, 1.2)
points_f2, opt_img_f2, sigma = harris(fountain2_copy, 1.2)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f1'
+'.jpg'
cv2.imwrite(filename1, opt_img_f1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f2'
+'.jpg'
cv2.imwrite(filename2, opt_img_f2)

SSDfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
fountain' +'.jpg'
cv2.imwrite(filename3, SSDfountain_12)
```

```
NCCfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
fountain' +'.jpg'
cv2.imwrite(filename4, NCCfountain_12)


# In[16]:


points_f1, opt_img_f1, sigma = harris(fountain1_copy, 2.4)
points_f2, opt_img_f2, sigma = harris(fountain2_copy, 2.4)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f1'
+'.jpg'
cv2.imwrite(filename1, opt_img_f1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'f2'
+'.jpg'
cv2.imwrite(filename2, opt_img_f2)

SSDfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
fountain' +'.jpg'
cv2.imwrite(filename3, SSDfountain_12)

NCCfountain_12 = harris_correspondences(fountain1_img, points_f1,
fountain2_img, points_f2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
fountain' +'.jpg'
cv2.imwrite(filename4, NCCfountain_12)


# ### 1.2 SIFT algorithm

# In[29]:


#convert image into grey scale of channel 1
def normalize_img2(image):
    if len (image.shape) == 3:
        norm_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        norm_img = image
    return norm_img
```

```
# In[30]:


#compute the SIFT matching using OpenCV techniques
def SIFT(image1, image2):
    img1 = normalize_img2(image1)
    img2 = normalize_img2(image2)

    #create SIFT
    sift = cv2.xfeatures2d.SIFT_create()

    #compute the keypoints and descriptors for the respective images
    keypoints_1, descriptors_1 = sift.detectAndCompute(img1, None)
    keypoints_2, descriptors_2 = sift.detectAndCompute(img2, None)

    #match the points based on sorted distance
    bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

    matches = bf.match(descriptors_1, descriptors_2)
    matches = sorted(matches, key = lambda x: x.distance)

    #draw lines for corresponding matches
    img_concat = cv2.drawMatches(image1, keypoints_1, image2, keypoints_2,
matches[0:80], image2, flags=2)

    plt.imshow(img_concat)
    return img_concat


# In[31]:


sift_books = SIFT(books1_img, books2_img)
filename = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\SIFT' + 'books' +'.jpg'
cv2.imwrite(filename, sift_books)


# In[32]:


sift_fountain = SIFT(fountain1_img, fountain2_img)
filename = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\SIFT' + 'fountain' +'.jpg'
cv2.imwrite(filename, sift_fountain)


# ## Task 2 (Own images)
# ###  2.1 Harris Corner Detector

# In[20]:
```

```
nova1_img = cv2.imread('nova1.jpg')
nova2_img = cv2.imread('nova2.jpg')

# night1_img = cv2.imread('night1.jpg')
# night2_img = cv2.imread('night2.jpg')

bottles1_img = cv2.imread('bottles1.jpg')
bottles2_img = cv2.imread('bottles2.jpg')

# apts1_img = cv2.imread('apts1.jpg')
# apts2_img = cv2.imread('apts2.jpg')

#make copy of images to not override the original images required for further
tasks
nova1_copy = nova1_img.copy()
nova2_copy = nova2_img.copy()
bottles1_copy = bottles1_img.copy()
bottles2_copy = bottles2_img.copy()


# In[21]:


points_n1, opt_img_n1, sigma = harris(nova1_copy, 0.3)
points_n2, opt_img_n2, sigma = harris(nova2_copy, 0.3)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n1'
+'.jpg'
cv2.imwrite(filename1, opt_img_n1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n2'
+'.jpg'
cv2.imwrite(filename2, opt_img_n2)

SSDnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
nova' +'.jpg'
cv2.imwrite(filename3, SSDnova_12)

NCCnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
nova' +'.jpg'
cv2.imwrite(filename4, NCCnova_12)
```

```
# In[22]:


points_n1, opt_img_n1, sigma = harris(nova1_copy, 0.6)
points_n2, opt_img_n2, sigma = harris(nova2_copy, 0.6)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n1'
+'.jpg'
cv2.imwrite(filename1, opt_img_n1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n2'
+'.jpg'
cv2.imwrite(filename2, opt_img_n2)

SSDnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
nova' +'.jpg'
cv2.imwrite(filename3, SSDnova_12)

NCCnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
nova' +'.jpg'
cv2.imwrite(filename4, NCCnova_12)


# In[23]:


points_n1, opt_img_n1, sigma = harris(nova1_copy, 1.2)
points_n2, opt_img_n2, sigma = harris(nova2_copy, 1.2)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n1'
+'.jpg'
cv2.imwrite(filename1, opt_img_n1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n2'
+'.jpg'
cv2.imwrite(filename2, opt_img_n2)

SSDnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
nova' +'.jpg'
cv2.imwrite(filename3, SSDnova_12)

NCCnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'NCC', 10 )
```

```
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
nova' +'.jpg'
cv2.imwrite(filename4, NCCnova_12)


# In[24]:


points_n1, opt_img_n1, sigma = harris(nova1_copy, 2.4)
points_n2, opt_img_n2, sigma = harris(nova2_copy, 2.4)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n1'
+'.jpg'
cv2.imwrite(filename1, opt_img_n1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'n2'
+'.jpg'
cv2.imwrite(filename2, opt_img_n2)

SSDnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
nova' +'.jpg'
cv2.imwrite(filename3, SSDnova_12)

NCCnova_12 = harris_correspondences(nova1_img, points_n1, nova2_img,
points_n2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
nova' +'.jpg'
cv2.imwrite(filename4, NCCnova_12)


# bootles-imgs-harris

# In[25]:


points_bt1, opt_img_bt1, sigma = harris(bottles1_copy, 0.3)
points_bt2, opt_img_bt2, sigma = harris(bottles2_copy, 0.3)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt1'
+'.jpg'
cv2.imwrite(filename1, opt_img_bt1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt2'
+'.jpg'
cv2.imwrite(filename2, opt_img_bt2)
```

```
SSDbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
bottles' +'.jpg'
cv2.imwrite(filename3, SSDbottles_12)

NCCbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
bottles' +'.jpg'
cv2.imwrite(filename4, NCCbottles_12)


# In[26]:


points_bt1, opt_img_bt1, sigma = harris(bottles1_copy, 0.6)
points_bt2, opt_img_bt2, sigma = harris(bottles2_copy, 0.6)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt1'
+'.jpg'
cv2.imwrite(filename1, opt_img_bt1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt2'
+'.jpg'
cv2.imwrite(filename2, opt_img_bt2)

SSDbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
bottles' +'.jpg'
cv2.imwrite(filename3, SSDbottles_12)

NCCbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
bottles' +'.jpg'
cv2.imwrite(filename4, NCCbottles_12)


# In[27]:


points_bt1, opt_img_bt1, sigma = harris(bottles1_copy, 1.2)
points_bt2, opt_img_bt2, sigma = harris(bottles2_copy, 1.2)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt1'
+'.jpg'
cv2.imwrite(filename1, opt_img_bt1)
```

```
filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt2'
+'.jpg'
cv2.imwrite(filename2, opt_img_bt2)

SSDbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
bottles' +'.jpg'
cv2.imwrite(filename3, SSDbottles_12)

NCCbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
bottles' +'.jpg'
cv2.imwrite(filename4, NCCbottles_12)


# In[28]:


points_bt1, opt_img_bt1, sigma = harris(bottles1_copy, 2.4)
points_bt2, opt_img_bt2, sigma = harris(bottles2_copy, 2.4)

filename1 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt1'
+'.jpg'
cv2.imwrite(filename1, opt_img_bt1)

filename2 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'bt2'
+'.jpg'
cv2.imwrite(filename2, opt_img_bt2)

SSDbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'SSD', 10 )
filename3 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'SSD
bottles' +'.jpg'
cv2.imwrite(filename3, SSDbottles_12)

NCCbottles_12 = harris_correspondences(bottles1_img, points_bt1, bottles2_img,
points_bt2, 'NCC', 10 )
filename4 = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\Harris' + str(sigma) + 'NCC
bottles' +'.jpg'
cv2.imwrite(filename4, NCCbottles_12)


# ### 2.2 SIFT algorithm

# In[33]:
```

```
sift_nova = SIFT(nova1_img, nova2_img)
filename = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\SIFT' + 'nova' +'.jpg'
cv2.imwrite(filename, sift_nova)


# In[34]:


sift_bottles = SIFT(bottles1_img, bottles2_img)
filename = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\SIFT' + 'bottles' +'.jpg'
cv2.imwrite(filename, sift_bottles)


# In[ ]:


# sift_apts = SIFT(apts1_img, apts2_img)
# filename = 'D:\Purdue\ECE661_CV\HW4_outputs' + '\SIFT' + 'apts' +'.jpg'
# cv2.imwrite(filename, sift_apts)
```