ECE 661 Homework #4

Name: Haoyu Chen
Email: chen1562@purdue.edu
ID Number: 00271-72202

26 Sept 2020

1 Logic and Computations

1.1 Theory Question

Q: 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.

A: By the fundamental theorem of scale space, we have $\frac{\partial}{\partial\sigma}ff(x,y,\sigma) = \sigma\nabla^2 ff(x,y,\sigma) = \sigma LoG(ff(x,y,\sigma))$. We also know that $\frac{\partial}{\partial x}f(x)$ can be estimated by $\frac{1}{\Delta x}(f(x+\Delta x) - f(x))$.

Hence, at a certain scale σ, we can estimate the $\frac{\partial}{\partial\sigma}ff(x,y,\sigma)$ by

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

By setting $\Delta\sigma = k\sigma - \sigma$, we now have

$$(k-1)\sigma^2\nabla^2 ff(x,y,\sigma) \approx ff(x,y,\sigma+\Delta\sigma) - ff(x,y,\sigma)$$

Since $(k-1)$ is a constant that does not affect the location of extrema, we can estimate the **scale-normalized Laplacian of Gaussian** $LOG_{normalized} = \sigma^2 LoG(ff(x,y,\sigma)) = \sigma^2\nabla^2 ff(x,y,\sigma)$ by difference of Gaussian $ff(x,y,\sigma+\Delta\sigma) - ff(x,y,\sigma)$.

When viewing both LoG and DoG as discrete convolutions, DoG requires a smaller kernel and therefore cost less computation power. For example, using a common choice $\sigma = \sqrt{2}$, LoG convolution requires a 13×13 kernel, while DoG only needs a 9×9 kernel.

In addition, DoG operation can be computed along x and y directions respectively, while LoG operator is inseparable and can only be carried out as 2D convolution.

1.2 Harris Corner Detector

The fundamental component in Harris Corner detector is the image intensity's gradient along x and y direction, denoted d_x and d_y. In this homework, d_x and d_y were computed using Haar filter (same as SURF) oriented on x and y directions respectively. For a given σ, the Haar filter is of size $N \times N$, where N is the smallest **even** integer larger than 4σ; it consists of $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ structure along y-direction and $\begin{bmatrix} -1 & 1 \end{bmatrix}$ structure along x-direction. For example, for $\sigma = 1.2$, the kernel used to compute d_x is

$$h_x = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

and the kernel used to compute d_y is

$$h_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

For each pixel, a matrix C is computed based on its $5\sigma \times 5\sigma$ neighbourhood:

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

Then, the Harris Response R is used to determine whether there is a corner on that pixel:

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

where the ratio $k = \frac{det(C)}{(Tr(C))^2}$. Larger R indicates higher likelihood of existence of a corner.

During implementation, the threshold was set to preserve the largest 500 computed Rs. Then, since multiple pixels could be detected for the same corner, non-maximum suppression scheme was used to eliminate excessive points.

1.3 Matching Point Pairs Using SSD or NCC

For interest points detected by Harris, there are no feature vectors to describe them; hence, the gray-scale version of the 21×21 region-of-interest around that pixel is used as the feature descriptor. Then, a distance metric is computed between two interest points. In this homework, I choose to assume that smallest distance indicates closest match. The distance is computed by either SSD or NCC.

For SSD (Sum of Squared Differences):

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

which is directly used as a distance metric. Also, an empirical threshold of 25 is set in this homework, which is to say, when the distance between two points is larger than 25, the match is no longer considered valid.

For NCC (Normalized Correlation Coefficient):

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - \mu_1)(f_2(i,j) - \mu_2)}{\sqrt{(\sum_i \sum_j (f_1(i,j) - \mu_1)^2)(\sum_i \sum_j (f_2(i,j) - \mu_2)^2)}}$$

where μ_1, μ_2 are means of the region-of-interest for the interest points in image 1 and image 2, respectively. Note that the direct output of NCC does not follow the assumption for a distance metric, since NCC=1 indicates the closest match. Hence, for NCC, I use $d = 1 - NCC$ as the distance metric so that the same scheme assuming that smaller distance equals closer match still stands,

1.4 SIFT Overview

In this homework, off-the-shelf SIFT algorithm was also used for interest point matching. SIFT (Scale-invariant feature transform) consists of following essential steps.

1. Constructing DoG (difference of Gaussian) pyramid; for a specific scale σ, the DoG values at (x, y) is denoted $D(x, y, \sigma)$ or sometimes $D(\boldsymbol{x})$, where $\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \sigma \end{bmatrix}$.

2. For finding locate extrema, each pixel is compared to (1) 8 surrounding pixels in it's 3×3 neighbourhood, (2) 9 pixels in it's 3×3 neighbourhood at next level in scale space, and (3) 9 pixels in it's 3×3 neighbourhood at previous level in scale space.

3. However, as σ increases, the extrema detected might be not as accurate as the image becomes more "coarse". Hence, we want to locate the extreme with better accuracy, and this can be done by estimating second-order derivatives. For a detected point $\boldsymbol{x_0} = \begin{bmatrix} x_0 \\ y_0 \\ \sigma_0 \end{bmatrix}$, the true extremum in its vicinity $\boldsymbol{x}$ can be estimated by $\boldsymbol{x} = -H^{-1}(\boldsymbol{x_0})J(\boldsymbol{x_0})$, where $H(\boldsymbol{x_0})$ and $J(\boldsymbol{x_0})$ are Hessian and gradient estimated at $\boldsymbol{x_0}$, respectively.

4. Then we apply threshold to reject weak extrema. A typical threshold is 0.03, i.e., if $|D(\boldsymbol{x})| < 0.03$, $\boldsymbol{x}$ is no longer considered an interest point.

5. At last, a dominant orientation and a 128-dimension feature descriptor is assigned to each extremum—which was considered as a candidate for interest point. This feature descriptor can be used directly for matching point pairs. In this homework, OpenCV's off-the-shelf brute force matcher was used to find point pairs by computing Euclidean distance between feature descriptors.

2 Task 1: Images and Results

2.1 Pair 1

Figure 1: Input image 1 and 2 for pair 1

2.1.1 Harris Corner Detector, $\sigma = 0.8$

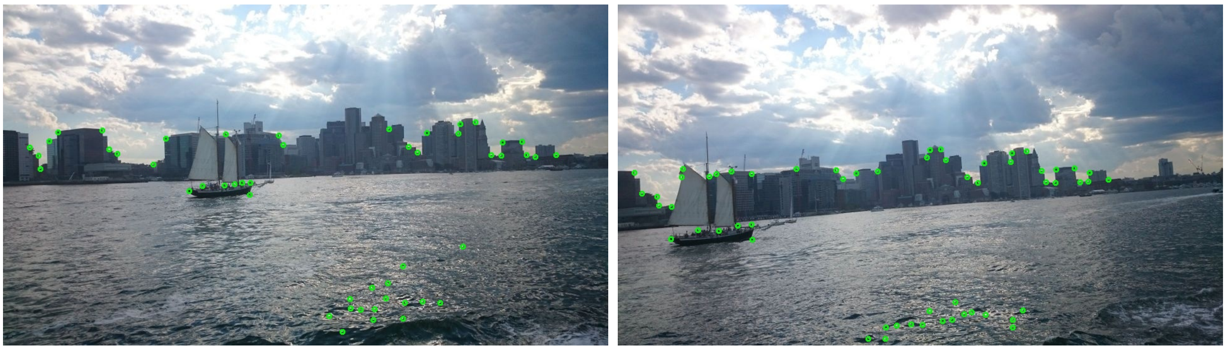

Figure 2: Output image 1 and 2 with detected corners

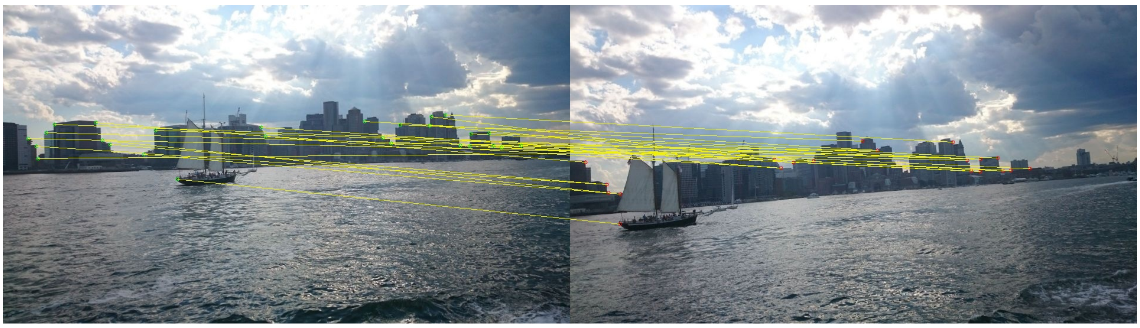

Figure 3: Harris output correspondences using SSD, $\sigma = 0.8$

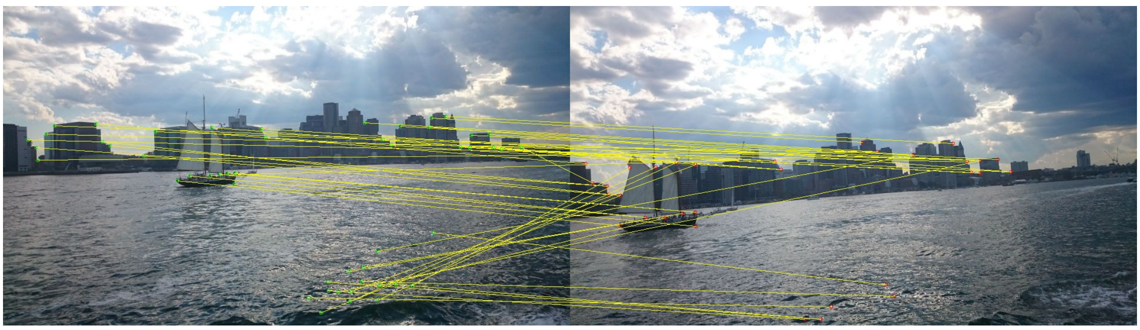

Figure 4: Harris output correspondences using NCC, $\sigma = 0.8$

2.1.2 Harris Corner Detector, $\sigma = 1.2$

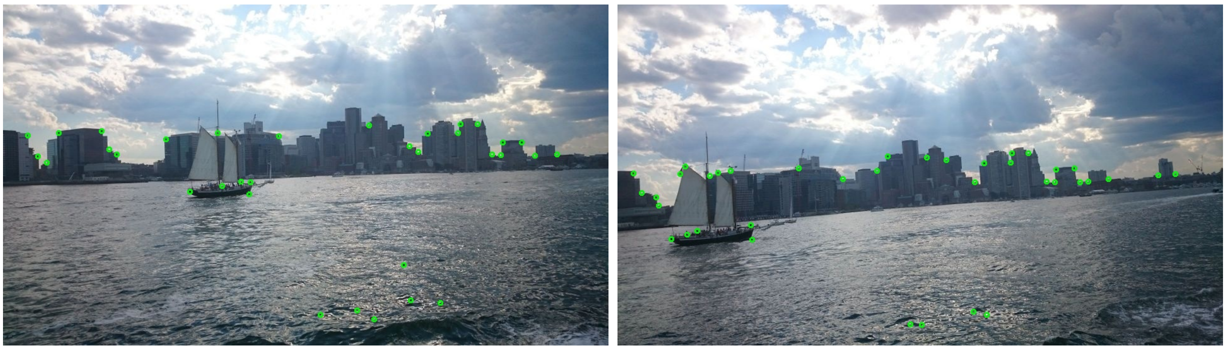

Figure 5: Output image 1 and 2 with detected corners

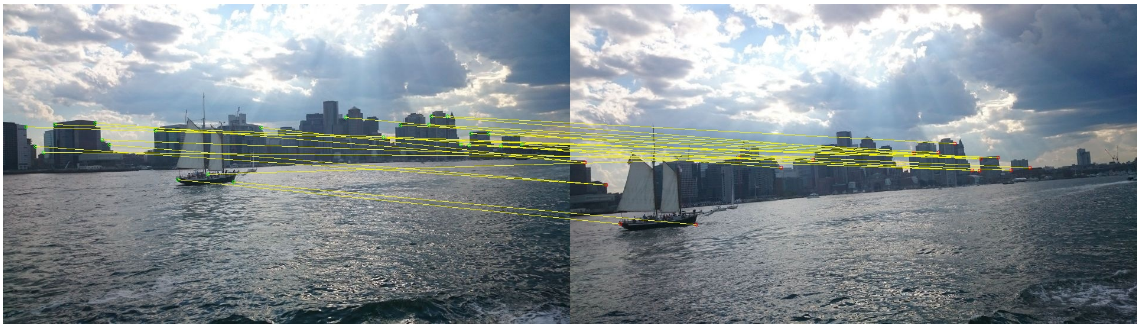

Figure 6: Harris output correspondences using SSD, $\sigma = 1.2$

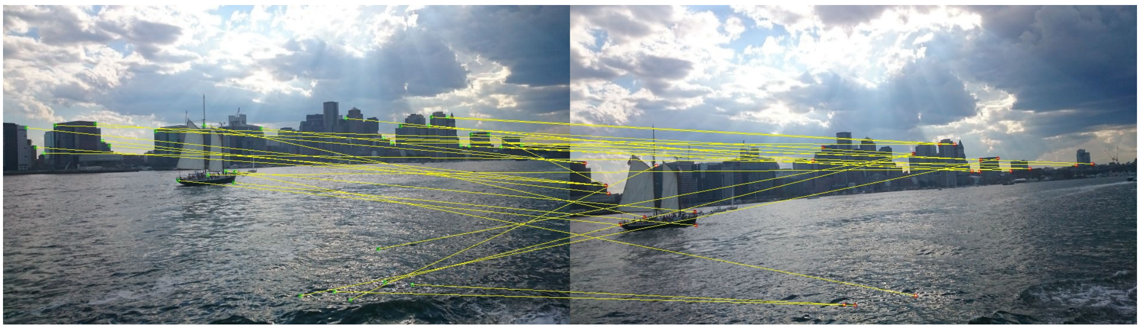

Figure 7: Harris output correspondences using NCC, $\sigma = 1.2$

At this point, we can already observe that as σ increases, the images' feature became more "coarse", and the number of interest points detected decreases. Hence, for following outputs, only point correspondences will be shown, and individual detected corners will not be shown.

2.1.3 Harris Corner Detector, $\sigma = 1.6$

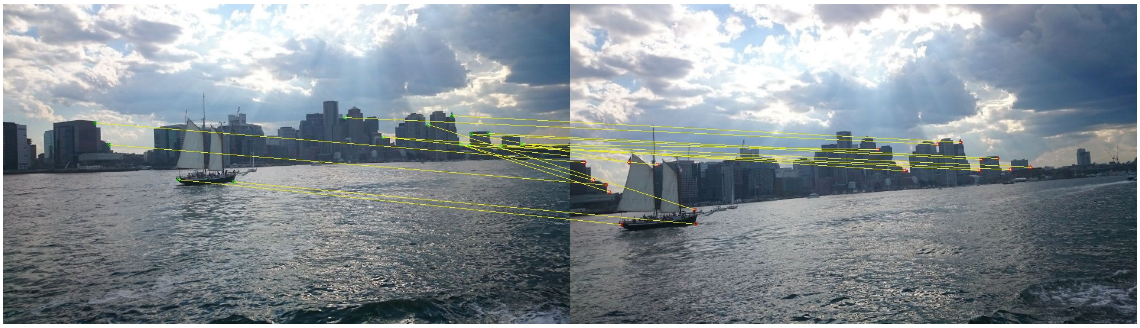

Figure 8: Harris output correspondences using SSD, $\sigma = 1.6$

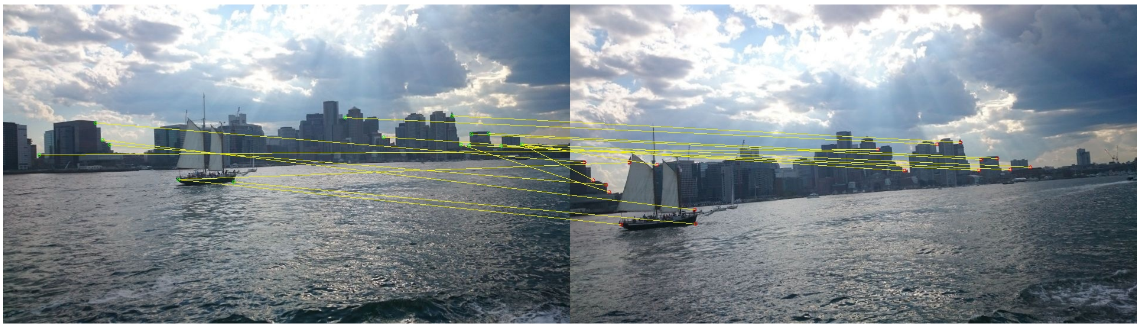

Figure 9: Harris output correspondences using NCC, $\sigma = 1.6$

2.1.4 Harris Corner Detector, $\sigma = 2.0$

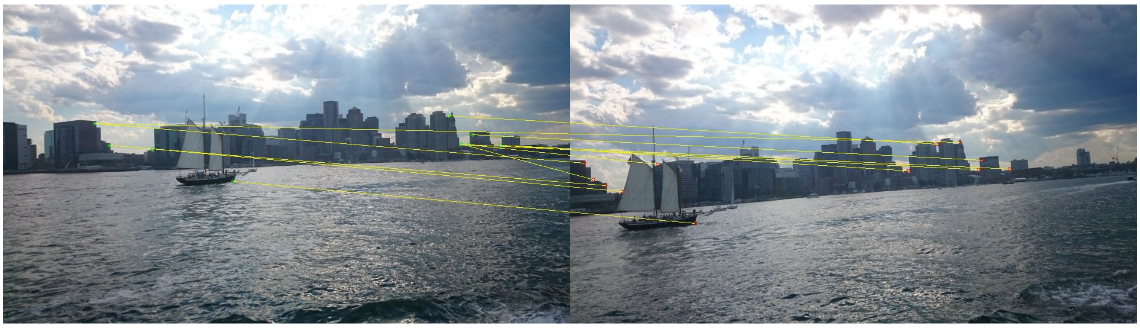

Figure 10: Harris output correspondences using SSD, $\sigma = 2.0$

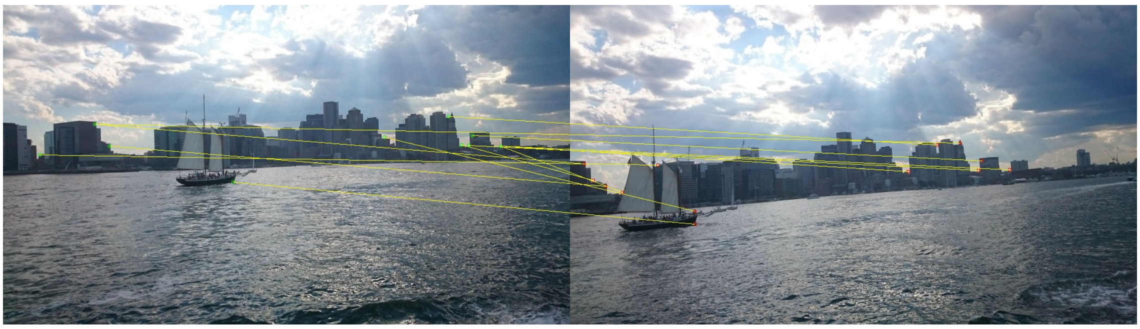

Figure 11: Harris output correspondences using NCC, $\sigma = 2.0$

2.1.5 SIFT

Since off-the-shelf SIFT algorithm can create a larger amount of feature points and correspondences, the first 100 SIFT correspondences are displayed in the second figure in the SIFT second for each pair, in order to present a more intuitive and less "messy" demonstration.

Figure 12: SIFT output correspondences

Figure 13: SIFT output correspondences (first 100 pairs)

2.2 Pair 2

Figure 14: Input image 1 and 2 for pair 2

2.2.1 Harris Corner Detector, $\sigma = 0.8$

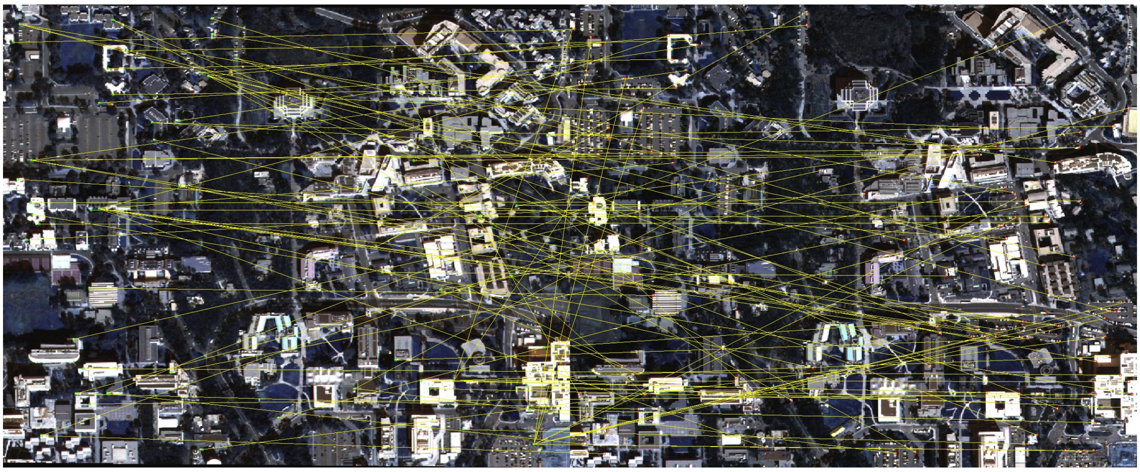

Figure 15: Harris output correspondences using SSD, $\sigma = 0.8$

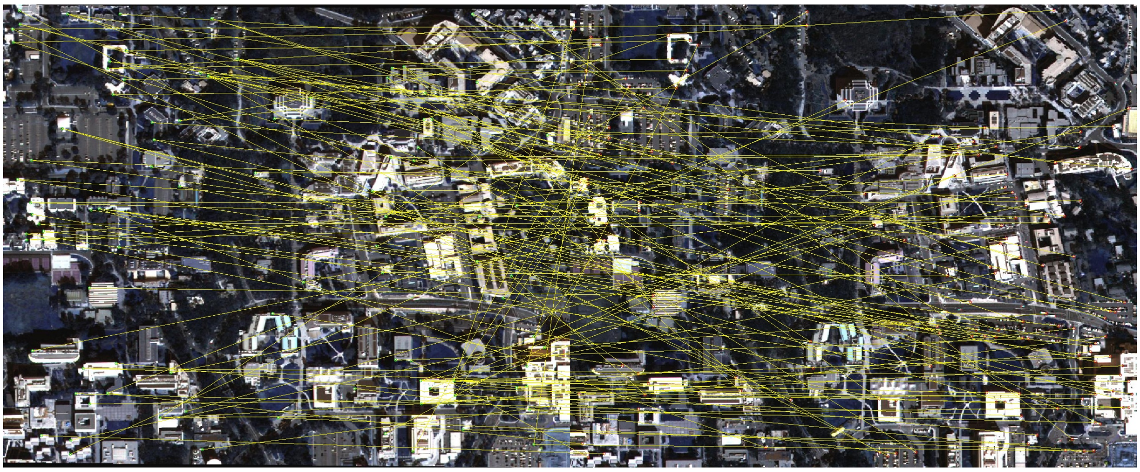

Figure 16: Harris output correspondences using NCC, $\sigma = 0.8$

2.2.2 Harris Corner Detector, $\sigma = 1.2$

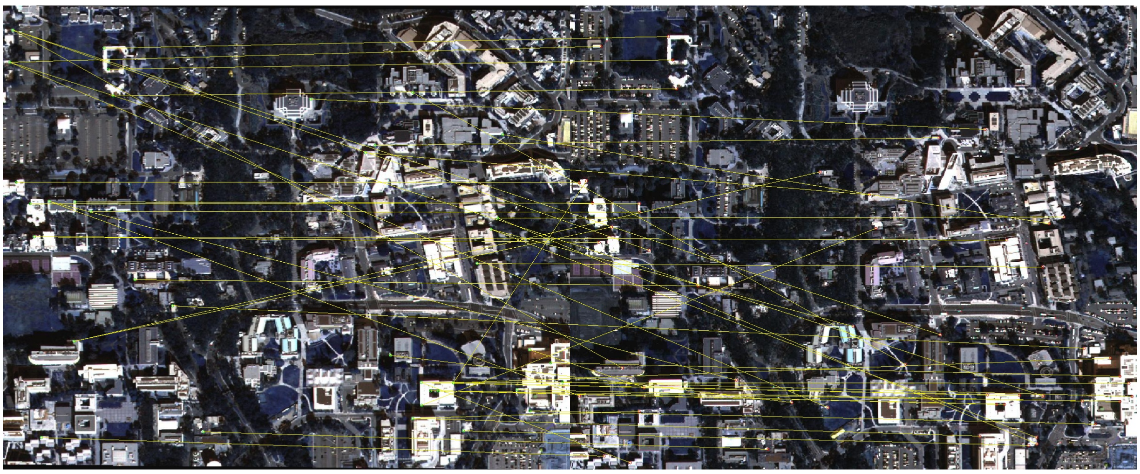

Figure 17: Harris output correspondences using SSD, $\sigma = 1.2$

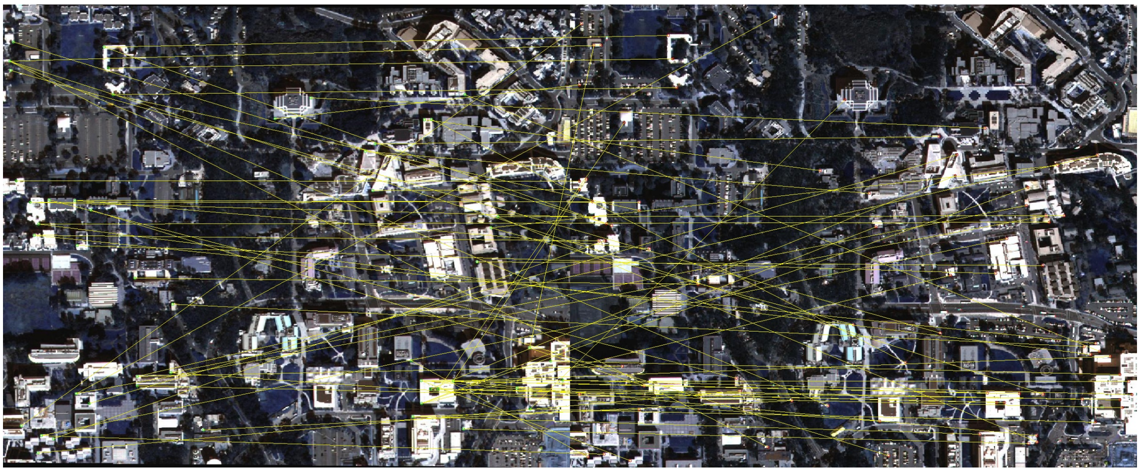

Figure 18: Harris output correspondences using NCC, $\sigma = 1.2$

2.2.3 Harris Corner Detector, $\sigma = 1.6$

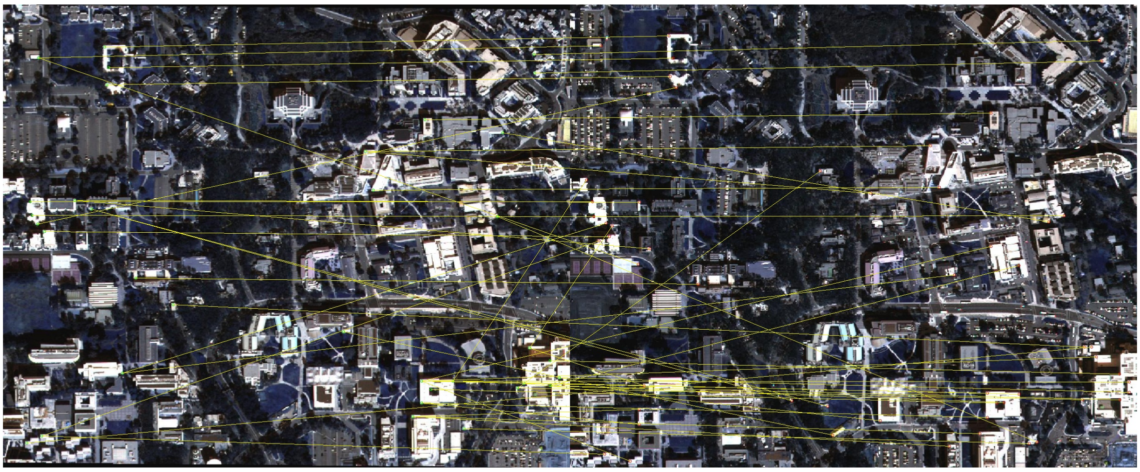

Figure 19: Harris output correspondences using SSD, $\sigma = 1.6$

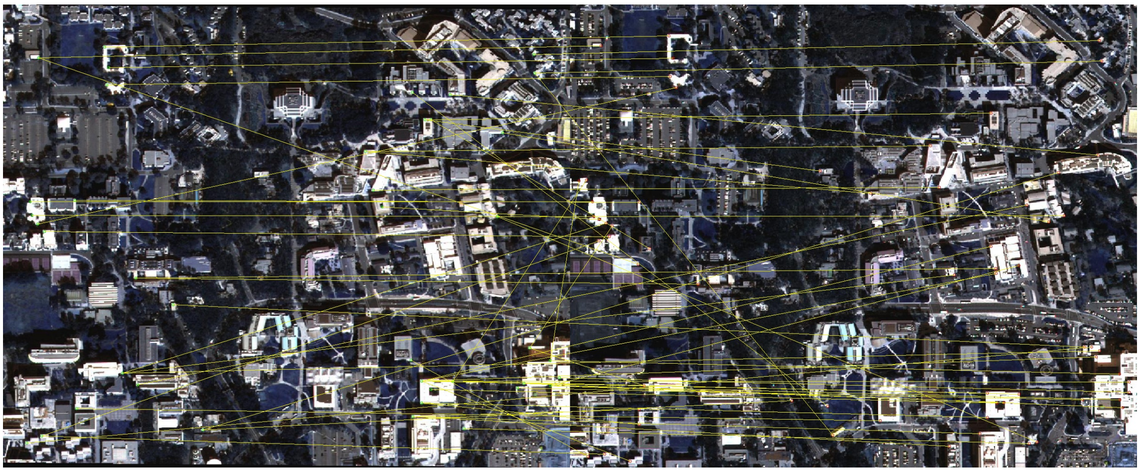

Figure 20: Harris output correspondences using NCC, $\sigma = 1.6$

2.2.4 Harris Corner Detector, $\sigma = 2.0$

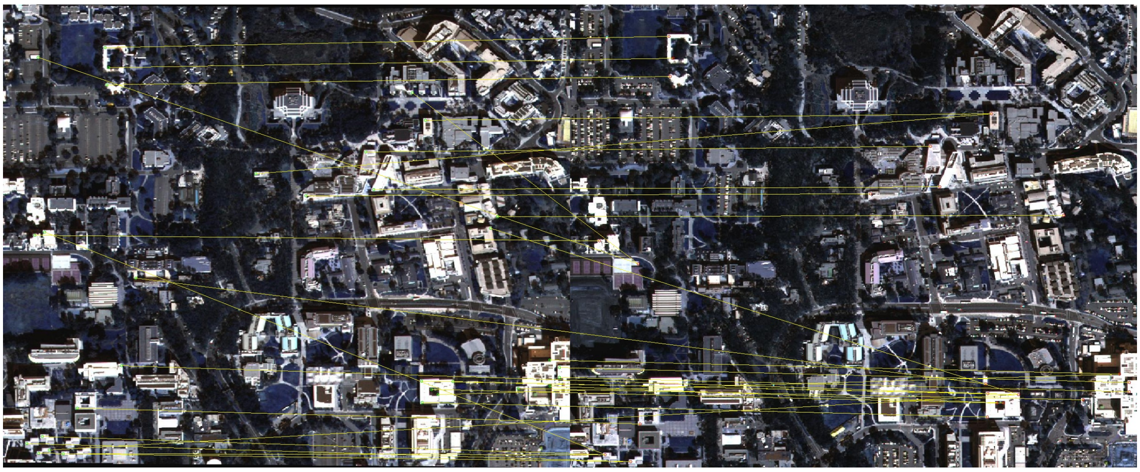

Figure 21: Harris output correspondences using SSD, $\sigma = 2.0$

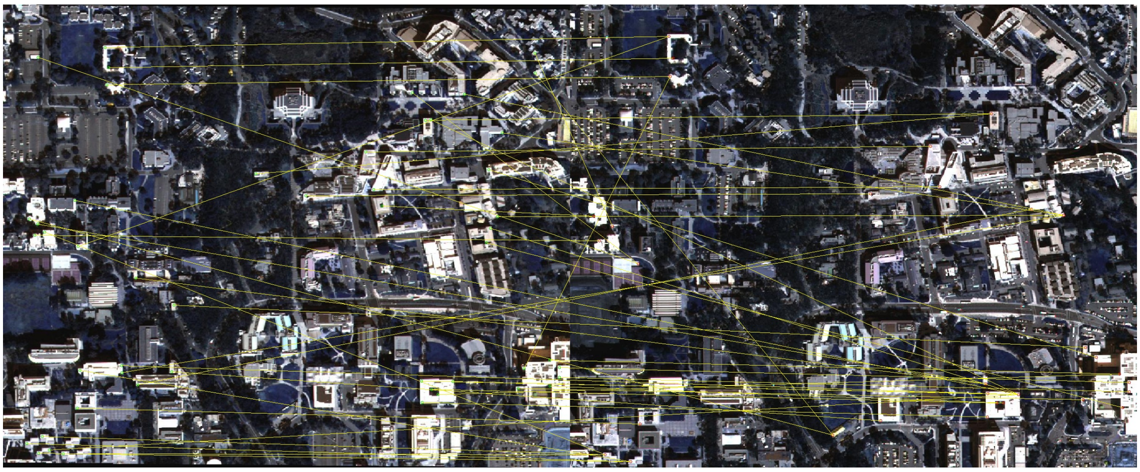

Figure 22: Harris output correspondences using NCC, $\sigma = 2.0$

2.2.5 SIFT

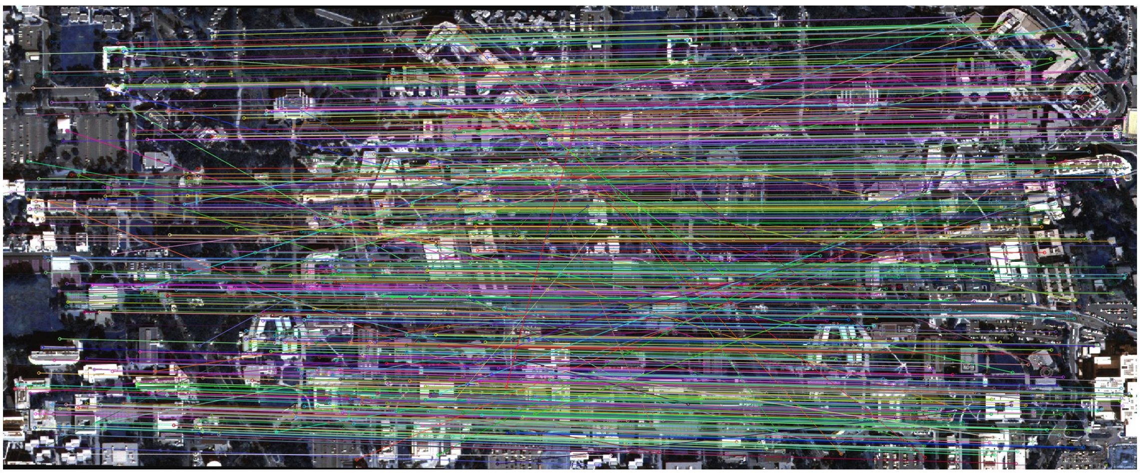

Figure 23: SIFT output correspondences

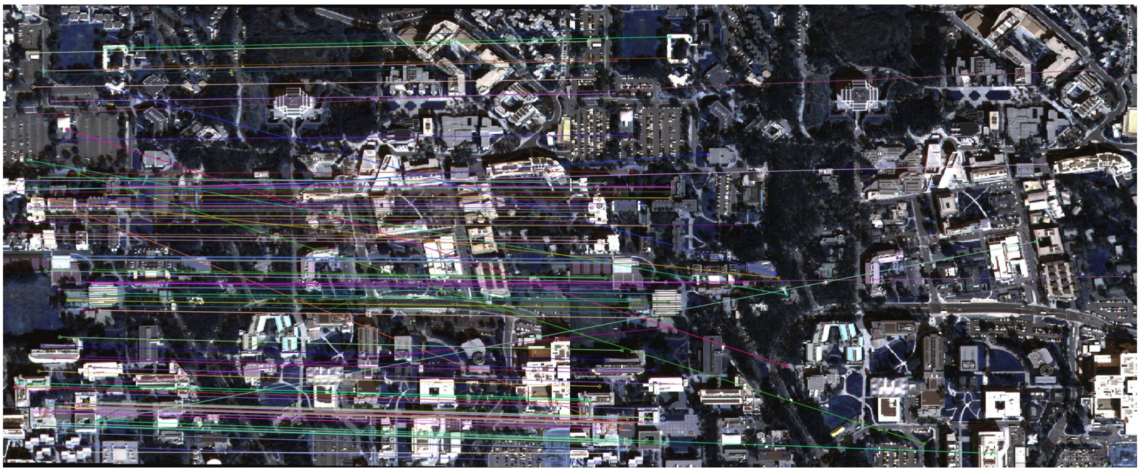

Figure 24: SIFT output correspondences (first 100 pairs)

2.3 Pair 3

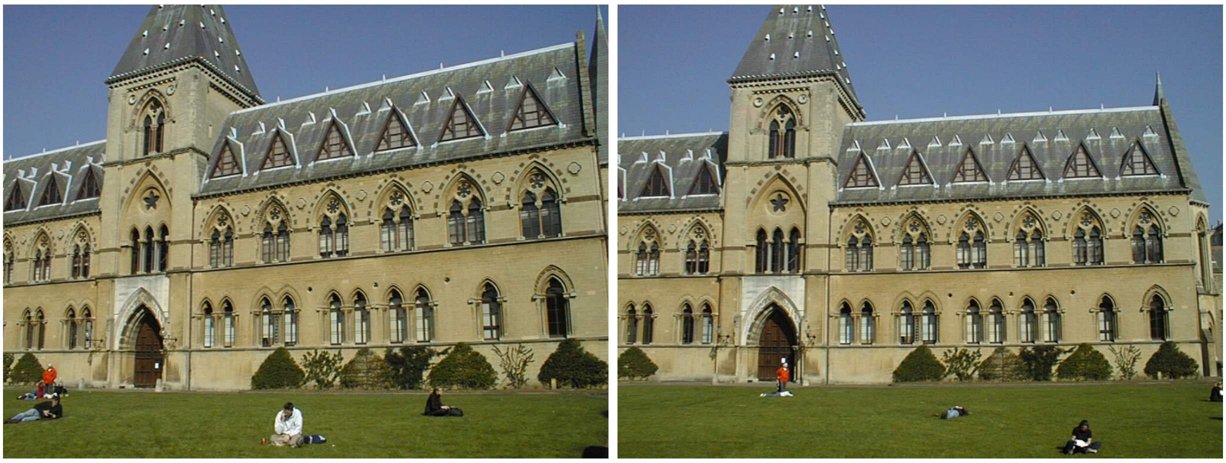

Figure 25: Input image 1 and 2 for pair 3

2.3.1 Harris Corner Detector, $\sigma = 0.8$

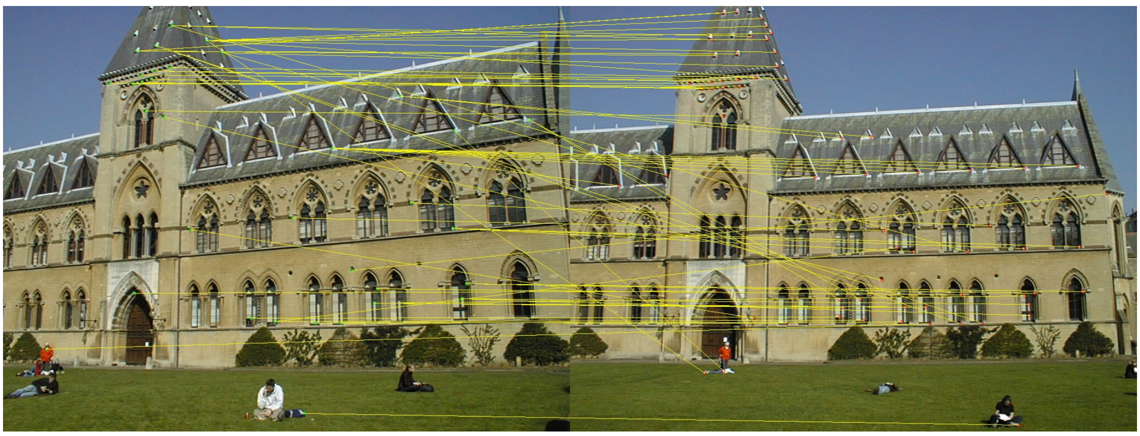

Figure 26: Harris output correspondences using SSD, $\sigma = 0.8$

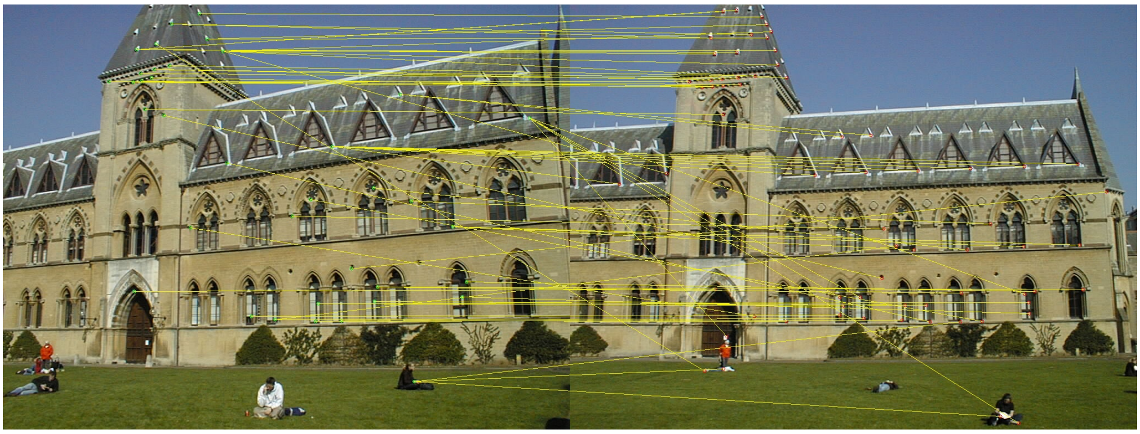

Figure 27: Harris output correspondences using NCC, $\sigma = 0.8$

2.3.2 Harris Corner Detector, $\sigma = 1.2$

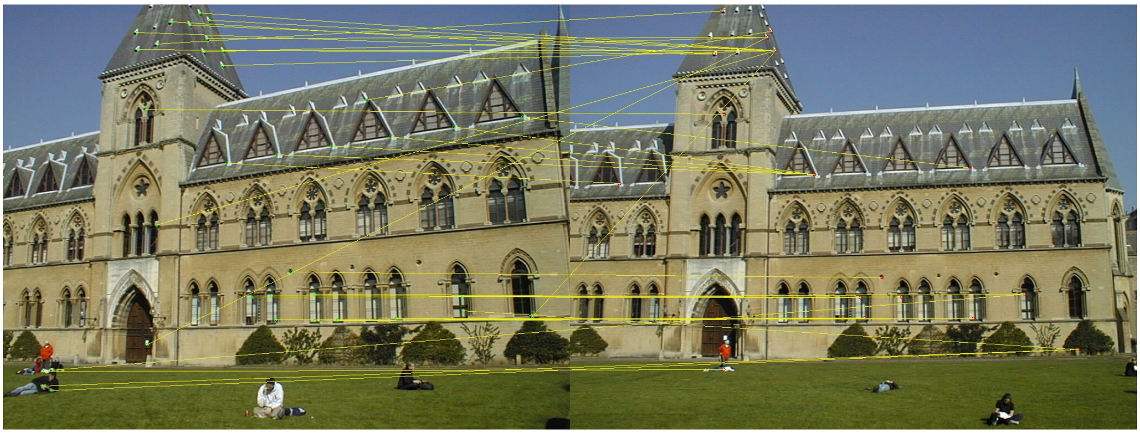

Figure 28: Harris output correspondences using SSD, $\sigma = 1.2$

Figure 29: Harris output correspondences using NCC, $\sigma = 1.2$

2.3.3 Harris Corner Detector, $\sigma = 1.6$

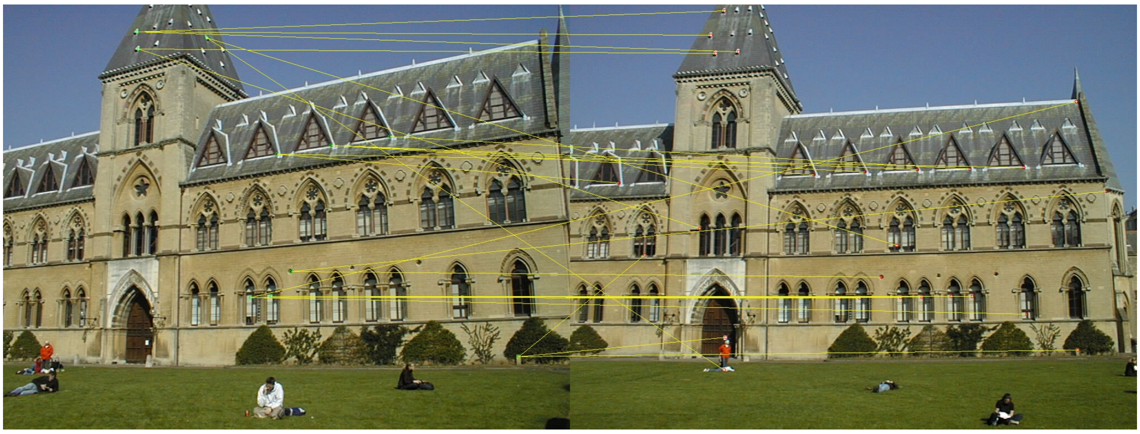

Figure 30: Harris output correspondences using SSD, $\sigma = 1.6$

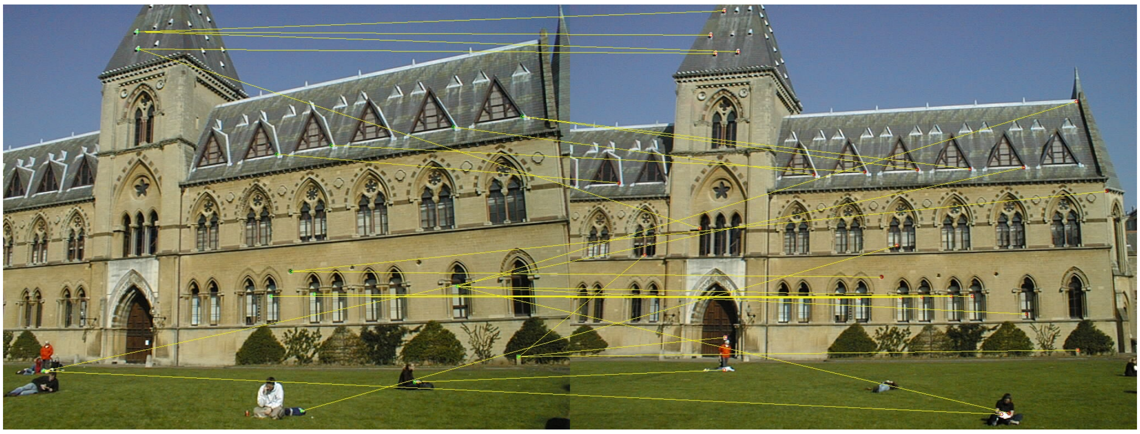

Figure 31: Harris output correspondences using NCC, $\sigma = 1.6$

2.3.4 Harris Corner Detector, $\sigma = 2.0$

Figure 32: Harris output correspondences using SSD, $\sigma = 2.0$

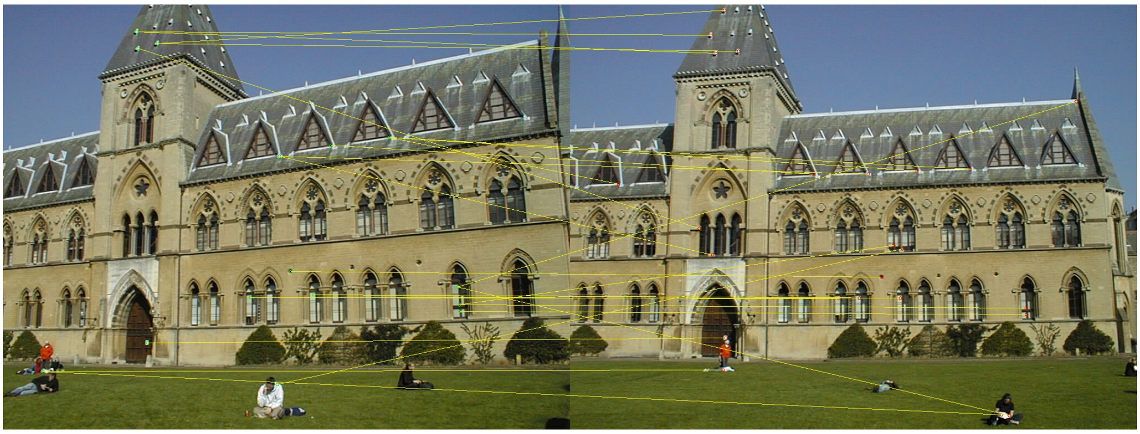

Figure 33: Harris output correspondences using NCC, $\sigma = 2.0$

2.3.5 SIFT

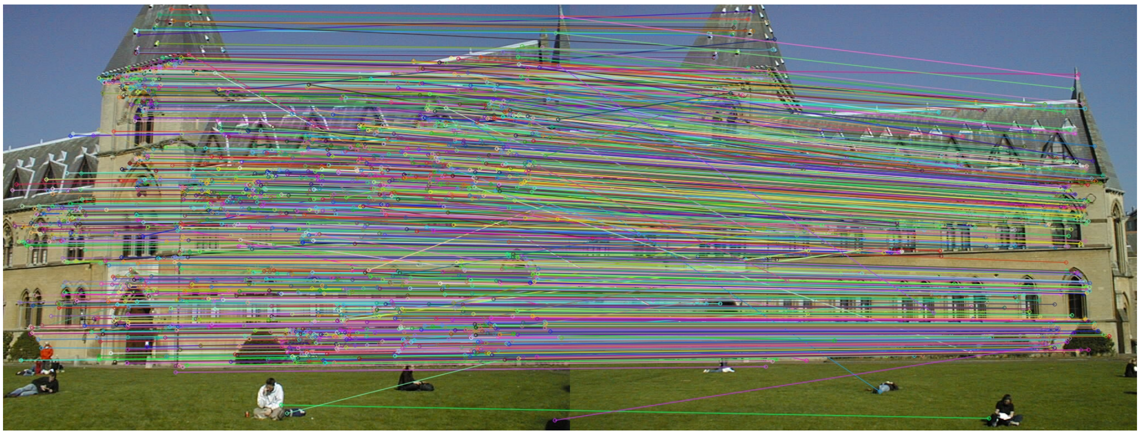

Figure 34: SIFT output correspondences

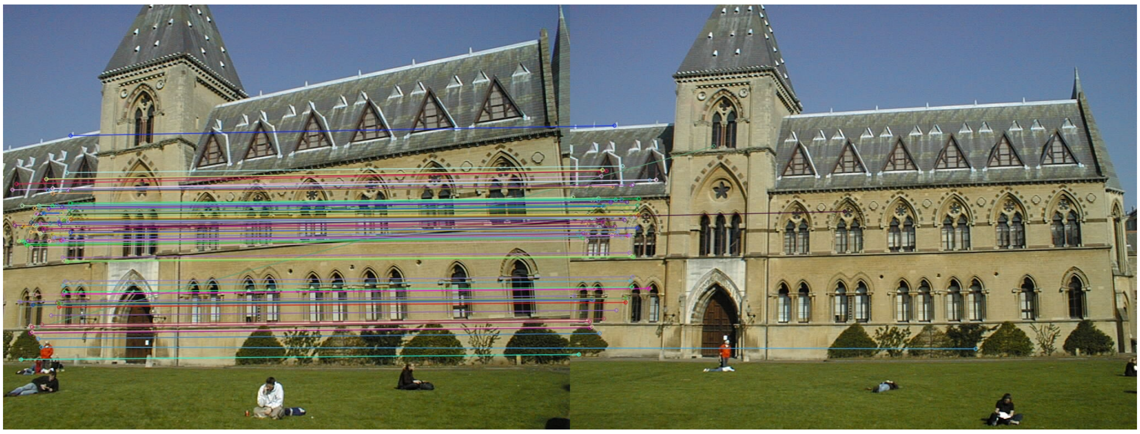

Figure 35: SIFT output correspondences (first 100 pairs)

2.4 Remarks

First, Harris corner detector seems to do a decent job at finding interest points (corners with strong transition) in all three cases. As mentioned before, as σ increases, the images' feature became more "coarse", and the number of interest points detected decreases.

As for matching schemes, both NCC and SSD are able to generate quite a lot correct or close matches, especially when the corners are more distinctive from one another (as in pair 1 and 3) However, for pair 2, where many corners are not as distinctive from each other, both matching schemes' performance are not as good.

In addition, as a more intuitive perception, I feel that SSD provides much larger gaps between correct and incorrect matches, and allows more effective thresholding.

At last, SIFT, as a patented off-the-shelf algorithm, demonstrated much better matching results.

3 Task 2: Custom Images and Results

3.1 Pair 4

Figure 36: Input image 1 and 2 for pair 4

3.1.1 Harris Corner Detector, $\sigma = 0.8$

Figure 37: Harris output correspondences using SSD, $\sigma = 0.8$

Figure 38: Harris output correspondences using NCC, $\sigma = 0.8$

3.1.2 Harris Corner Detector, $\sigma = 1.2$

Figure 39: Harris output correspondences using SSD, $\sigma = 1.2$

Figure 40: Harris output correspondences using NCC, $\sigma = 1.2$

3.1.3 Harris Corner Detector, $\sigma = 1.6$

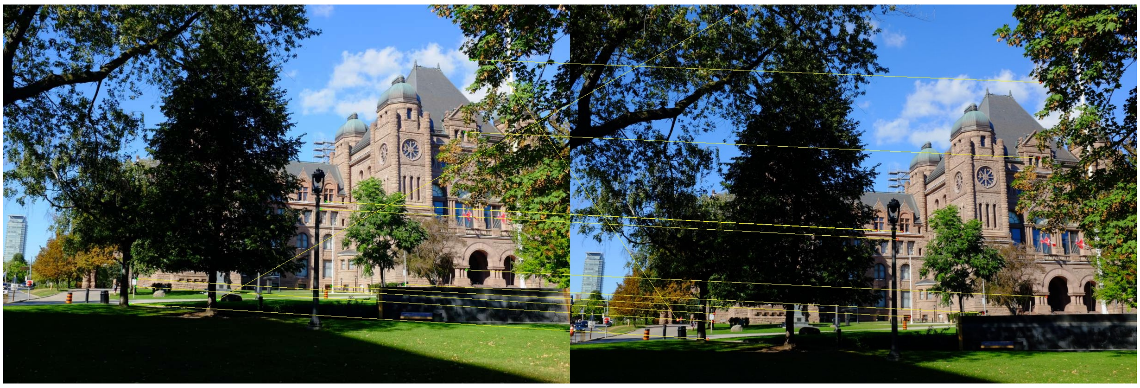

Figure 41: Harris output correspondences using SSD, $\sigma = 1.6$

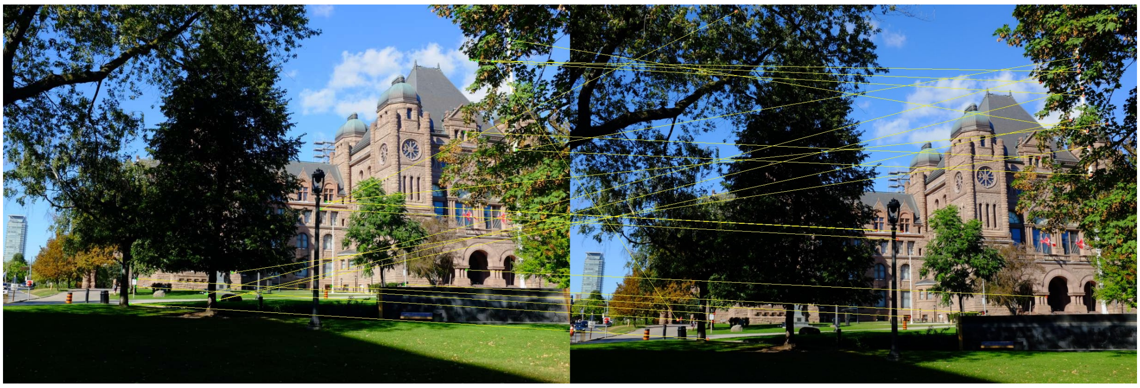

Figure 42: Harris output correspondences using NCC, $\sigma = 1.6$

3.1.4 Harris Corner Detector, $\sigma = 2.0$

Figure 43: Harris output correspondences using SSD, $\sigma = 2.0$

Figure 44: Harris output correspondences using NCC, $\sigma = 2.0$

3.1.5 SIFT

Figure 45: SIFT output correspondences

Figure 46: SIFT output correspondences (first 100 pairs)

3.2 Pair 5

Figure 47: Input image 1 and 2 for pair 4

3.2.1 Harris Corner Detector, $\sigma = 0.8$

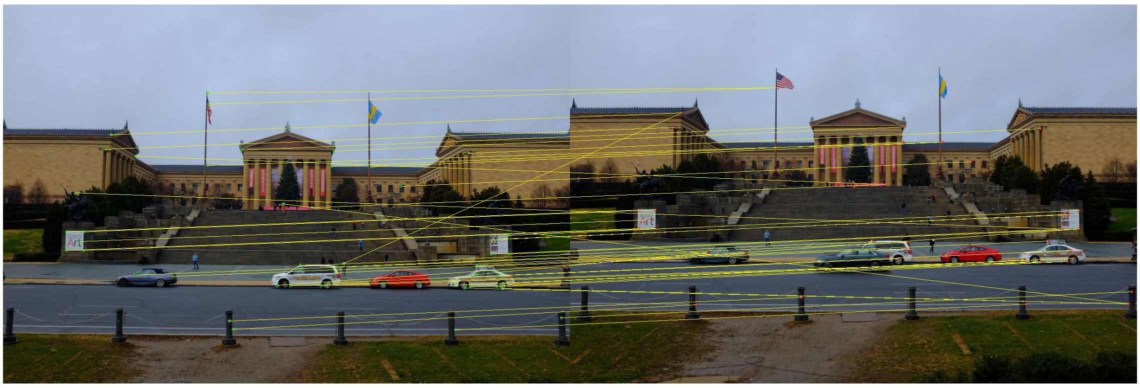

Figure 48: Harris output correspondences using SSD, $\sigma = 0.8$

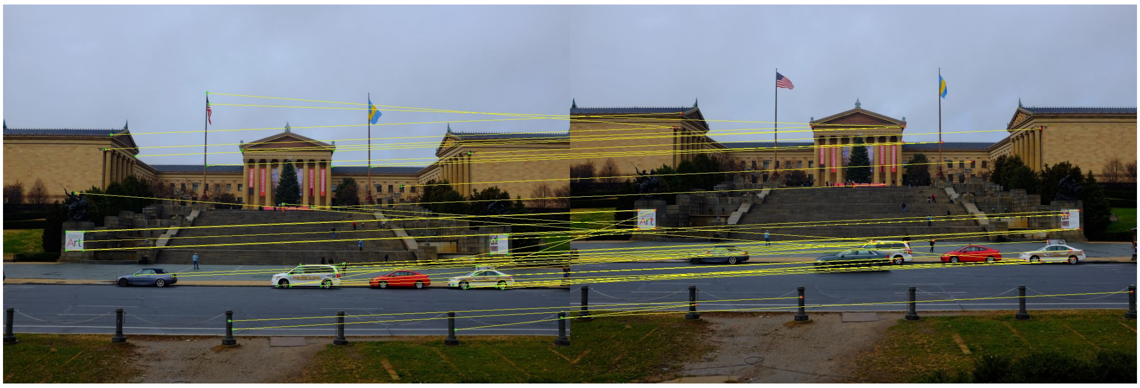

Figure 49: Harris output correspondences using NCC, $\sigma = 0.8$

3.2.2 Harris Corner Detector, $\sigma = 1.2$

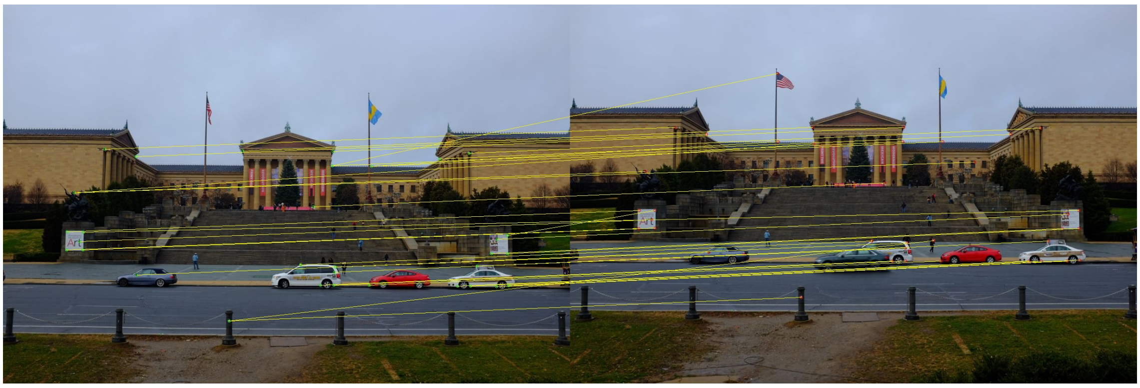

Figure 50: Harris output correspondences using SSD, $\sigma = 1.2$

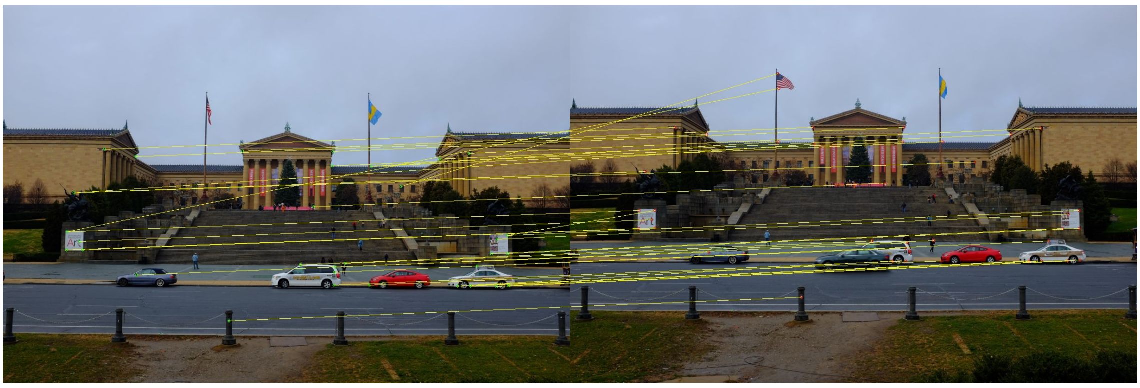

Figure 51: Harris output correspondences using NCC, $\sigma = 1.2$

3.2.3 Harris Corner Detector, $\sigma = 1.6$

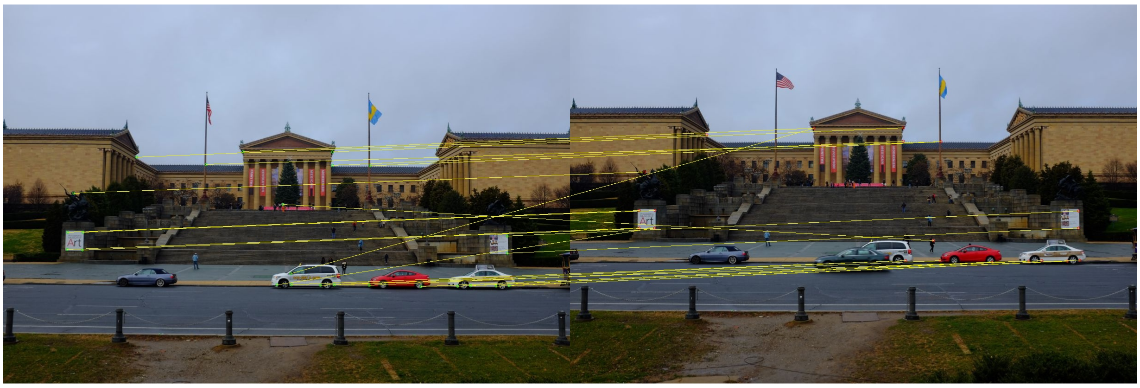

Figure 52: Harris output correspondences using SSD, $\sigma = 1.6$

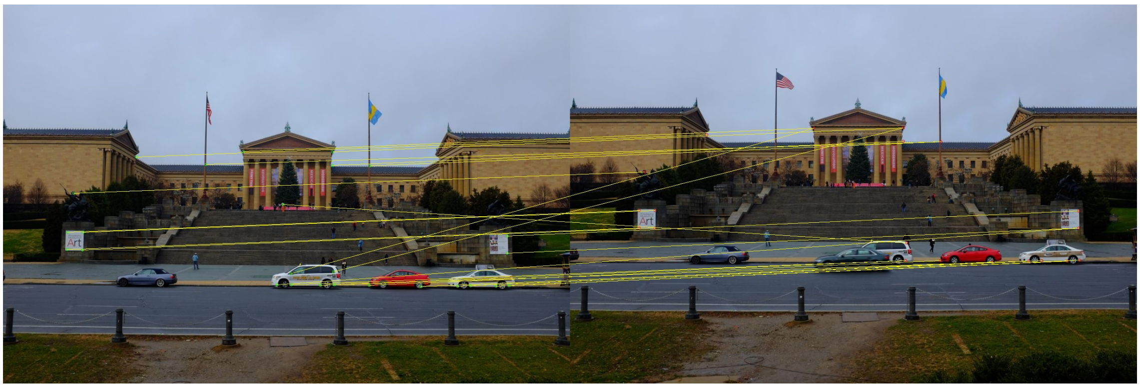

Figure 53: Harris output correspondences using NCC, $\sigma = 1.6$

3.2.4 Harris Corner Detector, $\sigma = 2.0$

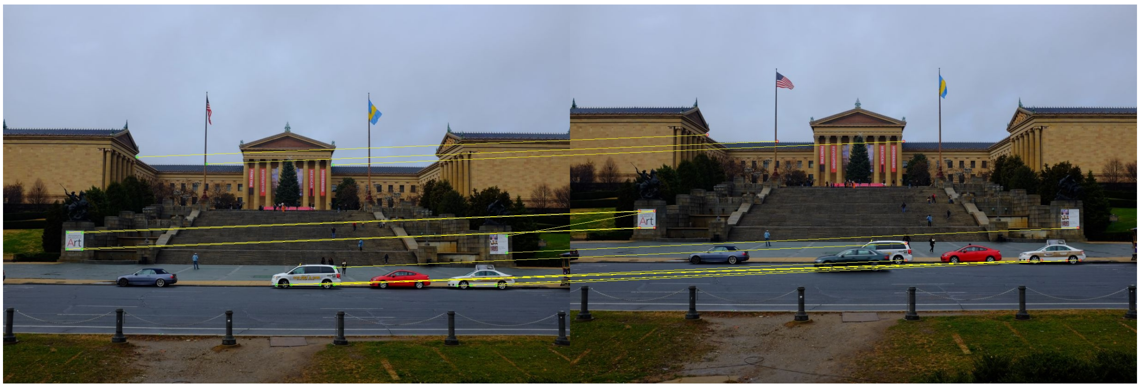

Figure 54: Harris output correspondences using SSD, $\sigma = 2.0$

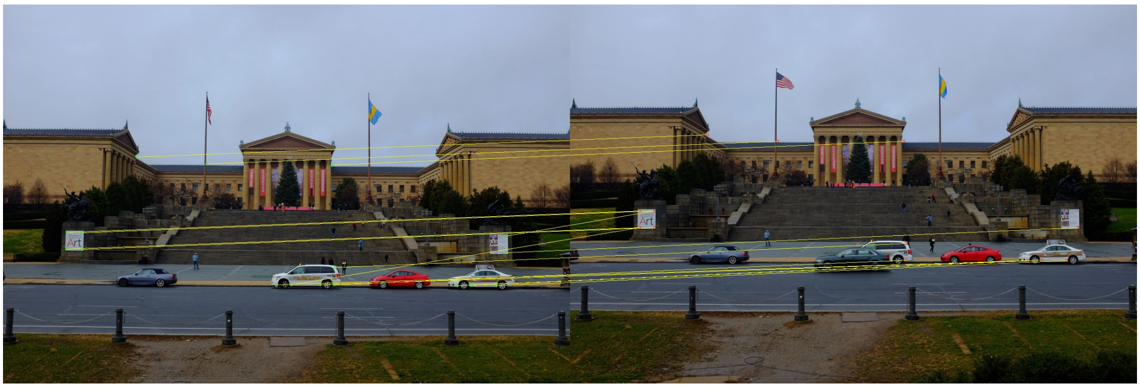

Figure 55: Harris output correspondences using NCC, $\sigma = 2.0$

3.2.5 SIFT

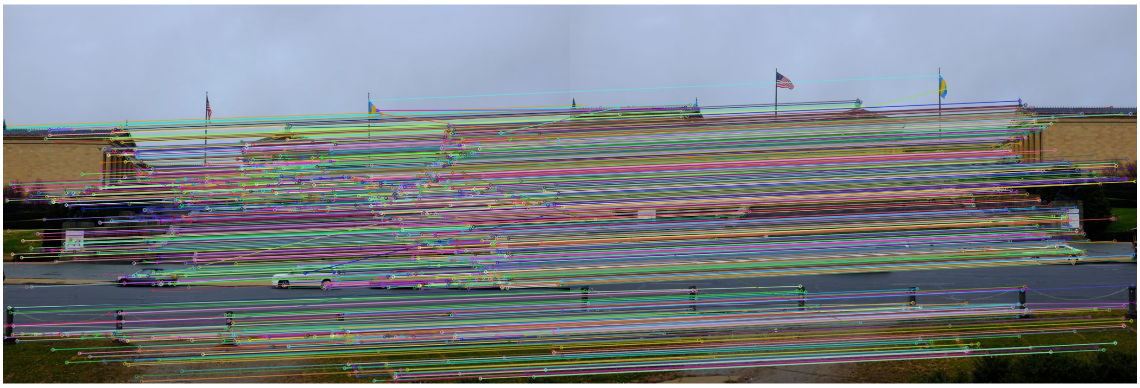

Figure 56: SIFT output correspondences

Figure 57: SIFT output correspondences (first 100 pairs)

3.3 Remarks

Adding the observation from two custom image pairs, one intuitive observation is that source image also has huge impact on the matching result. For example, pair 1 and 5 (custom pair 2) observed great performance across all scales for Harris corner detector, while the same scheme applied to pair 2 and 4 (custom pair 1) struggles to obtain accurate results.

Also, since the structure in pair 5 (Philadelphia Museum of Art) has rather clear corners as well as a clean background, we can also clearly see the effect of scale; with increasing σ, less points were detected, but key points on the building are still constantly detected across scales while obtaining decent matching results.

4 Source Codes

```

import cv2
import numpy as np
import math
import pdb

# ece 661 hw4
# haoyu chen
# chen1562@purdue.edu

def haar_kernel(sigma):
```

```python
  N = int(math.ceil(4*sigma))
  if N%2 == 1:
    N = N+1
  hx = np.ones((N,N))
  hy = np.ones((N,N))

  hx[:,:int(N//2)] = -1
  hy[int(N//2):,:] = -1
  return hx, hy

def harris(img_raw, sigma, save = None):
  if len(img_raw.shape) == 3:
    img = cv2.cvtColor(img_raw, cv2.COLOR_BGR2GRAY)
  else:
    img = img_raw
  img = img / 255
  # normalize image

  hx, hy = haar_kernel(sigma)
  # haar wavelet filter
  # pdb.set_trace()
  dx = cv2.filter2D(img, -1, kernel=hx)
  dy = cv2.filter2D(img, -1, kernel=hy)

  dx_sq = dx * dx
  dy_sq = dy * dy
  dxdy = dx * dy

  N = int(math.ceil(5*sigma))
  if N%2 == 1:
    N = N+1
  kernel_sum = np.ones((N,N))

  sum_dx_sq = cv2.filter2D(dx_sq, -1, kernel=kernel_sum)
  sum_dy_sq = cv2.filter2D(dy_sq, -1, kernel=kernel_sum)
  sum_dxdy = cv2.filter2D(dxdy, -1, kernel=kernel_sum)
  # sum within a window

  trace = sum_dx_sq + sum_dy_sq
  det = (sum_dx_sq * sum_dy_sq) - (sum_dxdy * sum_dxdy)
  # trace and determinant of C for each pixel

  # k = 0.04
  k_tmp = det / (trace**2 + 0.000001)
  k = np.sum(k_tmp) / (img.shape[0]*img.shape[1])
  print(k)
  # adaptive k value


  R = det - k * trace ** 2
  R_thresh = np.sort(R.flatten())[-500]
  # Harris response
  # select top-500 points as threshold

  # pdb.set_trace()
  R_threshed = []
  corner_coord = []
  # Non-maximum suppression + threshold
  for x in range(10, img.shape[1]-10, 1):
    for y in range(10, img.shape[0]-10, 1):
      R_region = R[y-10:y+11, x-10:x+11]
      R_max = np.amax(R_region)
      if R[y,x] == R_max and R_max >= R_thresh:
        R_threshed.append(R_max)
```

```
      corner_coord.append([x,y])
      cv2.circle(img_raw, (x,y), 3, (10,240,10), 2)



  if save != None:
    cv2.imwrite(save + '_harris_'+str(sigma)+'.jpg', img_raw)
  print('number of interest points: ',len(corner_coord))

  return corner_coord

def compute_distance(img1, img2, idx1, idx2, mode = 'SSD', M=21):
  # by default, I assume the input images are already padded
  patch1 = img1[idx1[1]:idx1[1]+M, idx1[0]:idx1[0]+M].flatten()
  # pixel intensities in a MxM patch from image 1
  patch2 = img2[idx2[1]:idx2[1]+M, idx2[0]:idx2[0]+M].flatten()
  if mode == 'SSD':
    diff = patch1 - patch2
    distance = np.sum(diff ** 2)
  elif mode == 'NCC':
    mu1 = np.mean(patch1)
    mu2 = np.mean(patch2)
    num = np.sum((patch1-mu1)*(patch2-mu2))
    denom = np.sqrt(np.sum((patch1-mu1)**2) * np.sum((patch2-mu2)**2))
    distance = 1 - (num / denom)
    # NCC's range is [-1,1], with 1 being the closest match
    # in order to apply the same "smaller distance = closer match" logic
    # I use d = 1-NCC, hence smaller = better
  return distance


def harris_matching(img1_raw, img2_raw, idx1, idx2, mode = 'SSD', N=10, save_name = None):

  if len(img1_raw.shape) == 3:
    img1 = cv2.cvtColor(img1_raw, cv2.COLOR_BGR2GRAY)
  else:
    img1 = img1_raw
  if len(img2_raw.shape) == 3:
    img2 = cv2.cvtColor(img2_raw, cv2.COLOR_BGR2GRAY)
  else:
    img2 = img2_raw

  img1 = img1 / 255
  img2 = img2 / 255


  white = (1,1,1)
  img1= cv2.copyMakeBorder(img1,N,N,N,N,cv2.BORDER_CONSTANT,value=white)
  img2= cv2.copyMakeBorder(img2,N,N,N,N,cv2.BORDER_CONSTANT,value=white)
  # padding the borders


  if len(idx1) <= len(idx2):
    w = img1_raw.shape[1]
    comb = np.concatenate((img1_raw, img2_raw), axis=1)
    for coord1 in idx1:
      d_tmp = []
      for coord2 in idx2:
        distance = compute_distance(img1, img2, coord1, coord2, mode=mode, M=int(N*2+1))
        d_tmp.append(distance)
      # pdb.set_trace()
      best_match = idx2[np.argsort(d_tmp)[0]]
```

```
      # print(np.min(d_tmp))

      # visualize if the distance is within reasonable range
      if np.min(d_tmp) < 25:
        pt1 = tuple(coord1)
        pt2 = (best_match[0]+w,best_match[1])

        cv2.circle(comb, pt1, 3, (10,240,10), 1)
        cv2.circle(comb, pt2, 3, (10,10,240), 1)
        cv2.line(comb, pt1, pt2, (10,240,240), 1)
  else:
    w = img2_raw.shape[1]
    comb = np.concatenate((img1_raw, img2_raw), axis=1)
    for coord2 in idx2:
      d_tmp = []
      for coord1 in idx1:
        distance = compute_distance(img1, img2, coord1, coord2, mode=mode, M=int(N*2+1))
        d_tmp.append(distance)
      best_match = idx1[np.argsort(d_tmp)[0]]
      # print(np.min(d_tmp))

      # visualize if the distance is within reasonable range
      if np.min(d_tmp) < 25:
        pt1 = (best_match[0],best_match[1])
        pt2 = (coord2[0]+w,coord2[1])

        cv2.circle(comb, pt1, 3, (10,240,10), 1)
        cv2.circle(comb, pt2, 3, (10,10,240), 1)
        cv2.line(comb, pt1, pt2, (10,240,240), 1)


  cv2.imwrite(save_name+'.jpg',comb)

def sift_pair(img1_raw, img2_raw, name):
  # need to use older version to bypass patent issues
  # opencv-contrib-python == 3.4.2.16
  if len(img1_raw.shape) == 3:
    img1 = cv2.cvtColor(img1_raw, cv2.COLOR_BGR2GRAY)
  else:
    img1 = img1_raw
  if len(img2_raw.shape) == 3:
    img2 = cv2.cvtColor(img2_raw, cv2.COLOR_BGR2GRAY)
  else:
    img2 = img2_raw

  sift = cv2.xfeatures2d.SIFT_create()
  kp1, des1 = sift.detectAndCompute(img1, None)
  kp2, des2 = sift.detectAndCompute(img2, None)
  bf = cv2.BFMatcher()
  matches = bf.knnMatch(des1,des2, k=2)
  good = []
  for m,n in matches:
    if m.distance < 0.75*n.distance:
      good.append([m])
  comb = np.concatenate((img1_raw, img2_raw), axis=1)
  cv2.drawMatchesKnn(img1_raw,kp1,img2_raw,kp2,good[0:100],comb,flags=2)
  cv2.imwrite(name+'.jpg',comb)




if __name__ == '__main__':
  pair_number = '1'
```

```
sigma = 0.8
# sigma choices:
path1 = 'pair'+pair_number+'/1'
path2 = 'pair'+pair_number+'/2'
img1 = cv2.imread(path1+ '.JPG')
img2 = cv2.imread(path2+ '.JPG')
h1,w1,_ = img1.shape
h2,w2,_ = img2.shape
if h1 > h2:
  img1 = cv2.resize(img1, (w2,h2), cv2.INTER_AREA)
elif h1 < h2:
  img2 = cv2.resize(img2, (w1,h1), cv2.INTER_AREA)

img1_cp = img1.copy()
img2_cp = img2.copy()
# idx1 = harris(img1_cp, sigma = sigma, save = path1)
# idx2 = harris(img2_cp, sigma = sigma, save = path2)

# Part 1. Harris

# idx1 = harris(img1_cp, sigma = sigma)
# idx2 = harris(img2_cp, sigma = sigma)
# save_name = 'pair'+pair_number+'/combined_SSD_'+str(sigma)
# # save_name = 'pair'+pair_number+'/combined_NCC_'+str(sigma)

# harris_matching(img1, img2, idx1, idx2, mode = 'NCC', N=10, save_name=save_name)


# Part 2. SIFT
save_name = 'pair'+pair_number+'/sift_100'

sift_pair(img1, img2, save_name)
```