ECE 661 Computer Vision: HW4

Qiuchen Zhai
qzhai@purdue.edu

September 29, 2020

1 Theory Question

1.1 The theoretical reason for why the LoG of and image can be computed as DoG

Let $f(x,y)$ denotes the image. Then its σ-smoothed version is given by

$$ff\left(x,y,\sigma\right)=\iint_{-\infty}^{\infty} f\left(x^{'},y^{'}\right) g\left(x-x^{'},y-y^{'}\right) dx^{'}dy^{'} \tag{1}$$

where $g\left(x,y\right)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$.

The LoG (Laplacian of Gaussian) of image is

$$\bigtriangledown^2 ff\left(x,y,\sigma\right)=f\left(x,y\right)*h\left(x,y,\sigma\right) \tag{2}$$

where $h\left(x,y,\sigma\right)=-\frac{1}{2\pi\sigma^4}\left(2-\frac{x^2+y^2}{\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$.

We can write the difference of Gaussian-smoothed versions of $f(x,y)$ as

$$\frac{\partial}{\partial\sigma}ff\left(x,y,\sigma\right)=\iint f\left(x^{'},y^{'}\right)\frac{\partial}{\partial\sigma}\{\frac{1}{2\pi\sigma^2}e^{-\frac{(x-x^{'})^2+(y-y^{'})^2}{2\sigma^2}}\}dx^{'}dy^{'} \tag{3}$$

$$=\iint f\left(x^{'},y^{'}\right)\left[-\frac{1}{\pi\sigma^3}+\frac{1}{2\pi\sigma^2}\left(-[(x-x^{'})^2+(y-y')^2]\right)\frac{-2}{2\sigma^3}\right]e^{-\frac{(x-x^{'})^2+(y-y^{'})^2}{2\sigma^2}}dx^{'}dy^{'} \tag{4}$$

$$=-\frac{4}{2\pi\sigma^4}\iint f\left(x^{'},y^{'}\right)\left[2-\frac{(x-x^{'})^2+(y-y^{'})^2}{2\sigma^2}\right]e^{-\frac{(x-x^{'})^2+(y-y^{'})^2}{2\sigma^2}}dx^{'}dy^{'} \tag{5}$$

$$=\sigma f(x,y)*h(x,y) \tag{6}$$

$$=\sigma\bigtriangledown^2 ff\left(x,y,\sigma\right) \tag{7}$$

Therefore, the LoG of an image can be approximated by the Difference-of-Gaussian (DoG).

1.2 The reason why computing the LoG of an image as DoG is computationally more efficient

The first reason why computing the LoG is more efficient than DoG is that, the Gaussian $g\left(x,y\right)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$ is separable in x and y while the LoG operator $h\left(x,y,\sigma\right)=-\frac{1}{2\pi\sigma^4}\left(2-\frac{x^2+y^2}{\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$ is not. Therefore, the 2-D application of DoG could be carried out by two applications of 1-D smoothing along x and y respectively.

The second reason is that, the 1-D Gaussian smoothing operator has smaller width of the central lobe than 2-D operator. For the same value of σ, the radius of central lobe for 1-D operator is approximately 40% less than the 2-D operator. The smaller width yields a smaller size operator.

Therefore, DoG is more computationally efficient considering the above two reasons.

2 Extract Interest Points with Harris Corner Detector

First, let d_x and d_y be the Haar filters at scale σ along x- and y- directions respectively. Then the filter size is determined by the minimum even integer greater than 4σ. For example, the filter has size 6x6 if $\sigma = 1.2$,

$$d_x = \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{pmatrix}, d_y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix} \tag{8}$$

And the filter size would be 8x8 if $\sigma = 1.6$,

$$d_x = \begin{pmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \end{pmatrix}, d_y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix} \tag{9}$$

Now we form the following matrix in a $5\sigma \times 5\sigma$ neighborhood of the pixel where we'd like to check the presence of a corner.

$$\mathbf{C} = \begin{pmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{pmatrix} \tag{10}$$

The window size of neighborhood pixels is determined by nearest odd number to 5σ to make sure a pixel is at center. The two-by-two matrix has full rank at a genuine corner except for the presence of a straight edge. For example, all d_x's will be zeros and the rank of matrix $\mathbf{C}$ will be reduced to zero if the edge is parallel to the x-axis.

Let λ_1 and λ_2 are two eigenvalues of $\mathbf{C}$. Assuming $\lambda_1 \geq \lambda_2$, we place the threshold on the ratio $r = \lambda_2/\lambda_1$ to detect interest points. For compute efficiency, we compute the ratio by the trace and determinants instead of taking eigen-decomposition of matrix $\mathbf{C}$.

$$r = det(C) - k(Tr(C))^2 \tag{11}$$

where $Tr(C) = \sum d_x^2 + \sum d_y^2 = \lambda_1 + \lambda_2$, $det(C) = \sum d_x^2 \sum d_y^2 - (\sum d_x d_y)^2 = \lambda_1\lambda_2$, and $k = 0.05$ is an empirically-determined constant parameter in this experiment.

Once finishing the calculation of the Harris Corner detector response, a threshold is required to filter out the pixels with small or negative corner response. And we also need to identify the points with local maxima values to extract the points with strong corner responses. Therefore, the non-maxima suppression is utilized to retain the local maxima interest points in neighboring $K \times K$ (the value of K is dependent to the image pairs) area.

3 Establishing Correspondences Between Image Pairs

After extracting interest points from the two images of same scene in different views, we need to establish correspondences between the image pairs. Using the SSD and NCC, the correspondences could be built by directly comparing the gray levels.

3.1 SSD: sum of Squared Differences

For each pair of interest points, we compare the gray levels within a $(M+1)\times(M+1)$ window around the corner pixels in two images $f_1(x, y)$ and $f_2(x, y)$.

$$SSD = \sum_i \sum_j \|f_1(i,j) - f_2(i,j)\|^2 \tag{12}$$

3.2 NCC: Normalized Cross Correlation

While using NCC metric for such comparison, let m_1 and m_2 denotes the mean values of gray scales within the $(M+1)\times(M+1)$ window around the corner pixels.

$$NCC = \frac{\sum\sum(f_1(i,j) - m_1)(f_2(i,j) - m_2)}{\sqrt{[\sum\sum(f_1(i,j) - m_1)^2][\sum\sum(f_2(i,j) - m_2)^2]}} \tag{13}$$

The computation of NCC metric returns values ranging from -1 to 1. NCC=1 implies the two interest points matches perfectly and vice versa if NCC=-1. In the experiments, the window size of neighborhood is decided by the value M and the value is set to be $M = 25$.

4 SIFT : Scale Invariant Feature Transform

The following 5 steps describe how to use SIFT features to extract Interest Points and how to create SIFT descriptor.

1. **Step 1**: Find all the local extrema in the Dog (Difference of Gaussian) pyramid.
   Let $D(x, y, \sigma)$ denote the DoG values at scale σ. By comparing the 8 points in the immediate 3x3 neighborhood at the same scale σ, the 9 points in the 3x3 neighborhood in the DoG that at the next level and the 9 points in the 3x3 neighborhoood in the DoG that at the level below, we could find the points that are either locally maximum or locally miminum in the space (x, y, σ)

2. **Step 2**: Find location of the extremum.
   The individual discrete DoG points represent increasingly coarse sampling of the original image as σ increase. Therefore, an extremum may not point directly to a pixel representing a SIFT point in the original image, especially when you go from a lower octave to the higher octave. To localize the extremum with "sub-pixel" accuracy, we estimate the second-order derivatives of the DoG values $D(x, y, \sigma)$ using the following expansion

$$D(\vec{x}) \cong D(\vec{x_0}) + J^T(\vec{x_0})\vec{x} + \frac{1}{2}\vec{x}^T H(\vec{x_0})\vec{x} \tag{14}$$

where $\vec{x}$ is the incremental deviation from $\vec{x}_0$, J is the gradient vector at $\vec{x}_0$ such that $J(\vec{x}_0) = (\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial \sigma})^T$, H is the Hessian at $\vec{x}_0$ that $H(\vec{x}_0) = \begin{pmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial \sigma} \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial \sigma} \\ \frac{\partial^2 D}{\partial \sigma \partial x} & \frac{\partial^2 D}{\partial \sigma \partial y} & \frac{\partial^2 D}{\partial \sigma^2} \end{pmatrix}$.

The true extremum must satisfy $\frac{\partial D(\vec{x})}{\partial \vec{x}} = 0$ which yields

$$0 \approx J(\vec{x}_0) + H(\vec{x}_0)\vec{x} \tag{15}$$

Thus the true location is given by

$$\vec{x} = -H(\vec{x}_0) \cdot J(\vec{x}_0) \tag{16}$$

3. **Step 3**: Filter the extrema by thresholding.
   The poor local extrema are rejected by thresholding $|D(\vec{x})|$ at the locations $\vec{x} = (x, y, \sigma)^T$ of the extremums through $|D(\vec{x})| < 0.03$.

4. **Step 4**: Associate a 'dominant local orientation' with each extremum.
   At each point in a $K \times K$ neighborhood around the extremum, the gradient magnitude and the gradient orientation are given by

$$m(x,y) = \sqrt{|ff(x+1,y,\sigma) - ff(x,y,\sigma)|^2 + |ff(x,y+1,\sigma) - ff(x,y,\sigma)|^2} \tag{17}$$

$$\theta(x,y) = arctan\frac{ff(x,y+1,\sigma) - ff(x,y,\sigma)}{ff(x+1,y,\sigma) - ff(x,y,\sigma)} \tag{18}$$

   Then we build a histogram of $\theta(x,y)$ values using 36 bins after weighting $\theta(x,y)$ with $m(x,y)$. The bin corresponding to the histogram peak gives us the dominant local orientation.

5. **Step 5**: Create SIFT descriptor.
   At the scale of the extremum, we divide the 16x16 neighborhood of the extremum point into 4x4 cells with each cell consisting of a 4x4 block of points. To reduce the importance of the points that are far away from extremum, the magnitudes of the gradients in the 16x16 neighborhood is weighted by a Gaussian with σ equal to the half the width of the neighborhood. Then an 8-bin orientation histogram is calculated from the gradient-magnitude-weighted values of $\theta(x,y)$ at the 16 pixels in each cell. At each retained extremum in the DoG pyramid, the SIFT descriptor ends up with a 128 dimensional vector by stringing together the 16 8-bin histograms. Finally, the length of the 128-element descriptor is normalized to unity in order to make it invariant to illumination conditions.

5 Results

5.1 First pair of images

5.1.1 Extracted Interest Points

Figure 1: Corners detection with $\sigma = 0.8$

Figure 2: Corners detection with $\sigma = 1.2$

Figure 3: Corners detection with $\sigma = 1.6$

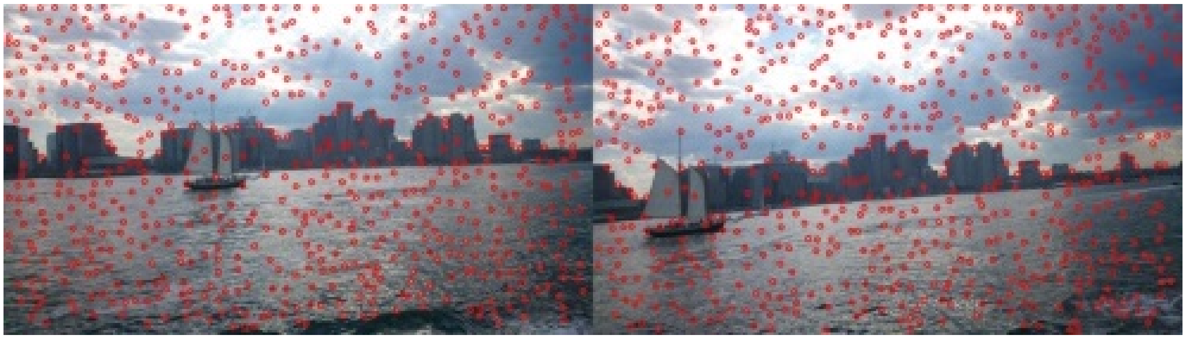

Figure 4: Corners detection with $\sigma = 2.2$

5.1.2 Corners Correspondences Based on SSD

Figure 5: Image pair 1 with $\sigma = 0.8$ based on SSD

Figure 6: Image pair 1 with $\sigma = 1.2$ based on SSD

Figure 7: Image pair 1 with $\sigma = 1.6$ based on SSD

Figure 8: Image pair 1 with $\sigma = 2.2$ based on SSD

5.1.3 Corners Correspondences Based on NCC

Figure 9: Image pair 1 with $\sigma = 0.8$ based on NCC

Figure 10: Image pair 1 with $\sigma = 1.2$ based on NCC

Figure 11: Image pair 1 with $\sigma = 1.6$ based on NCC

Figure 12: Image pair 1 with $\sigma = 2.2$ based on NCC

5.1.4 Corners correspondences using SIFT

Figure 13: Image pair 1 using SIFT features

5.2 Second pair of images

5.2.1 Extracted Interest Points

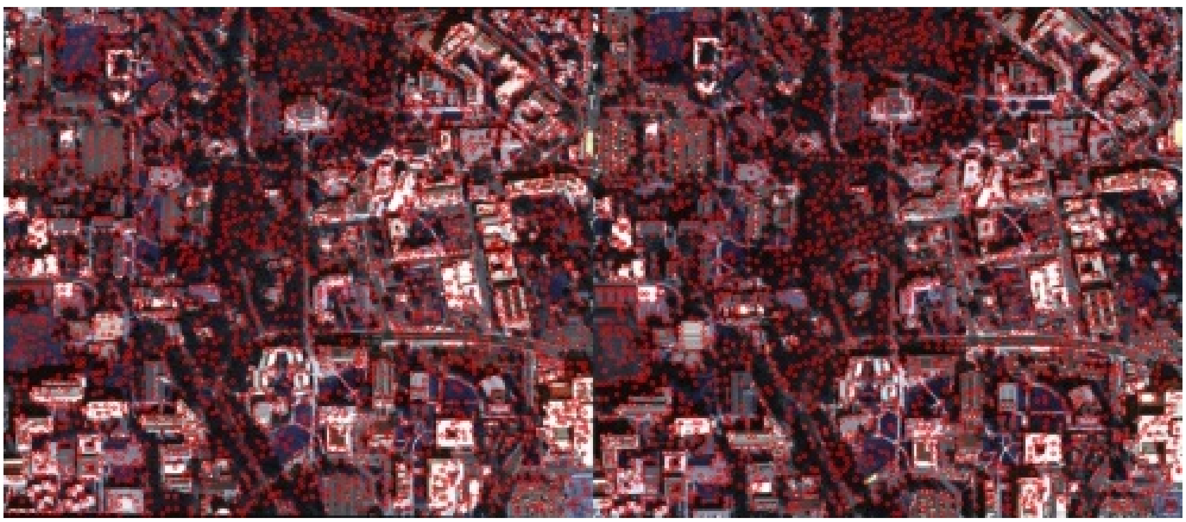

Figure 14: Corners detection with $\sigma = 0.8$

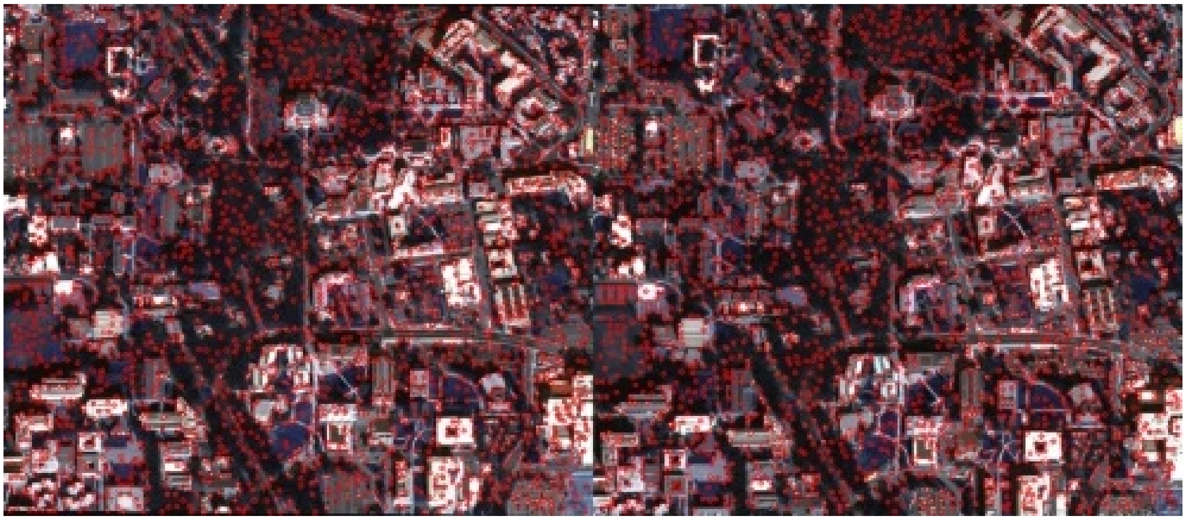

Figure 15: Corners detection with $\sigma = 1.2$

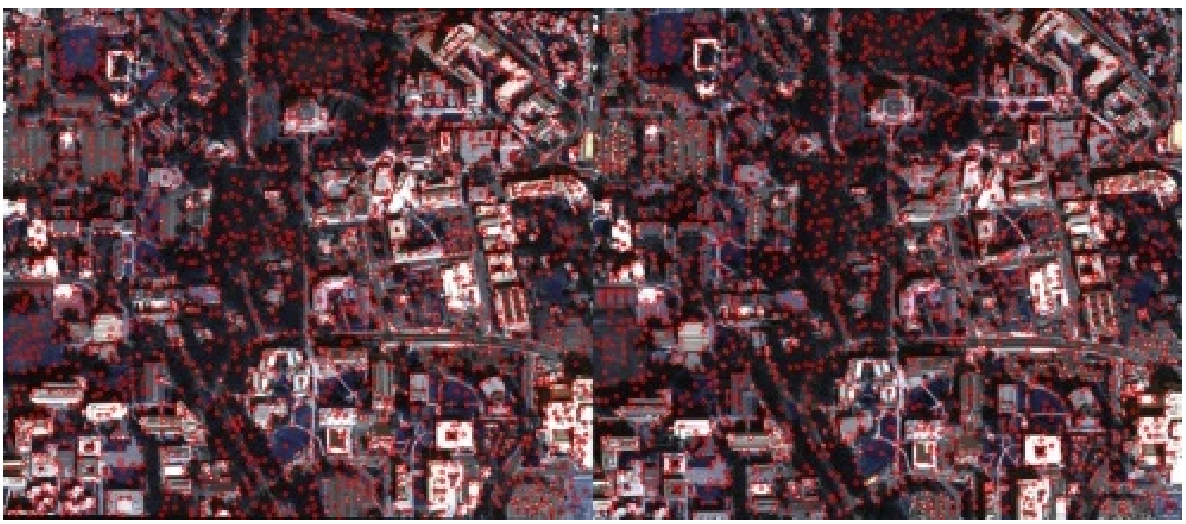

Figure 16: Corners detection with $\sigma = 1.6$

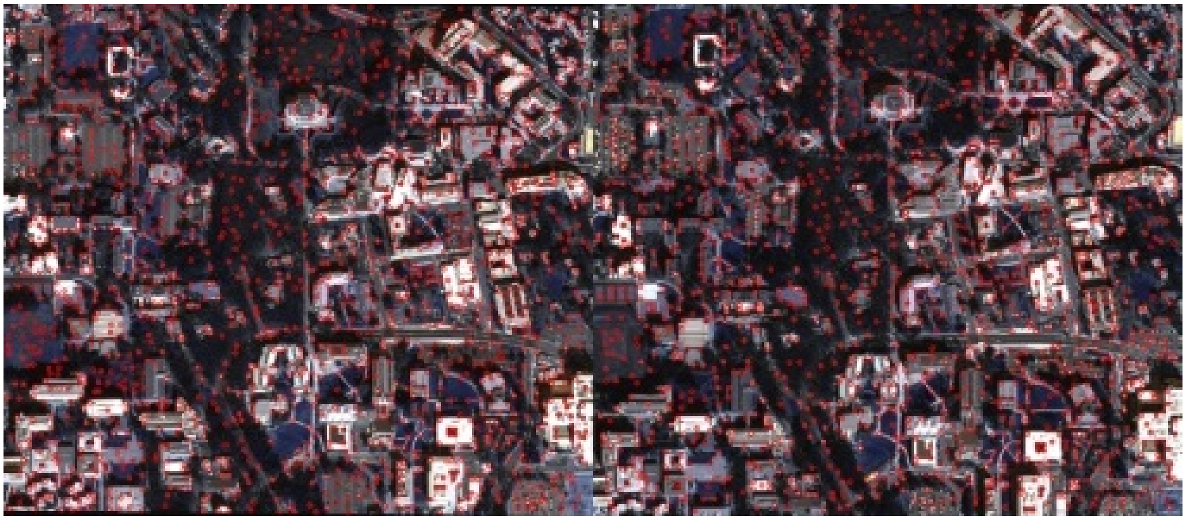

Figure 17: Corners detection with $\sigma = 2.2$

5.2.2 Corners Correspondences Based on SSD

Figure 18: Image pair 2 with $\sigma = 0.8$ based on SSD

Figure 19: Image pair 2 with $\sigma = 1.2$ based on SSD

Figure 20: Image pair 2 with $\sigma = 1.6$ based on SSD

Figure 21: Image pair 2 with $\sigma = 2.2$ based on SSD

5.2.3 Corners Correspondences Based on NCC

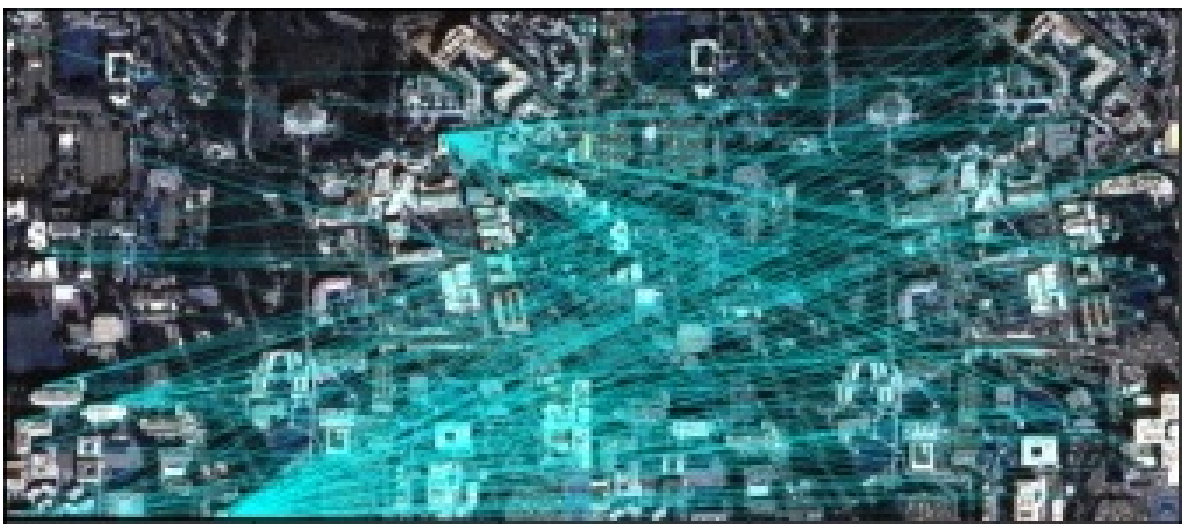

Figure 22: Image pair 2 with $\sigma = 0.8$ based on NCC

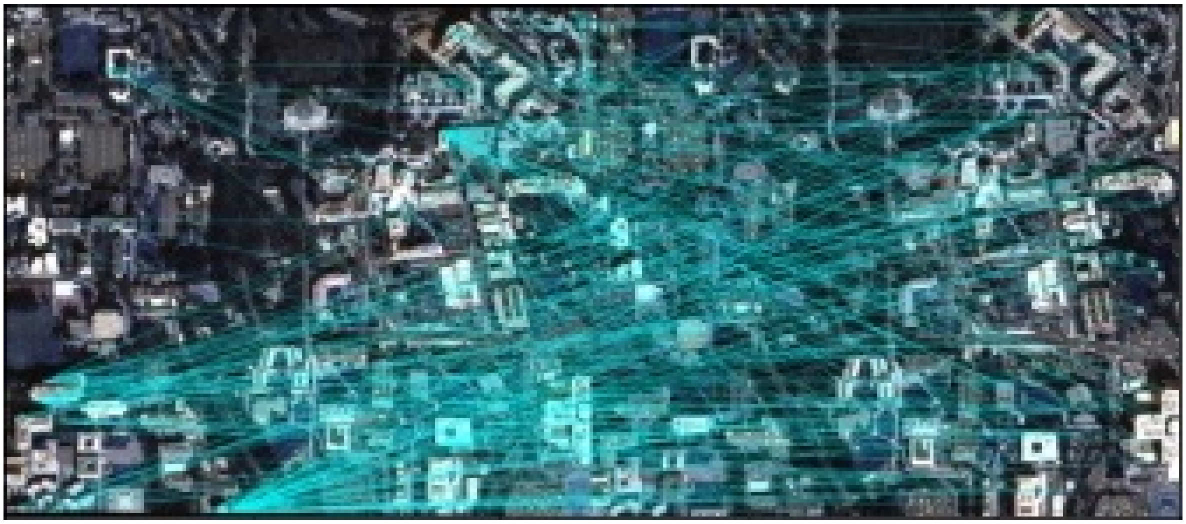

Figure 23: Image pair 2 with $\sigma = 1.2$ based on NCC

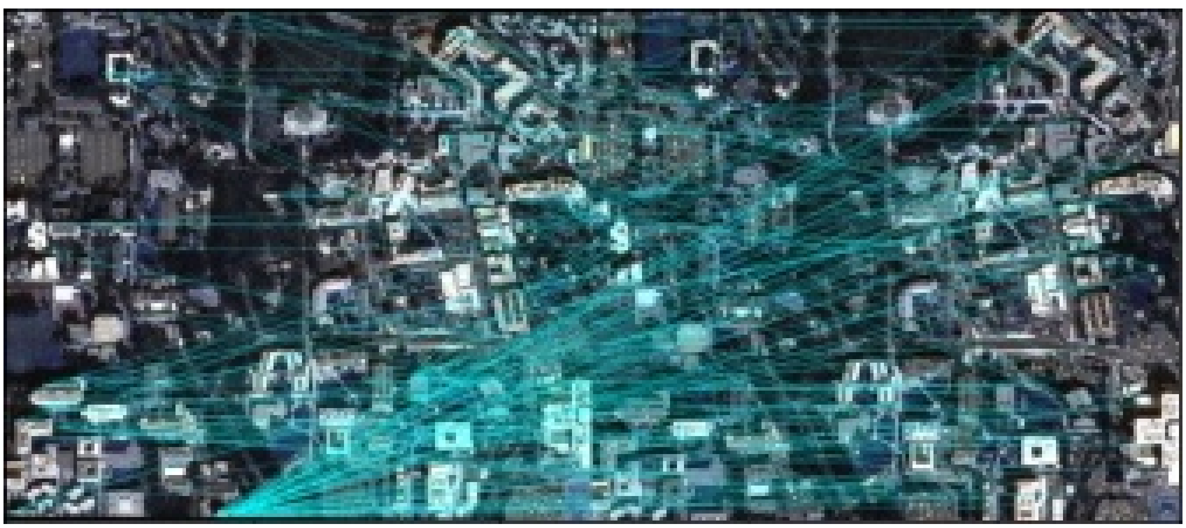

Figure 24: Image pair 2 with $\sigma = 1.6$ based on NCC

Figure 25: Image pair 2 with $\sigma = 2.2$ based on NCC

5.2.4 Corners correspondences using SIFT

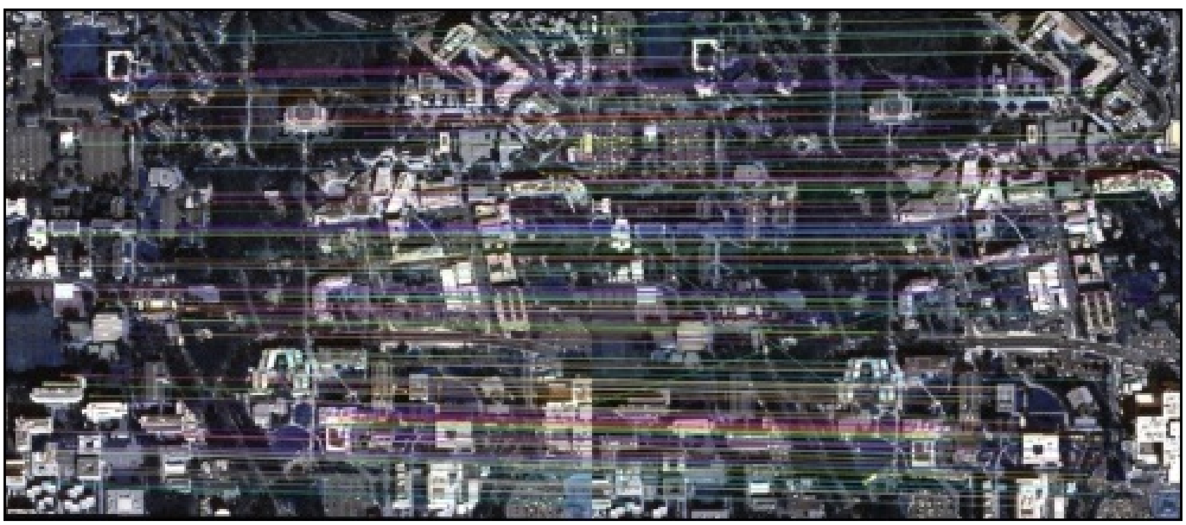

Figure 26: Image pair 2 using SIFT features

5.3 Third pair of images

5.3.1 Extracted Interest Points

Figure 27: Corners detection with $\sigma = 0.8$

Figure 28: Corners detection with $\sigma = 1.2$

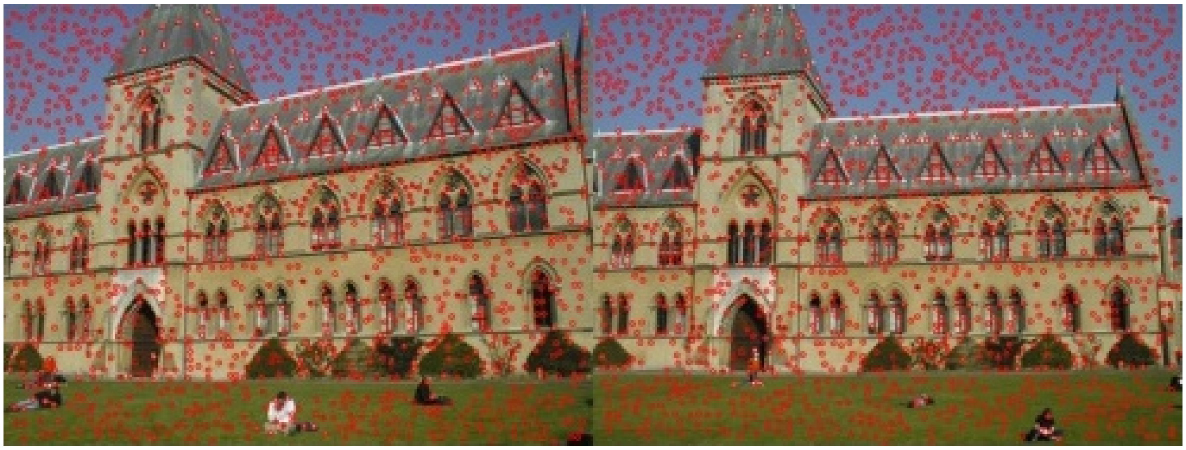

Figure 29: Corners detection with $\sigma = 1.6$

Figure 30: Corners detection with $\sigma = 2.2$

5.3.2 Corners Correspondences Based on SSD

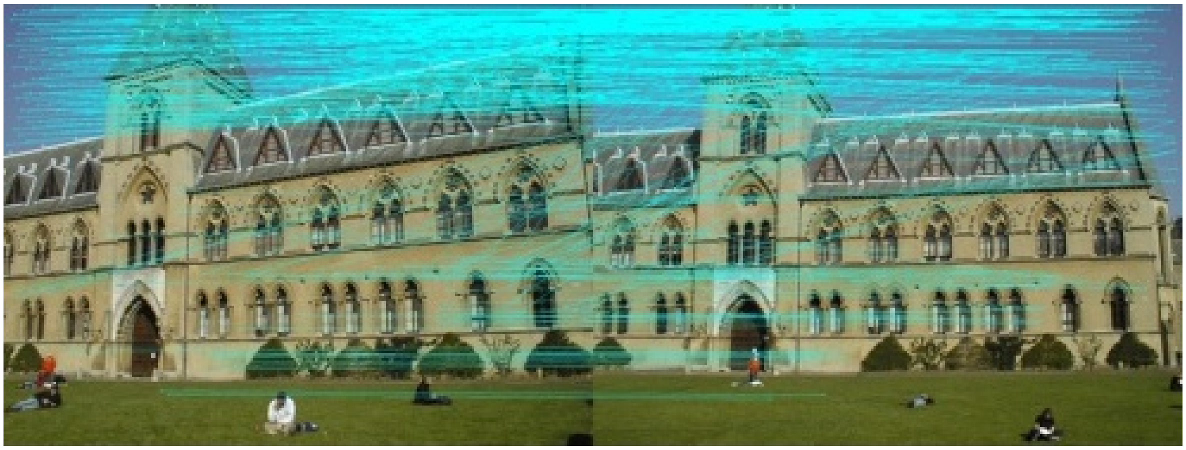

Figure 31: Image pair 3 with $\sigma = 0.8$ based on SSD

Figure 32: Image pair 3 with $\sigma = 1.2$ based on SSD

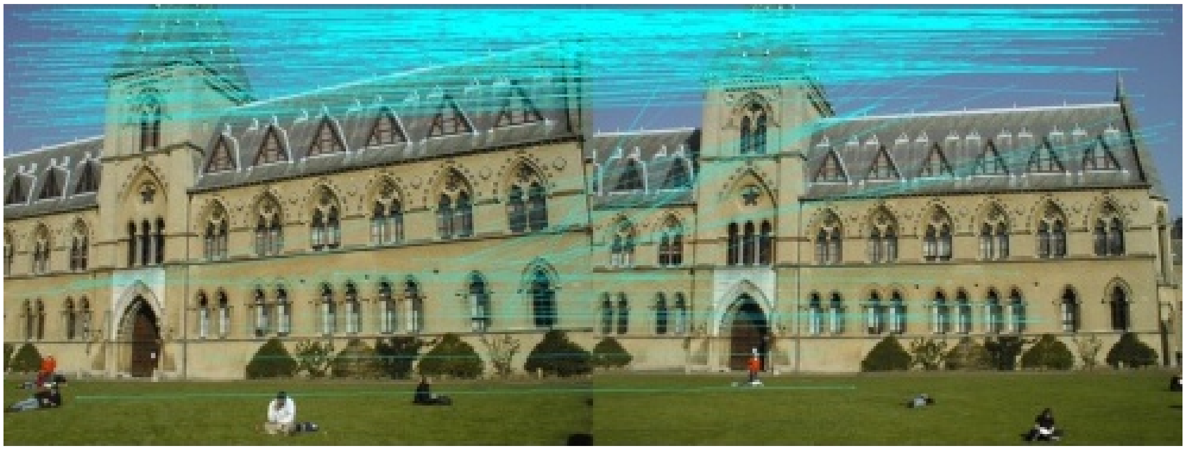

Figure 33: Image pair 3 with $\sigma = 1.6$ based on SSD

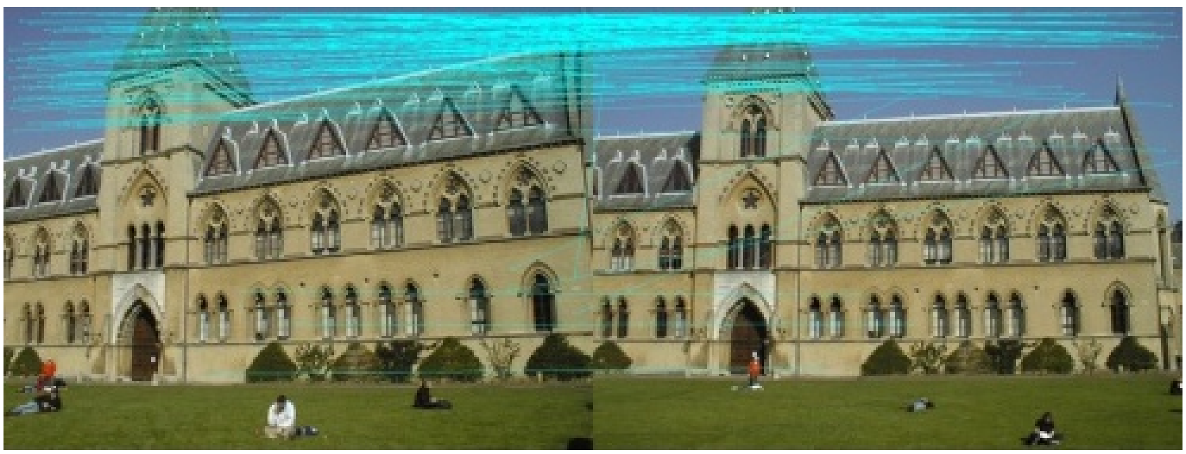

Figure 34: Image pair 3 with $\sigma = 2.2$ based on SSD

5.3.3 Corners Correspondences Based on NCC

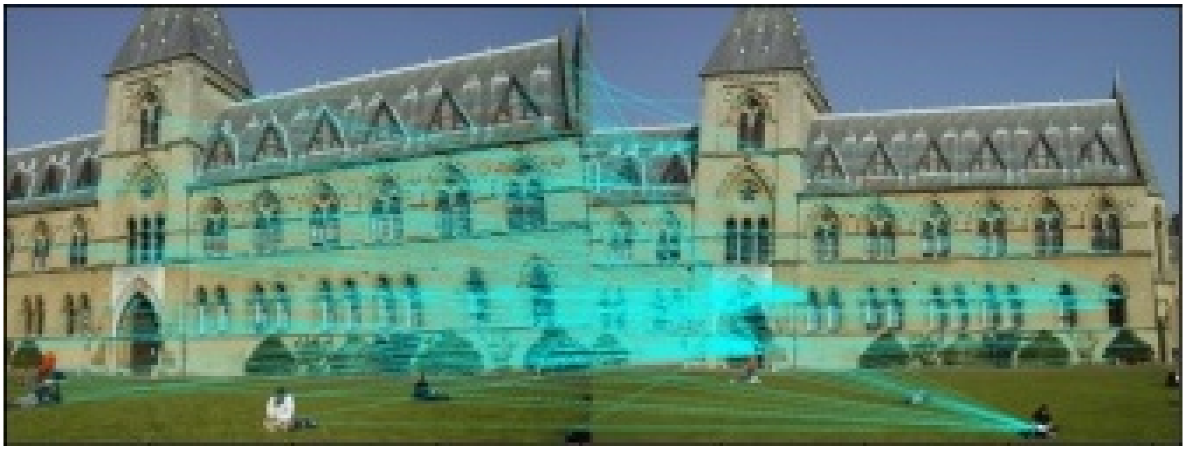

Figure 35: Image pair 3 with $\sigma = 0.8$ based on NCC

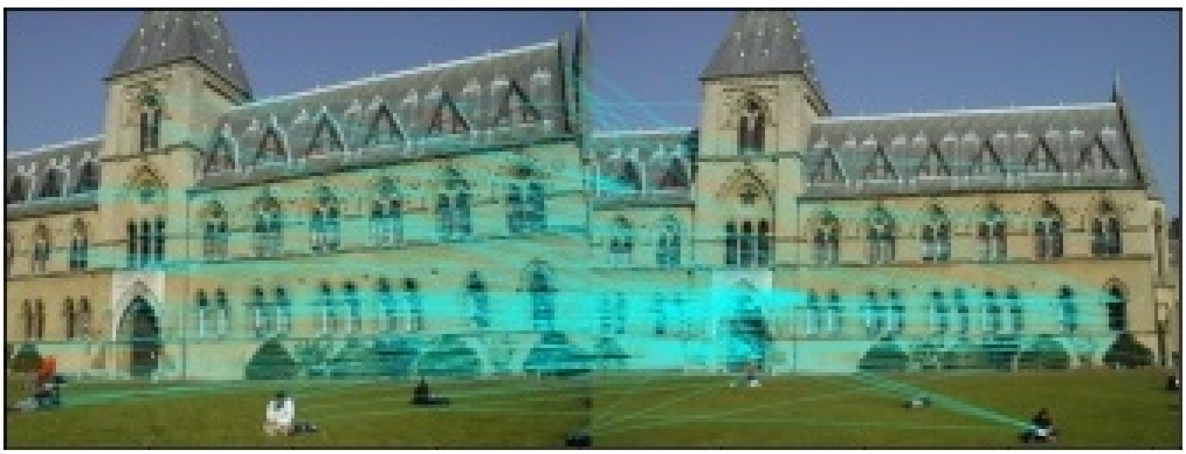

Figure 36: Image pair 3 with $\sigma = 1.2$ based on NCC

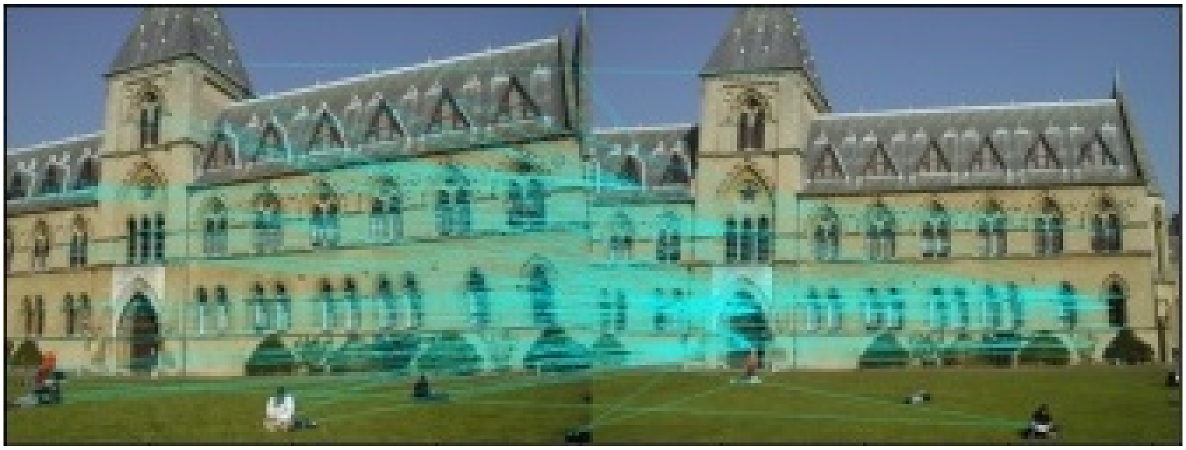

Figure 37: Image pair 3 with $\sigma = 1.6$ based on NCC

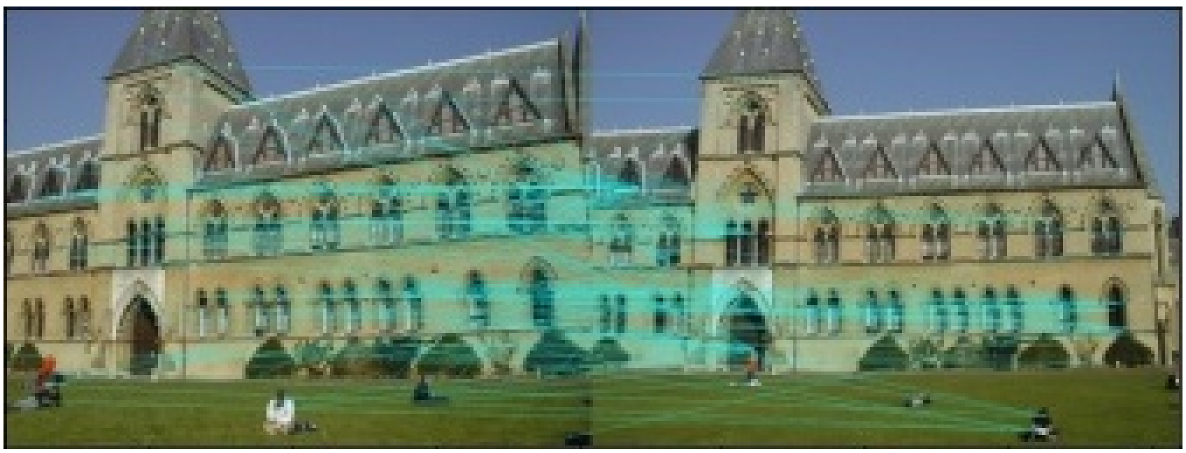

Figure 38: Image pair 3 with $\sigma = 2.2$ based on NCC

5.3.4 Corners correspondences using SIFT

Figure 39: Image pair 3 using SIFT features

5.4 Fourth pair of images

5.4.1 Extracted Interest Points

Figure 40: Corners detection with $\sigma = 0.8$

Figure 41: Corners detection with $\sigma = 1.2$

Figure 42: Corners detection with $\sigma = 1.6$

Figure 43: Corners detection with $\sigma = 2.2$

5.4.2 Corners Correspondences Based on SSD

Figure 44: Image pair 4 with $\sigma = 0.8$ based on SSD

Figure 45: Image pair 4 with $\sigma = 1.2$ based on SSD

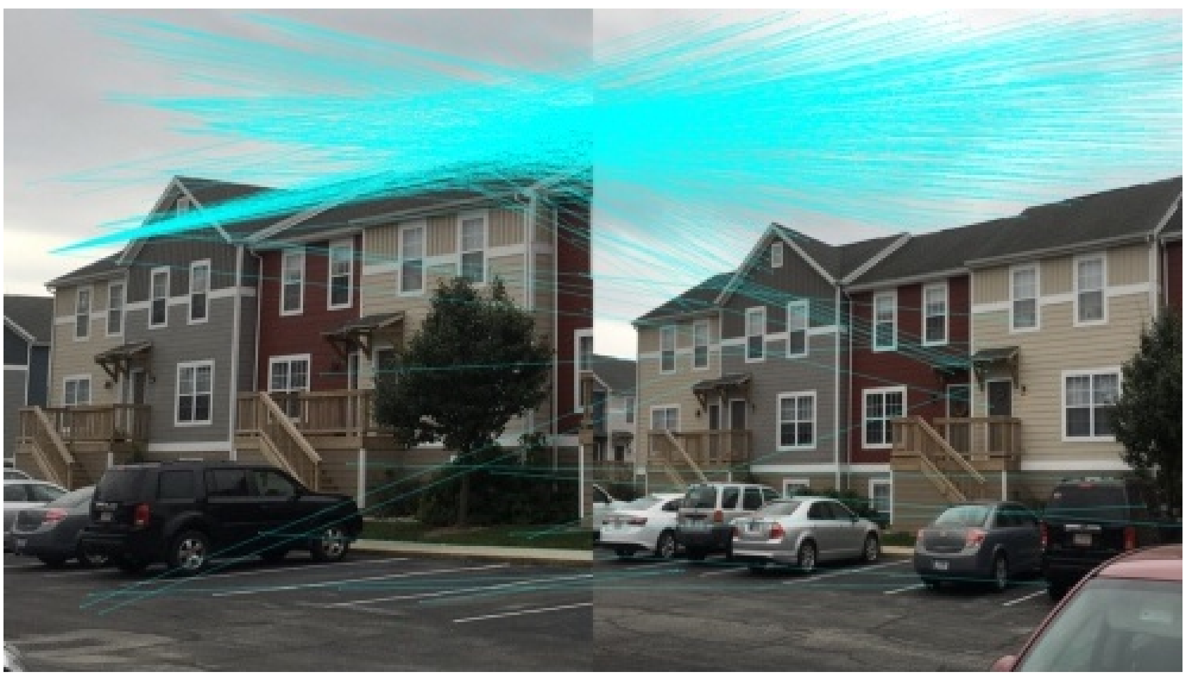

Figure 46: Image pair 4 with $\sigma = 1.6$ based on SSD

Figure 47: Image pair 4 with $\sigma = 2.2$ based on SSD

5.4.3 Corners Correspondences Based on NCC

Figure 48: Image pair 4 with $\sigma = 0.8$ based on NCC

Figure 49: Image pair 4 with $\sigma = 1.2$ based on NCC

Figure 50: Image pair 4 with $\sigma = 1.6$ based on NCC

Figure 51: Image pair 4 with $\sigma = 2.2$ based on NCC

5.4.4 Corners correspondences using SIFT

Figure 52: Image pair 4 using SIFT features

5.5 Fifth pair of images

5.5.1 Extracted Interest Points

Figure 53: Corners detection with $\sigma = 0.8$

Figure 54: Corners detection with $\sigma = 1.2$

Figure 55: Corners detection with $\sigma = 1.6$

Figure 56: Corners detection with $\sigma = 2.2$

5.5.2 Corners Correspondences Based on SSD

Figure 57: Image pair 5 with $\sigma = 0.8$ based on SSD

Figure 58: Image pair 5 with $\sigma = 1.2$ based on SSD

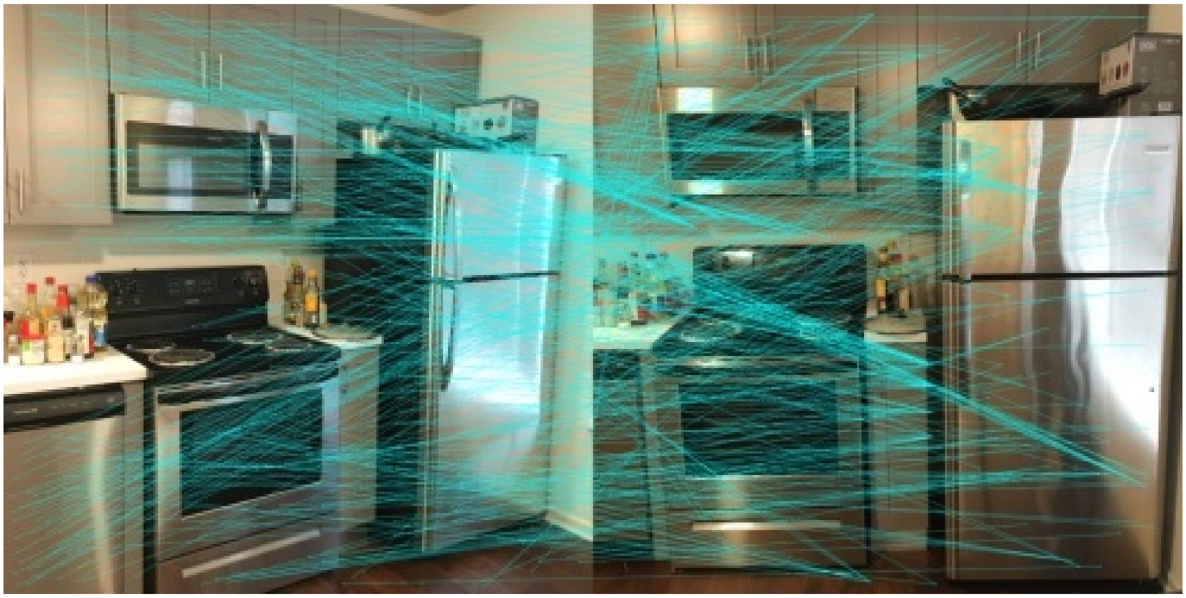

Figure 59: Image pair 5 with $\sigma = 1.6$ based on SSD

Figure 60: Image pair 5 with $\sigma = 2.2$ based on SSD

5.5.3 Corners Correspondences Based on NCC

Figure 61: Image pair 5 with $\sigma = 0.8$ based on NCC

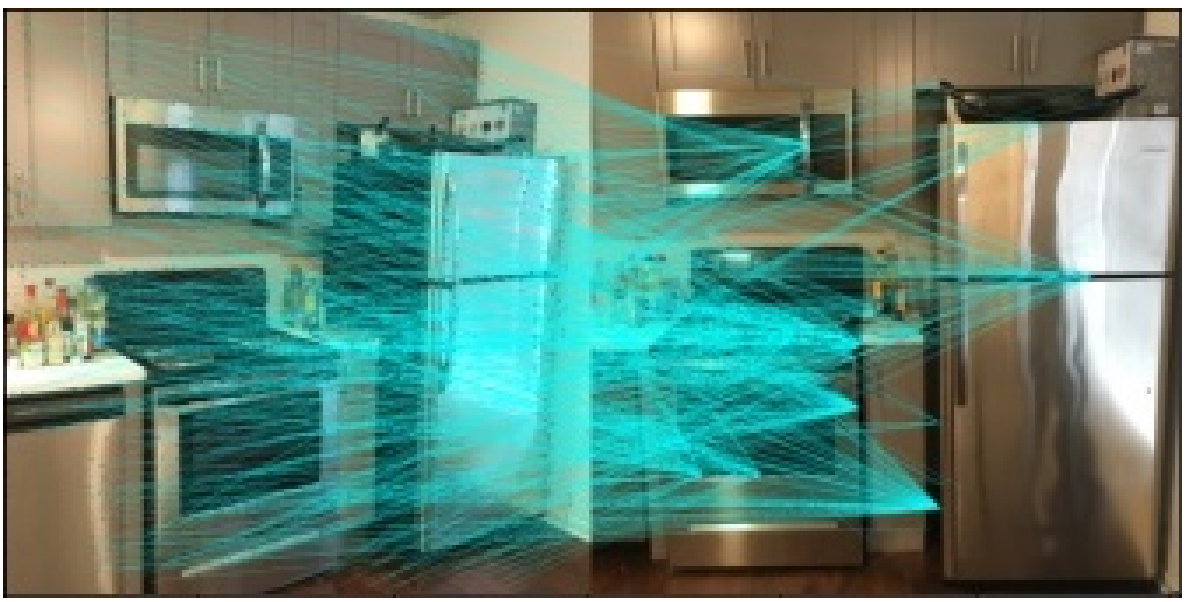

Figure 62: Image pair 5 with $\sigma = 1.2$ based on NCC

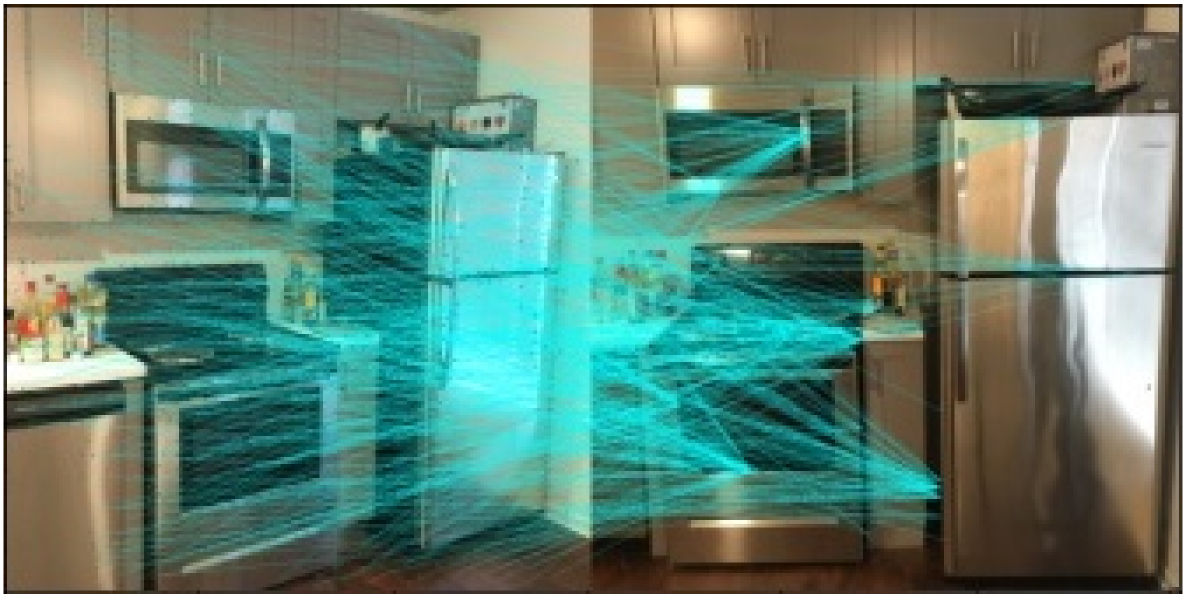

Figure 63: Image pair 5 with $\sigma = 1.6$ based on NCC

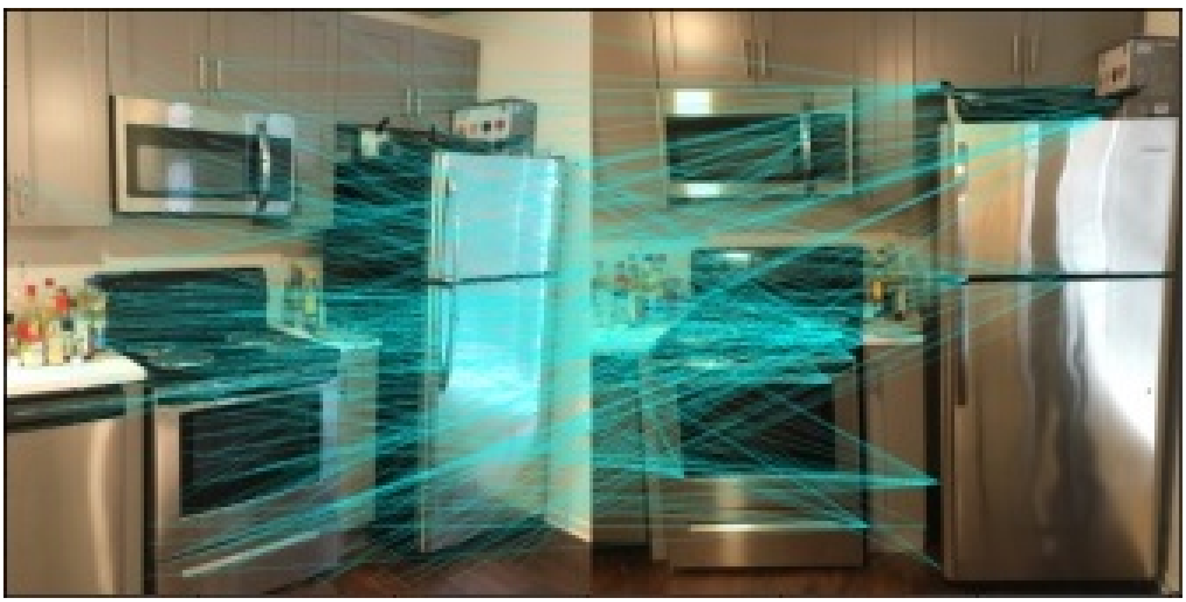

Figure 64: Image pair 5 with $\sigma = 2.2$ based on NCC

5.5.4 Corners correspondences using SIFT

Figure 65: Image pair 5 using SIFT features

6 Observations and Discussions

- The performance of Harris corner detection and corner correspondences depend on the scale value σ. As σ increases, the amount of extracted points decreases and the extracted points represent increasingly coarse.
- The Harris Corner detection based on NCC metric is more robust than the detection based on SSD metric. It's clear to see from the image pairs 1, 3 and 4 that the corner correspondences concentrate on the sky area while the image details on boat and buildings are left to be taken.
- The SIFT features detection outperforms the Harries corner detection for the corner pairs. And it creates more accurate corner correspondences. This is expected since the SIFT is invariant to scales and rotations, and it's robust to changes of illuminations as well as it takes advantage of dominant local orientation for measuring the gradient directions.

7 Source Code

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
from skimage import io
from scipy import signal


def Haar(sigma):
    # Haar filter
    kernel_size = int(np.ceil(4 * sigma))
    if kernel_size % 2 > 0:
        kernel_size += 1
    dx = np.ones((kernel_size, kernel_size))
    dx[:, :int(kernel_size/2)] = -1
    dy = np.ones((kernel_size, kernel_size))
    dy[int(kernel_size/2):, :] = -1

    return dx, dy


def Harris_corners(img, sigma, k, **kwargs):
    # Initialization
    win_size = kwargs.pop("win_size", 15)
    height, width = img.shape
    dx_filter, dy_filter = Haar(sigma=sigma)
    dx = signal.convolve2d(img, dx_filter, mode='same')
    dy = signal.convolve2d(img, dy_filter, mode='same')
    # determine neighboring window size
    kernel_size = int(np.ceil(5 * sigma))
    if kernel_size % 2 > 0:
        kernel_size += 1
    kernel = np.ones((kernel_size, kernel_size))
    # compute entries of matrix C
    sum11 = signal.convolve2d(dx ** 2, kernel, mode='same')
    sum22 = signal.convolve2d(dy ** 2, kernel, mode='same')
    sum12 = signal.convolve2d(dx * dy, kernel, mode='same')
    # trace and determinant of matrix C
    tr = sum11 + sum22
    det = sum11 * sum22 - sum12 ** 2
    # Response of harris corner detector
    r = det - k * tr * tr
    mask = np.ones(r.shape)
    # to reject negative corner detector responses
    mask[r < 0] = 0
    corners = []
    n = int(win_size/2)
    # Suppress non-maximum values
    for i in range(n, width-n):
        for j in range(n, height-n):
            if mask[j, i] > 0:
              max_val = np.amax(r[j-n: j+n+1, i-n: i+n+1])
              if r[j, i] == max_val:
                  corners.append([i, j])

    return corners


def ssd_metric(img1, kp1, img2, corners, k):
    # Initialization
    h, w = img1.shape
    n = int(k/2)
    # Neighborhood of kp1
```

```
    img1_padded = np.zeros([h + 2 * n, w + 2 * n])
    img1_padded[n: n+h, n: n+w] = img1
    img2_padded = np.zeros([h + 2 * n, w + 2 * n])
    img2_padded[n: n+h, n: n+w] = img2
    # find the ssd between kp1 and kp2's in corner list of img2 and return the correspondence
    neighbor1 = img1_padded[kp1[1]: kp1[1] + 2 * n, kp1[0]: kp1[0] + 2 * n]
    min_ssd = 1e6
    index = int(len(corner2)-1)
    for idx, kp2 in enumerate(corners):
        neighbor2 = img2_padded[kp2[1]: kp2[1] + 2 * n, kp2[0]: kp2[0] + 2 * n]
        ssd = np.sum((neighbor1 - neighbor2) ** 2)
        if min_ssd > ssd:
            min_ssd = ssd
            index = idx

    return index, min_ssd


def ncc_metric(img1, kp1, img2, corners, k):
    # Initialization
    h, w = img1.shape
    n = int(k/2)
    # Neighborhood of kp1
    img1_padded = np.zeros([h + 2 * n, w + 2 * n])
    img1_padded[n: n+h, n: n+w] = img1
    img2_padded = np.zeros([h + 2 * n, w + 2 * n])
    img2_padded[n: n+h, n: n+w] = img2
    # find the ncc between kp1 and kp2's in corner list of img2 and return the correspondence
    neighbor1 = img1_padded[kp1[1]: kp1[1] + 2 * n, kp1[0]: kp1[0] + 2 * n]
    max_ncc = -1e6
    index = int(len(corner2)-1)
    for idx, kp2 in enumerate(corners):
        neighbor2 = img2_padded[kp2[1]: kp2[1] + 2 * n, kp2[0]: kp2[0] + 2 * n]
        sum1 = np.sum(neighbor1 - np.mean(neighbor1) ** 2)
        sum2 = np.sum(neighbor2 - np.mean(neighbor2) ** 2)
        ncc = np.sum((neighbor1 - np.mean(neighbor1)) * (neighbor2 - np.mean(neighbor2))) / np.sqrt(sum1 * sum2)
        if ncc > max_ncc:
            max_ncc = ncc
            index = idx

    return index, max_ncc


def build_corr(img1, corner1, img2, corner2, metric):
    # Assuming the corner list of img1 is shorter than corner list of img2
    match_corners = corner2
    metric_value = np.zeros((len(corner1), 1))
    # Determine the correspondence for each point in the corner list of img1
    if metric == 'SSD':
        for idx, kp in enumerate(corner1):
            index, min_ssd = ssd_metric(img1, kp, img2, corner2, k=25)
            match_corners[idx] = corner2[int(index)]
            metric_value[idx] = min_ssd
    if metric == 'NCC':
        for idx, kp in enumerate(corner1):
            index, max_ncc = ncc_metric(img1, kp, img2, corner2, k=25)
            match_corners[idx] = corner2[int(index)]
            metric_value[idx] = max_ncc
    # Return the matching pairs between two corner lists
    match_pairs = [(corner1[i], match_corners[i]) for i in range(len(corner1))]
```

```
    return match_corners, match_pairs, metric_value


def filter_corr(match_pairs, metric_value, mode, thres):
    # Find the good correspondences among all the correspondences by comparing the
    metric value
    pairs = []
    if mode == 'SSD':
        for i in range(len(metric_value)):
            if metric_value[i] < thres:
                pairs.append(match_pairs[i])
    if mode == 'NCC':
        for i in range(len(metric_value)):
            if metric_value[i] > thres:
                pairs.append(match_pairs[i])

    return pairs


# read images
img1 = io.imread('hw4_Task1_Images/pair3/1.JPG')
img2 = io.imread('hw4_Task1_Images/pair3/2.JPG')
# Initialization
img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]), interpolation=cv2.INTER_AREA)
w, h, d = img1.shape
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
sigma = 2.2
k = 0.05
win_size = 15          # 15 for pair 1, 3 ; 13 for pair 2; 25 for pair 4; 31 for pair 5
corner1 = Harris_corners(gray1, sigma=sigma, k=k, win_size=win_size)
corner2 = Harris_corners(gray2, sigma=sigma, k=k, win_size=win_size)
# Plot the detected corners
img = np.hstack((img1, img2))
for i in range(len(corner1)):
    pt1 = corner1[i]
    cv2.circle(img, tuple(pt1), 4, (255, 0, 0), 2)
    cv2.circle(img, tuple(pt1), 4, (255, 0, 0), 2)
for j in range(len(corner2)):
    pt2 = [corner2[j][0] + h, corner2[j][1]]
    cv2.circle(img, tuple(pt2), 4, (255, 0, 0), 2)
    cv2.circle(img, tuple(pt2), 4, (255, 0, 0), 2)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.savefig('Corners_sigma_{}.jpeg'.format(sigma))
plt.show()
plt.clf()

# Create corner correspondence
method = 'SSD'          # 'SSD' or 'NCC'
threshold = 10000       # 10000 for pair 1, 3, 5; 20000 for pair 2; 12000 for pair 4;
name = 'SSD_sigma_{}.jpeg'.format(sigma)
# # Hyper-parameters for using NCC metric
# method = 'NCC'
# threshold = 0.042     # 0.042 for pair 1; 0.58 for pair 2; 0.22 for pair 3, 4; 0.041
     for pair 5;
# name = 'NCC_sigma_{}.jpeg'.format(sigma)
if len(corner1) > len(corner2):
    match_corners, match_pairs, metric_value = build_corr(gray2, corner2, gray1,
    corner1, metric=method)
    good_pairs = filter_corr(match_pairs, metric_value, mode=method, thres=threshold)
    img = np.hstack((img1, img2))
    for idx, i in enumerate(good_pairs):
        pt1 = i[1]
```

```
        pt2 = [i[0][0] + h, i[0][1]]
        cv2.line(img, tuple(pt1), tuple(pt2), (0, 255, 255), 1)
        cv2.circle(img, tuple(pt1), 2, (0, 255, 255), 1)
        cv2.circle(img, tuple(pt2), 2, (0, 255, 255), 1)
else:
    match_corners, match_pairs, metric_value = build_corr(gray1, corner1, gray2,
    corner2, metric=method)
    good_pairs = filter_corr(match_pairs, metric_value, mode=method, thres=threshold)
    img = np.hstack((img1, img2))
    for idx, i in enumerate(good_pairs):
        pt1 = i[0]
        pt2 = [i[1][0] + h, i[1][1]]
        cv2.line(img, tuple(pt1), tuple(pt2), (0, 255, 255), 1)
        cv2.circle(img, tuple(pt1), 2, (0, 255, 255), 1)
        cv2.circle(img, tuple(pt2), 2, (0, 255, 255), 1)
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.savefig(name)
plt.show()
plt.clf()

# SIFT
sift = cv2.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)
good_matches = []
for a, b in matches:
    if a.distance < 0.6 * b.distance:
        good_matches.append([a])
img = np.zeros((1, 1))
img = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, img, flags=2)
plt.imshow(img, cmap='gray')
plt.savefig('SIFT_img.jpeg')
plt.show()
plt.clf()
```

hw4.py