ECE 661: Homework 4

Xingguang Zhang

(Fall 2020)

Task 1

1. Harris Corner Detector

By a corner we mean any pixel in the vicinity of which the gray levels show significant variations in at least two directions. To detect the variations along any direction on an image, we need the following steps:

1. Use Haar wavelet to compute the first-order derivative along x and y directions. The Haar wavelet filter is defined by σ, for a given σ, the filter is scaled up to an $M \times M$ operator where M is the smallest even integer greater than 4σ. For example, if $\sigma = 1.2$, the Haar filters along x and y direction are:

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

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

2. After we get the first order derivative maps D_x and D_y by filtering the original gray scale image by F_x and F_y, we then construct a matrix C for each pixel location to extract the local features from its associated neighborhood area. The neighborhood area is of size $5\sigma \times 5\sigma$, the matrix $C^{i,j}$ on each pixel location (i, j) is defined as:

$$C^{i,j} = \begin{bmatrix} \sum D_x^2 & \sum D_x D_y \\ \sum D_y D_x & \sum D_y^2 \end{bmatrix}$$

Where sum operation is implemented on the neighborhood area centered on (i, j). This can be achieved simply by filtering the derivative maps D_x^2, $D_x D_y$ and D_y^2 by a $5\sigma \times 5\sigma$ all-ones filter.

Extract interest points using Harris Corner detector

3. Once we get C, we can find the direction of derivative by eigen-decomposing it. If C has full rank, which is 2, the orthonormal eigen vectors of C indicate the 2 primary directions of the derivative and the eigenvalues λ_1 and λ_2 indicate the magnitude of the derivative along the associated directions. For a corner, the derivative along two primary directions should have a reasonable ratio. Since we assume the corner is invariant to rotation, we only need to care about the eigenvalues. Assume $\lambda_1 \geq \lambda_2$, the ratio $r = \lambda_2/\lambda_1$ has to be larger than a threshold k.

4. Since C is only a 2×2 matrix, we can compute r very efficiently without an explicit eigen-decomposition of C by realizing:

$$Trace(C) = \sum D_x^2 + \sum D_y^2 = \lambda_1 + \lambda_2$$

$$Det(C) = \sum D_x^2 \sum D_y^2 - (\sum D_x D_y)^2 = \lambda_1 \lambda_2$$

This implies

$$\frac{Det(C)}{[Tr(C)]^2} = \frac{\lambda_1 \lambda_2}{(\lambda_1 + \lambda_2)^2} = \frac{r}{(1+r)^2}$$

Setting the threshold on the $\frac{Det(C)}{[Tr(C)]^2}$, we can instead select all R where

$$R = Det(C) - k * [Tr(C)]^2 \geq 0$$

R is the Harris corner detector Response image. Only those R greater than 0 is selected as corner candidates, and k is usually set as $0.04 - 0.06$. To filter out derivative noises, we set another threshold r_t on R to control the number of candidate corners. We further adaptively select those $R \geq r_t * R_{max}$

5. Non-maximum suppression. Usually many pixels around a corner have large response values, to avoid redundant, for each corner candidate, we need to evaluate if the corner candidate has the largest response value inside its neighborhood, if it has, then we finally recognize it is a corner.

Use NCC and SSD metrics to establish correspondences between the two sets of interest points of the image pairs

After we detected two sets of corners, we need to find the correspondence between them. The correspondences are established by comparing their associated features, which are the neighborhood pixels of the interest points. For the interest point P_i located at and its neighbor set $\mathcal{N}(P_i)$ on the first image, the feature vector $f^i(x, y)$ is defined as the flattened gray scales of the pixels in $\mathcal{N}(P_i)$, then the sum of squared difference(SSD) and normalized cross-correlation(NCC) between the i-th interest point on image 1 and the j-th interest point on image 2 can be evaluated by:

$$SSD_{i,j} = \sum_x \sum_y \left| f_1^i(x, y) - f_2^j(x, y) \right|^2$$

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

Where m is the average value of the associate feature f.

Both SSD and NCC can be vectorized by taking f as an 1D vector instead of a 2D block. This makes the pipeline to compute SSD and NCC can also be directly implemented on SIFT and SURF features.

For the i-th interest point on image 1, the most similar point on image 2 is the n-th interest point, where $n = \underset{j}{argmin}(SSD_{i,j})$ or $n = \underset{j}{argmax}(NCC_{i,j})$. If the i-th interest point on image 1 is also the most similar point of the n-th interest point on image 2, which means $i = \underset{m}{argmin}(SSD_{m,n})$ or $i = \underset{m}{argmax}(NCC_{m,n})$, we claim that these two points are a good match, and we find a correspondence.

Apply the Harris corner detector for at least 4 different scales

My experiment tested the Harris corner detector for 5 scales: $\sigma = 0.5, 0.707, 1, 1.414, 2$, the results are shown below, where the interest points are marked by red circles, among them the matched interest points are marked by green circles, and the corresponding points are connected by green lines.

Figure 1: The detected interest points on pair 1 images when $\sigma = 0.5$

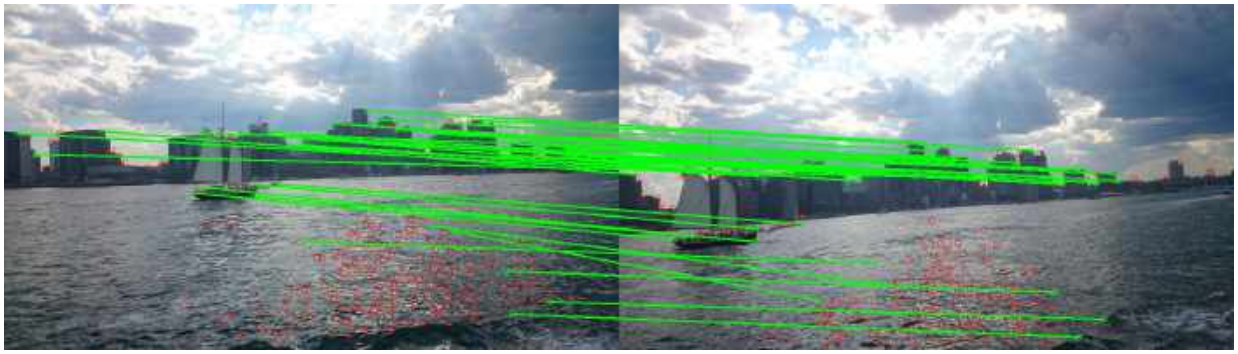

Figure 2: The matched correspondences using SSD on pair 1 images when $\sigma = 0.5$

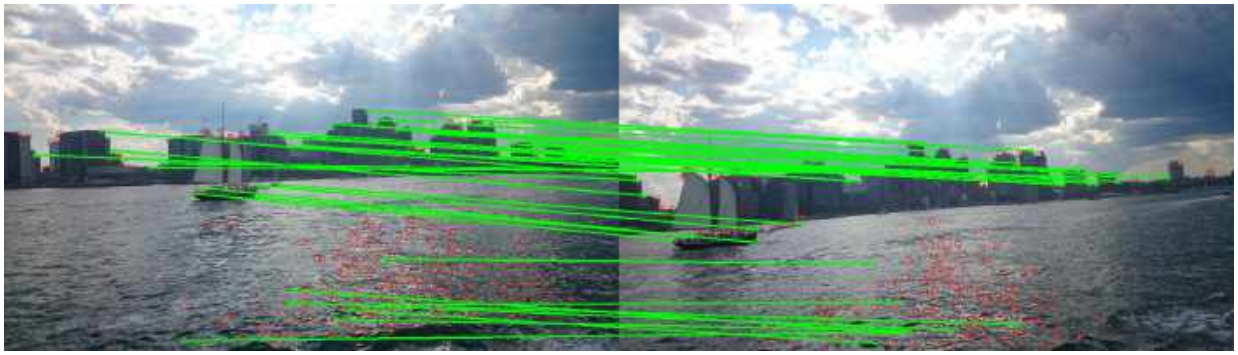

Figure 3: The matched correspondences using NCC on pair 1 images when $\sigma = 0.5$

Figure 4: The detected interest points on pair 1 images when $\sigma = 0.707$

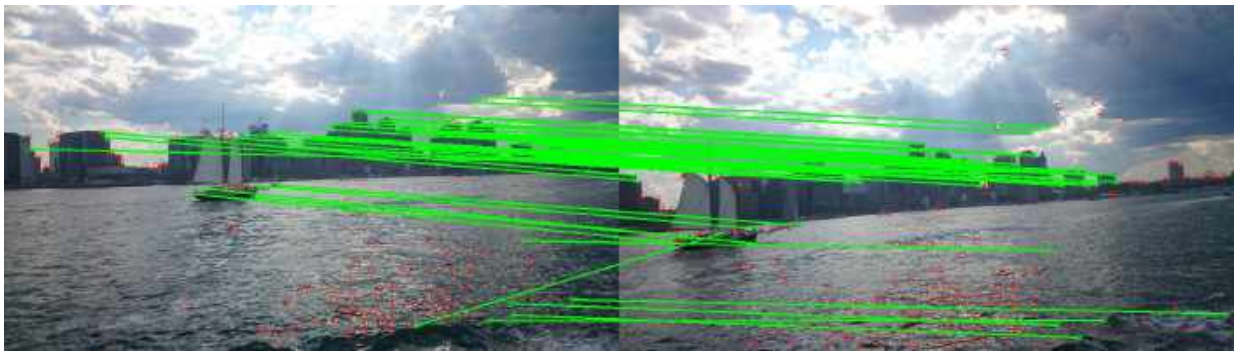

Figure 5: The matched correspondences using SSD on pair 1 images when $\sigma = 0.707$

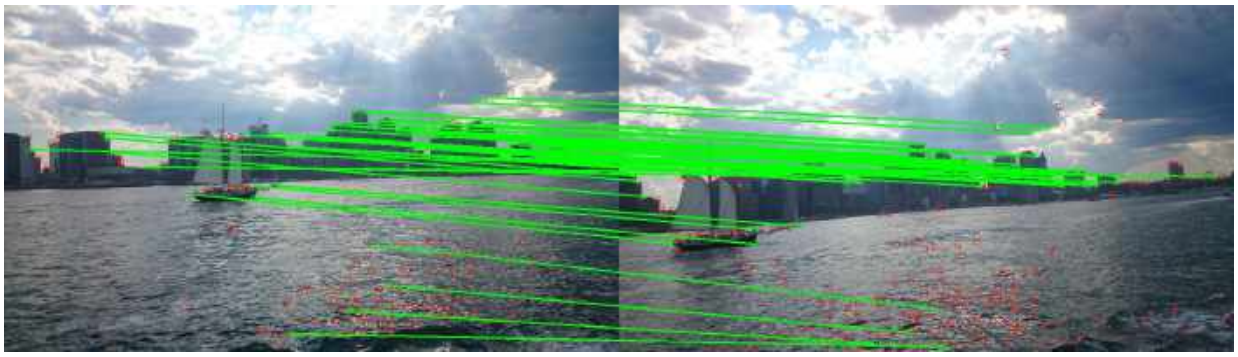

Figure 6: The matched correspondences using NCC on pair 1 images when $\sigma = 0.707$

Figure 7: The detected interest points on pair 1 images when $\sigma = 1$

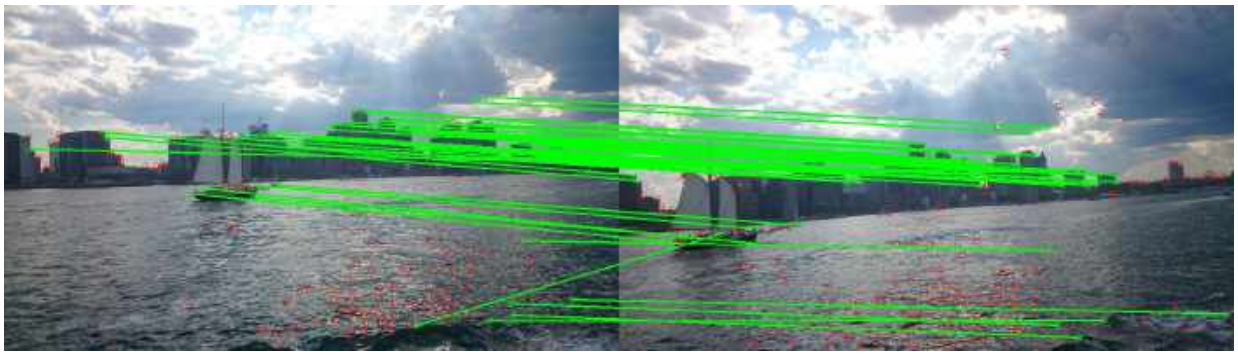

Figure 8: The matched correspondences using SSD on pair 1 images when $\sigma = 1$

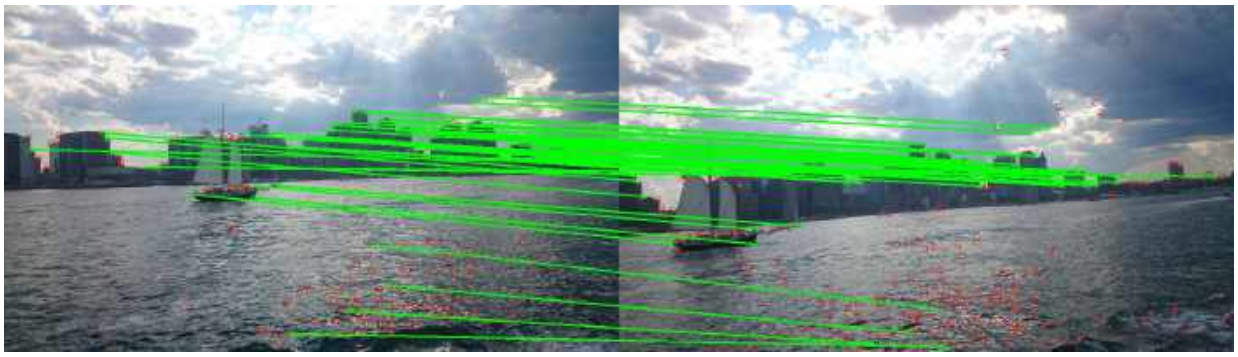

Figure 9: The matched correspondences using NCC on pair 1 images when $\sigma = 1$

Figure 10: The detected interest points on pair 1 images when $\sigma = 1.414$

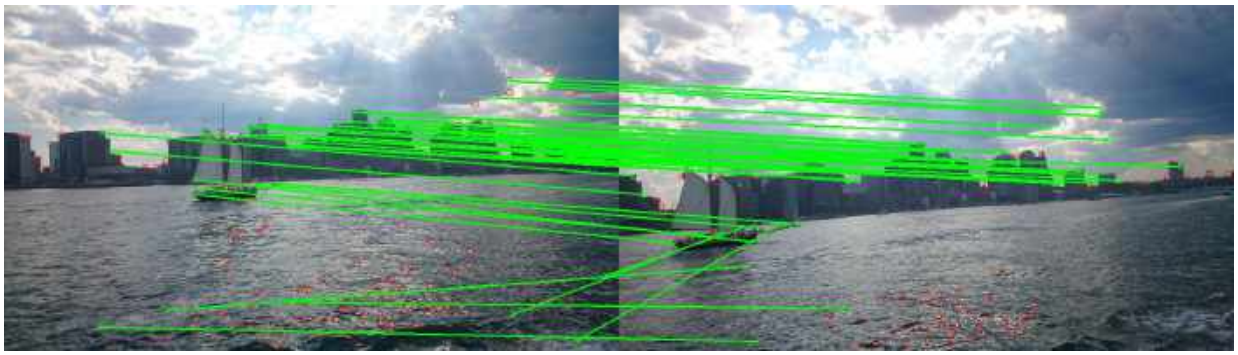

Figure 11: The matched correspondences using SSD on pair 1 images when $\sigma = 1.414$

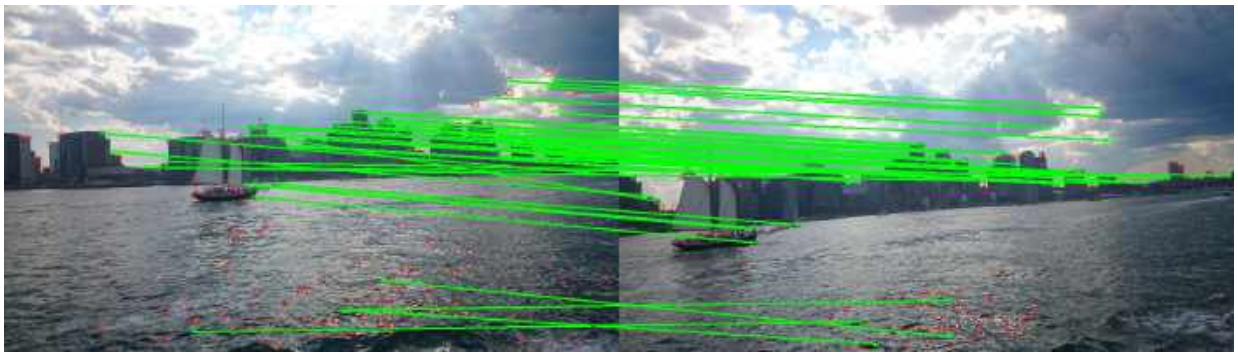

Figure 12: The matched correspondences using NCC on pair 1 images when $\sigma = 1.414$

Figure 13: The detected interest points on pair 1 images when $\sigma = 2$

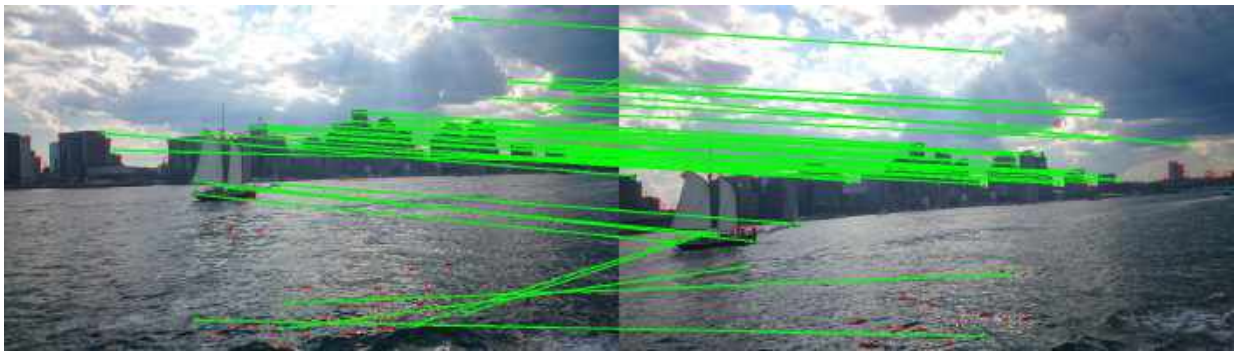

Figure 14: The matched correspondences using SSD on pair 2 images when $\sigma = 2$

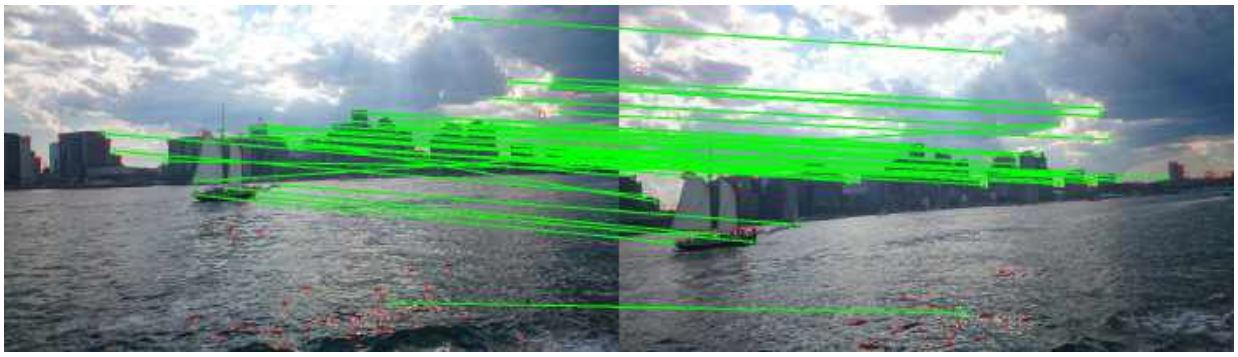

Figure 15: The matched correspondences using NCC on pair 1 images when $\sigma = 2$

Figure 16: The detected interest points on pair 2 images when $\sigma = 0.5$

Figure 17: The matched correspondences using SSD on pair 2 images when $\sigma = 0.5$

Figure 18: The matched correspondences using NCC on pair 2 images when $\sigma = 0.5$

Figure 19: The detected interest points on pair 2 images when $\sigma = 0.707$

Figure 20: The matched correspondences using SSD on pair 2 images when $\sigma = 0.707$

Figure 21: The matched correspondences using NCC on pair 2 images when $\sigma = 0.707$

Figure 22: The detected interest points on pair 2 images when $\sigma = 1$

Figure 23: The matched correspondences using SSD on pair 2 images when $\sigma = 1$

Figure 24: The matched correspondences using NCC on pair 2 images when $\sigma = 1$

Figure 25: The detected interest points on pair 2 images when $\sigma = 1.414$

Figure 26: The matched correspondences using SSD on pair 2 images when $\sigma = 1.414$

Figure 27: The matched correspondences using NCC on pair 2 images when $\sigma = 1.414$

Figure 28: The detected interest points on pair 2 images when $\sigma = 2$

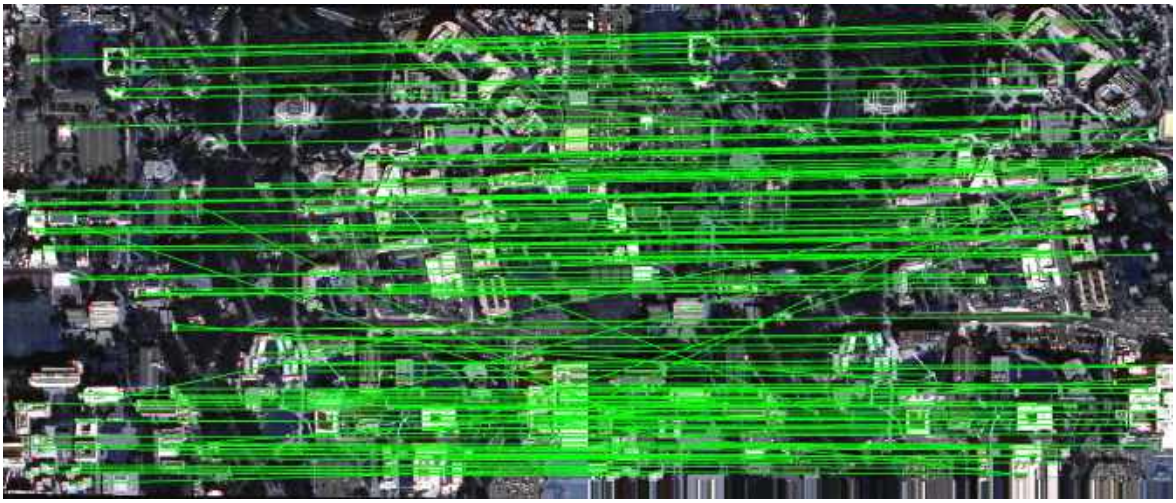

Figure 29: The matched correspondences using SSD on pair 2 images when $\sigma = 2$

Figure 30: The matched correspondences using NCC on pair 2 images when $\sigma = 2$

Figure 31: The detected interest points on pair 3 images when $\sigma = 0.5$

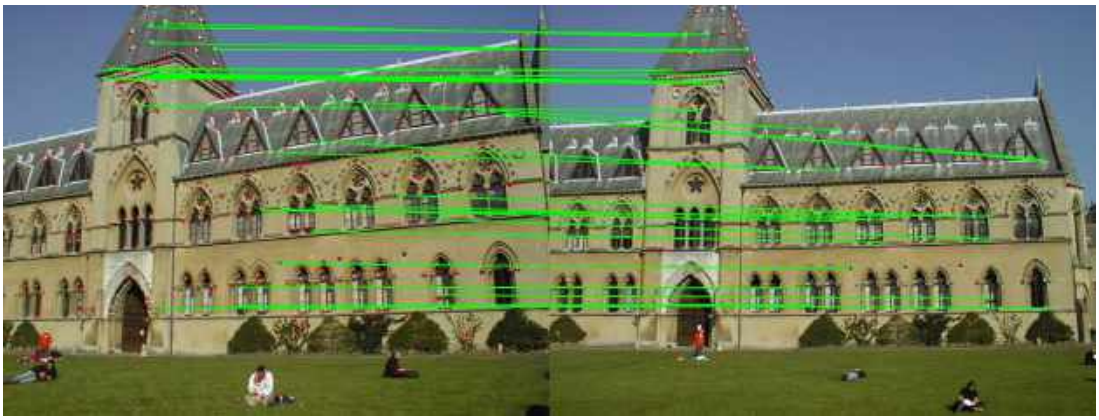

Figure 32: The matched correspondences using SSD on pair 3 images when $\sigma = 0.5$

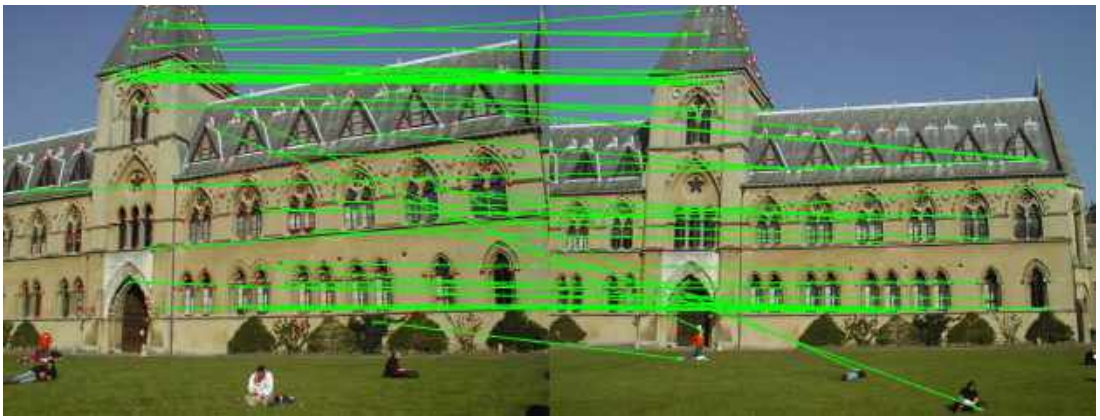

Figure 33: The matched correspondences using NCC on pair 3 images when $\sigma = 0.5$

Figure 34: The detected interest points on pair 3 images when $\sigma = 0.707$

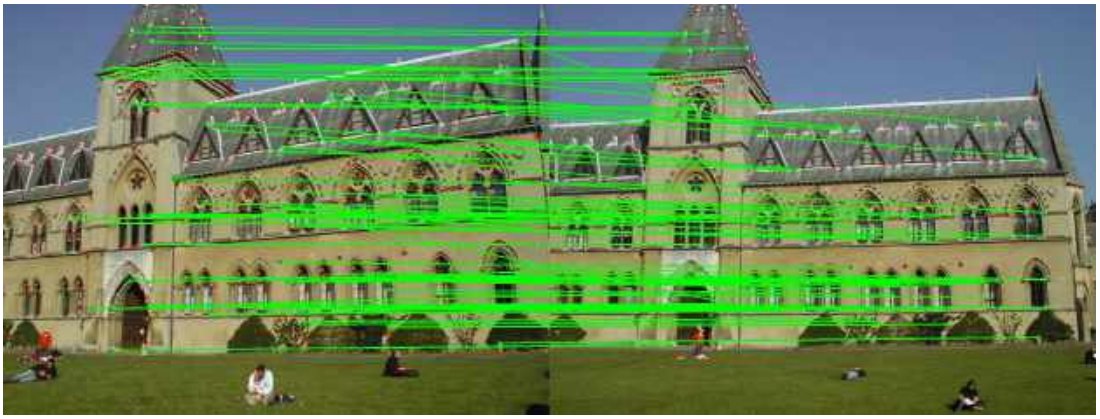

Figure 35: The matched correspondences using SSD on pair 3 images when $\sigma = 0.707$

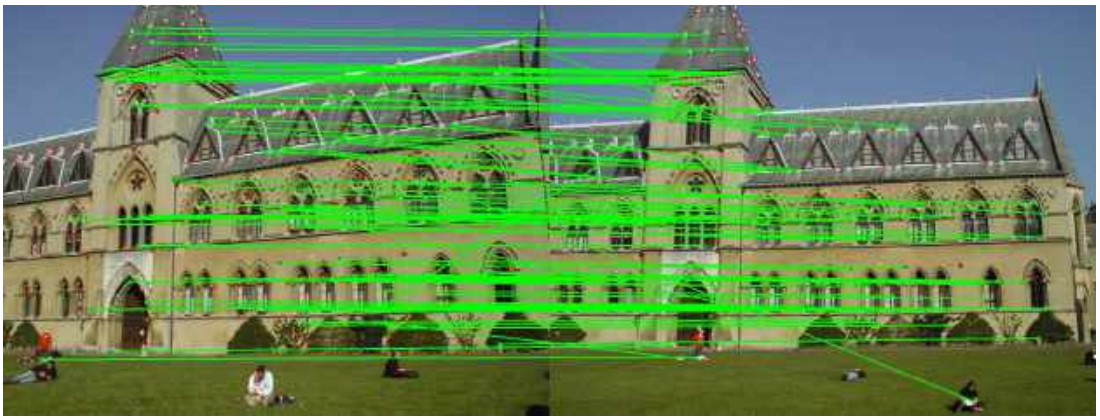

Figure 36: The matched correspondences using NCC on pair 3 images when $\sigma = 0.707$

Figure 37: The detected interest points on pair 3 images when $\sigma = 1$

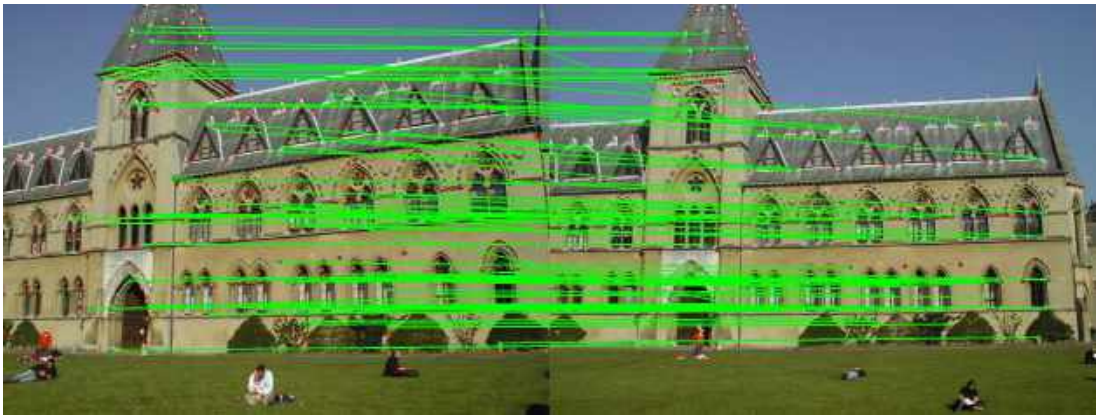

Figure 38: The matched correspondences using SSD on pair 3 images when $\sigma = 1$

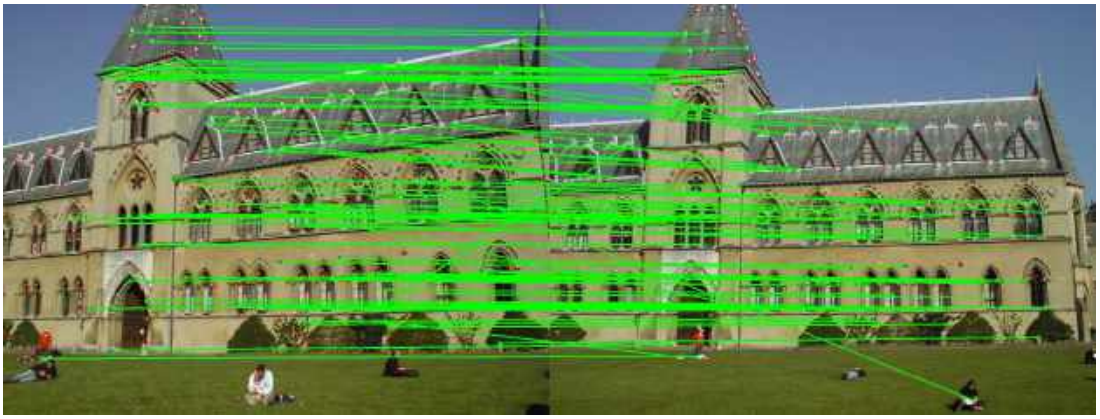

Figure 39: The matched correspondences using NCC on pair 3 images when $\sigma = 1$

Figure 40: The detected interest points on pair 3 images when $\sigma = 1.414$

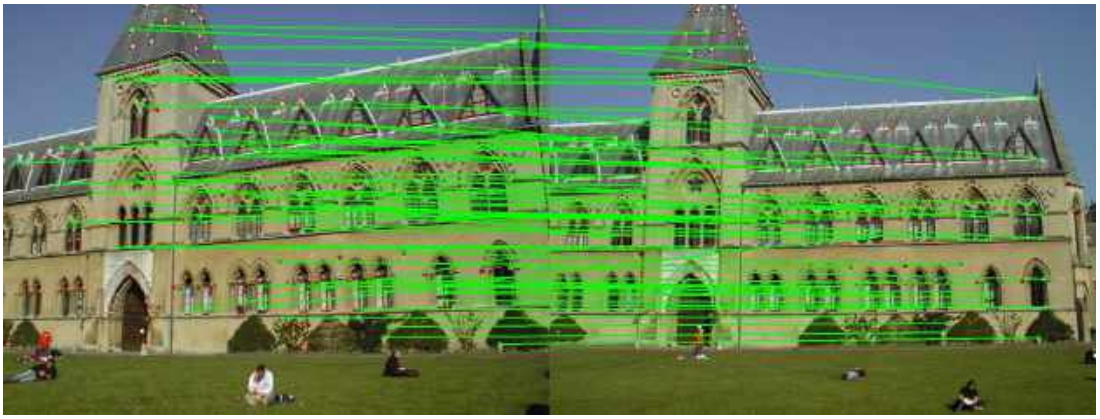

Figure 41: The matched correspondences using SSD on pair 3 images when $\sigma = 1.414$

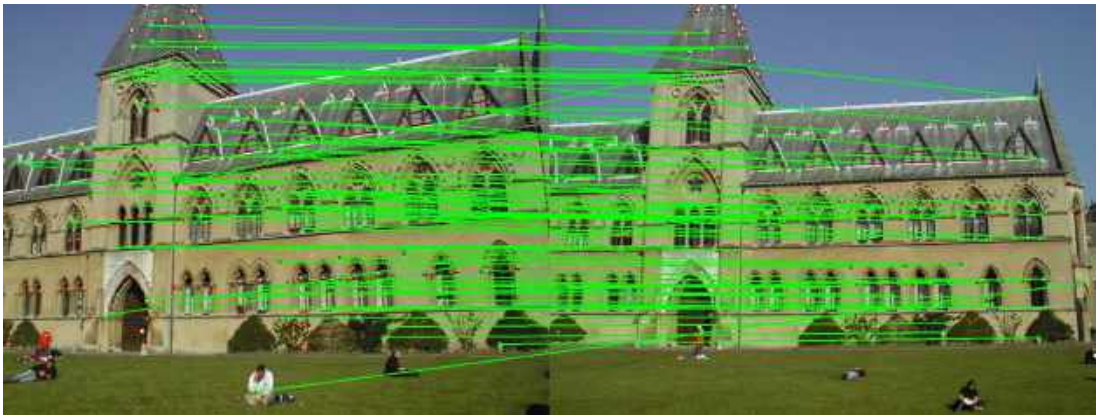

Figure 42: The matched correspondences using NCC on pair 3 images when $\sigma = 1.414$

Figure 43: The detected interest points on pair 3 images when $\sigma = 2$

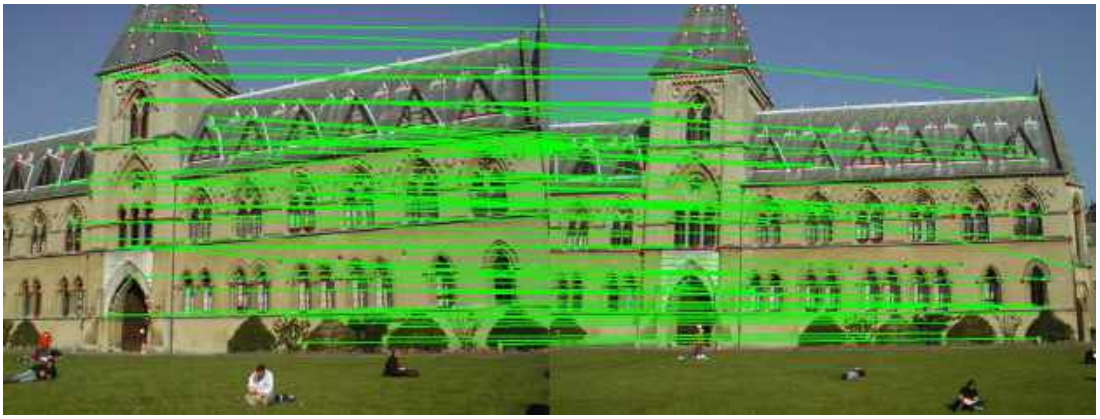

Figure 44: The matched correspondences using SSD on pair 3 images when $\sigma = 2$

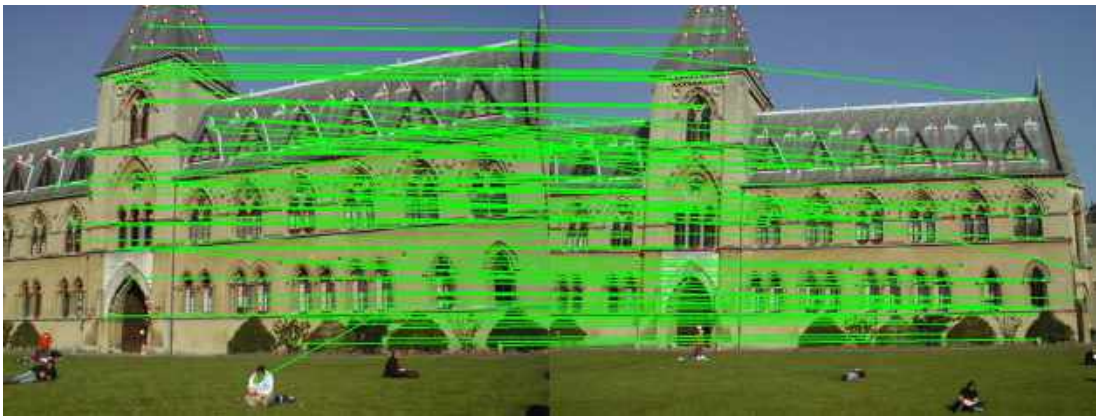

Figure 45: The matched correspondences using NCC on pair 3 images when $\sigma = 2$

Parameters and Observation

- The parameter k for thresholding the $det(C)$ and $tr(C)^2$ is set to 0.05 for all experiments.
- The parameter r_t for controlling the number of interest points is set to 0.02 for pair 1, 0.15 for pair 2, 0.05 for pair 3.
- The window size for non-maximum suppression is set to 15×15, the window size for construct neighborhood to find interest point correspondences is set to 41×41 for all experiments.
- Usually when σ increases from a small value, for example 0.5, the extracted interest points will be less, but we may get more matched pairs. This is because when we compute the derivative in a small window, the noise may affect the result severely, this will give us lots of useless interest points. Larger σ will bring us robustness, but the accuracy to locate the corners will be reduced. Robustness and sensitivity are always trade off.
- On pair 1, most mismatches are caused by SSD, but on pair 2 and 3, most mismatches are caused by NCC method. SSD deals with the similarity on absolute gray scale value, while NCC focuses more on the similarity of the variation inside the interest area. When the lightening on the object surface changes rapidly, SSD may suffer from lots of mismatches, just like the water surface on image pair 1 shows. But NCC sometimes will match two totally different features together, just like shown by image pair 3, where there are always some lines from people to building, they may have similar local variation, but that's absolutely a mistake. Actually we can use SSD and NCC together and find the correspondences when both are satisfied. When we have enough interest points, the combination can bring extra robustness.

2. SURF or SIFT

Scale-invariant feature transform (SIFT) and Speeded up robust features (SURF) are both interest point detectors supported by OpenCV library, so they can be easily implemented in our codes, we will introduce the pipeline of these two algorithms and show the results below.

SIFT

∗ We first construct a DoG (different of Gaussian) pyramid for taking advantage of the scale space. Find the local extremum in the DoG pyramid w.r.t x, y and σ(scale variable), which indicate

the potential locations where gray levels change rapidly in several directions under different scales, so it has the potential to be find the scale-invariant local extremum.

* As σ increases, the individual discrete points of DoG represent become coarser. To locate the original local extrama on the original image, we need to find the "subpixel" local extremum in high octave of the pyramid given the higher level local extremum $\overrightarrow{x}_0 = (x_0, y_0, \sigma_0)$, this requires to use the Gaussian matrix $H(\overrightarrow{x}_0)$ and Jacobian vector $J(x_0)$, the new subpixel extremum will be at $\overrightarrow{x} = \overrightarrow{x}_0 - H^{-1}(\overrightarrow{x}_0)J(x_0)$.

* Weed out the weak extremum by thresholding the DoG value at the extremum.

* Associate 'dominate local orientation' with each extremum extracted from above.

* enerate a 128-element descriptor for the extracted local extremum

* Using the SIFT detector, we extracted the interest points from each image, and then match the interest points just like what we did in Harris detector. The feature vector here is a 128-dimension descriptor, and I used SSD for computing the similarity between the descriptors and matching the similar interst points, the result is shown below, where I extracted 300 feature points with the highest confident scores as my interest points for each individual image.

Figure 46: The detected interest points on pair 1 images using SIFT

Figure 47: The matched correspondences on pair 1 images with SIFT feature

Figure 48: The detected interest points on pair 1 images using SIFT

Figure 49: The matched correspondences on pair 2 images with SIFT feature

Figure 50: The detected interest points on pair 3 images using SIFT

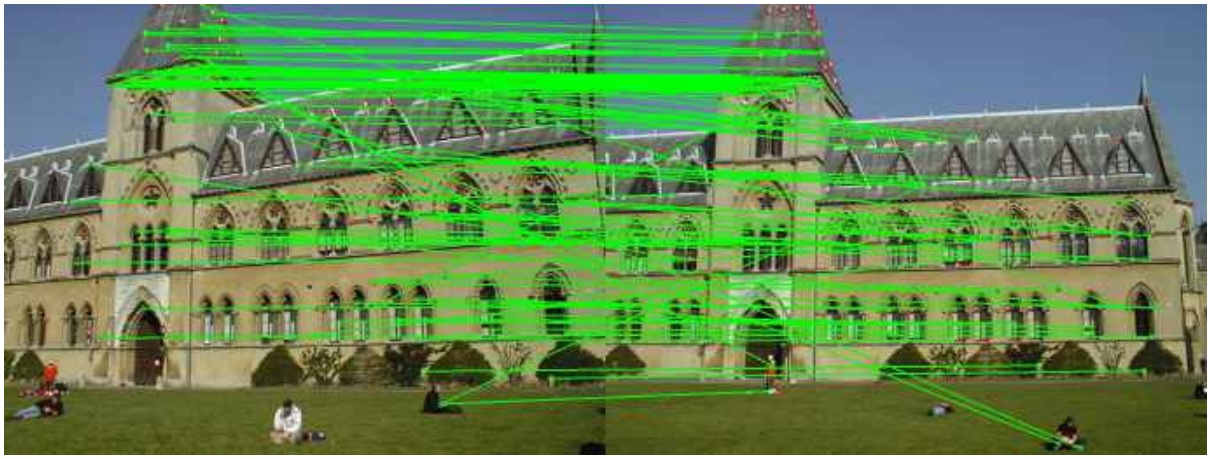

Figure 51: The matched correspondences on pair 3 images with SIFT feature

SURF

Whereas SIFT places the interest points at the extremum of the values of LoG/DoG, SURF places the interest points at locations where the determinant of the Hessian is maximized. Different from Harris where the first order is used to construct C, SURF uses the second order derivative for detecting interest point, it also uses the first order derivative for orientation computing. The use of SURF detector under OpenCV is pretty the same as SIFT, the results are shown below:

Figure 52: The detected interest points on pair 1 images using SURF when Hessian threshold is set to 2000

Figure 53: The matched correspondences on pair 4 images with SURF features when Hessian threshold is set to 2000

Figure 54: The detected interest points on pair 2 images using SURF when Hessian threshold is set to 20000

Figure 55: The matched correspondences on pair 2 images with SURF features when Hessian threshold is set to 20000

Figure 56: The detected interest points on pair 3 images using SURF when Hessian threshold is set to 7500

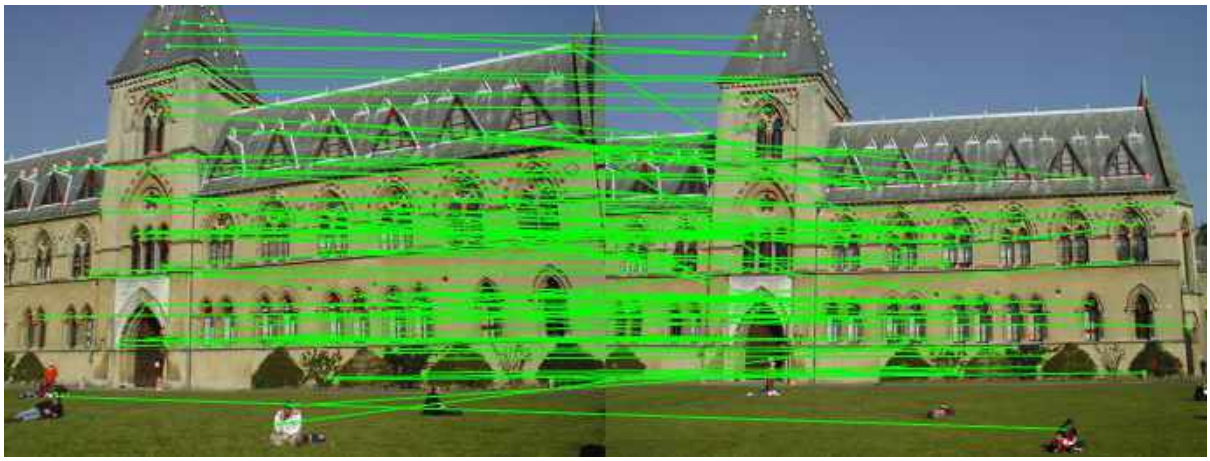

Figure 57: The matched correspondences on pair 3 images with SURF features when Hessian threshold is set to 7500

Task 2

1. Harris

Figure 58: The detected interest points on pair 4 images when $\sigma = 0.5$

Figure 59: The matched correspondences using SSD on pair 4 images when $\sigma = 0.5$

Figure 60: The matched correspondences using NCC on pair 4 images when $\sigma = 0.5$

Figure 61: The detected interest points on pair 4 images when $\sigma = 0.707$

Figure 62: The matched correspondences using SSD on pair 4 images when $\sigma = 0.707$

Figure 63: The matched correspondences using NCC on pair 4 images when $\sigma = 0.707$

Figure 64: The detected interest points on pair 4 images when $\sigma = 1$

Figure 65: The matched correspondences using SSD on pair 4 images when $\sigma = 1$

Figure 66: The matched correspondences using NCC on pair 4 images when $\sigma = 1$

Figure 67: The detected interest points on pair 4 images when $\sigma = 1.414$

Figure 68: The matched correspondences using SSD on pair 4 images when $\sigma = 1.414$

Figure 69: The matched correspondences using NCC on pair 4 images when $\sigma = 1.414$

Figure 70: The detected interest points on pair 4 images when $\sigma = 2$

Figure 71: The matched correspondences using SSD on pair 4 images when $\sigma = 2$

Figure 72: The matched correspondences using NCC on pair 4 images when $\sigma = 2$

Figure 73: The detected interest points on pair 5 images when $\sigma = 0.5$

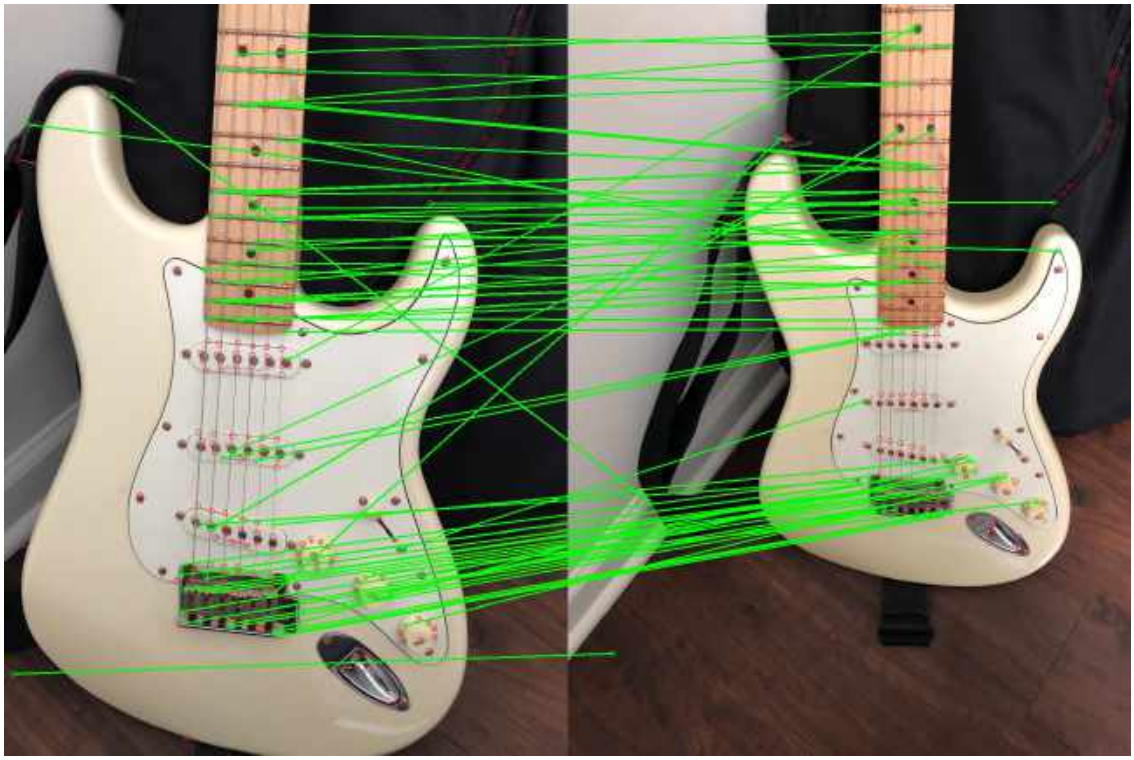

Figure 74: The matched correspondences using SSD on pair 5 images when $\sigma = 0.5$

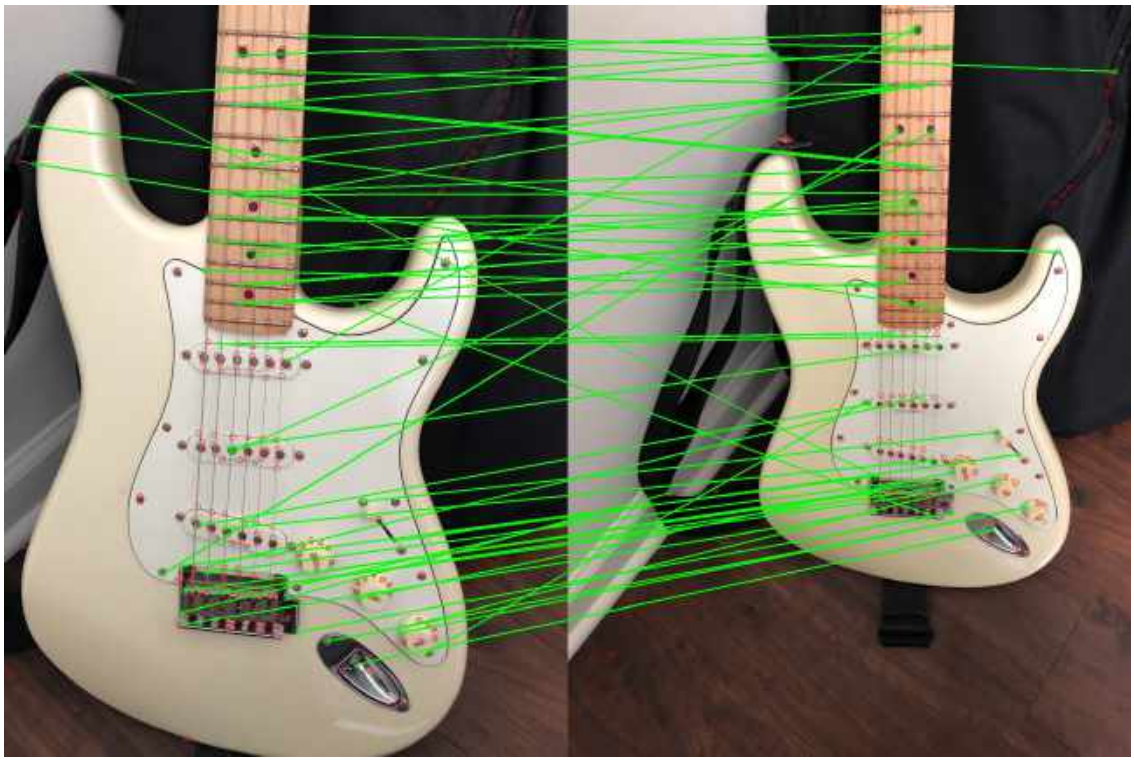

Figure 75: The matched correspondences using NCC on pair 5 images when $\sigma = 0.5$

Figure 76: The detected interest points on pair 5 images when $\sigma = 0.707$

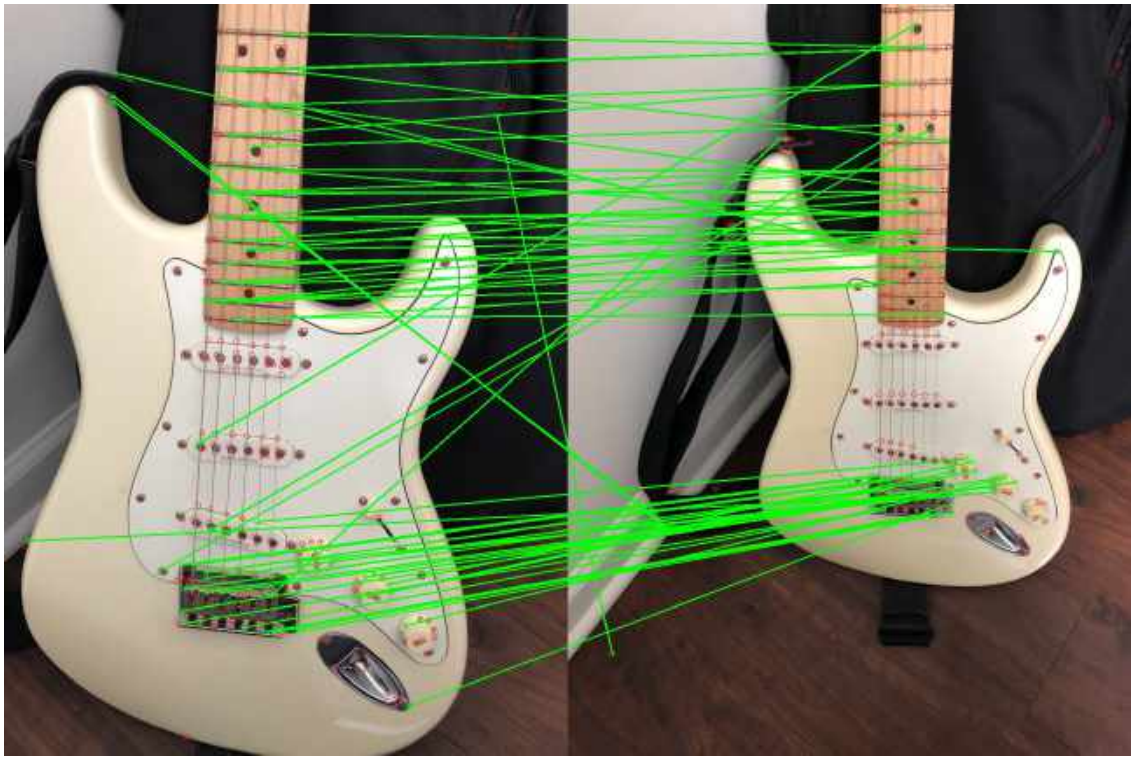

Figure 77: The matched correspondences using SSD on pair 5 images when $\sigma = 0.707$

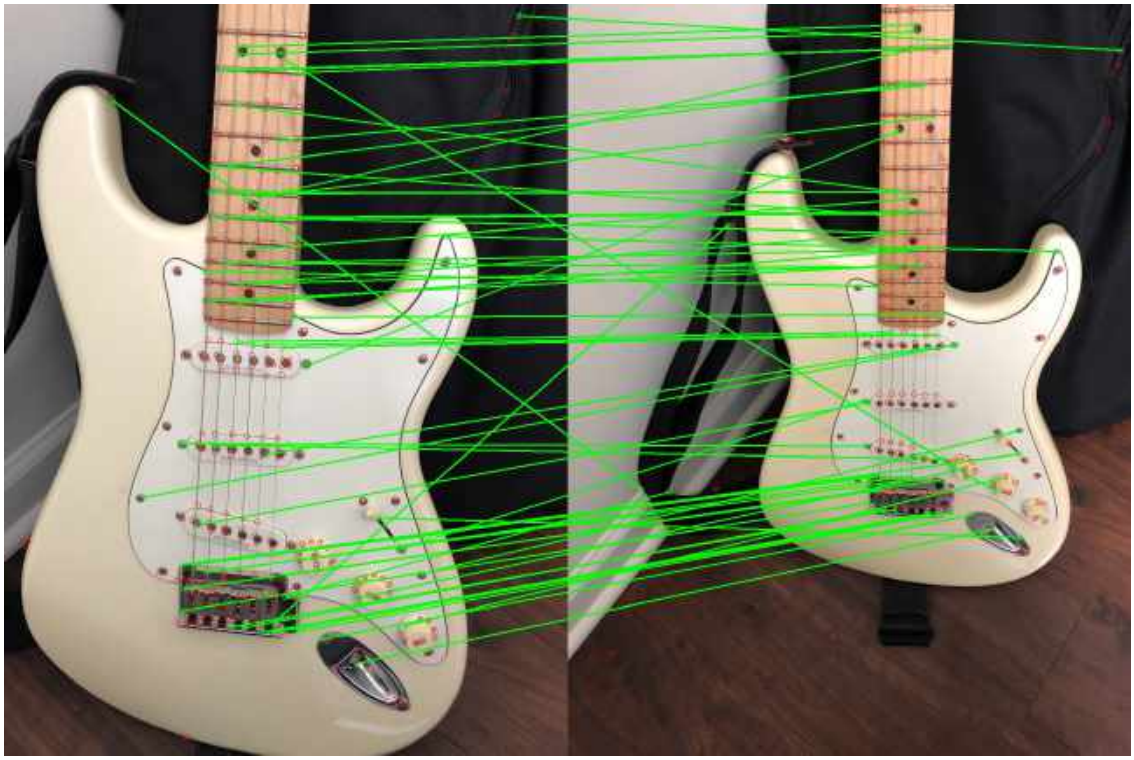

Figure 78: The matched correspondences using NCC on pair 5 images when $\sigma = 0.707$

Figure 79: The detected interest points on pair 5 images when $\sigma = 1$

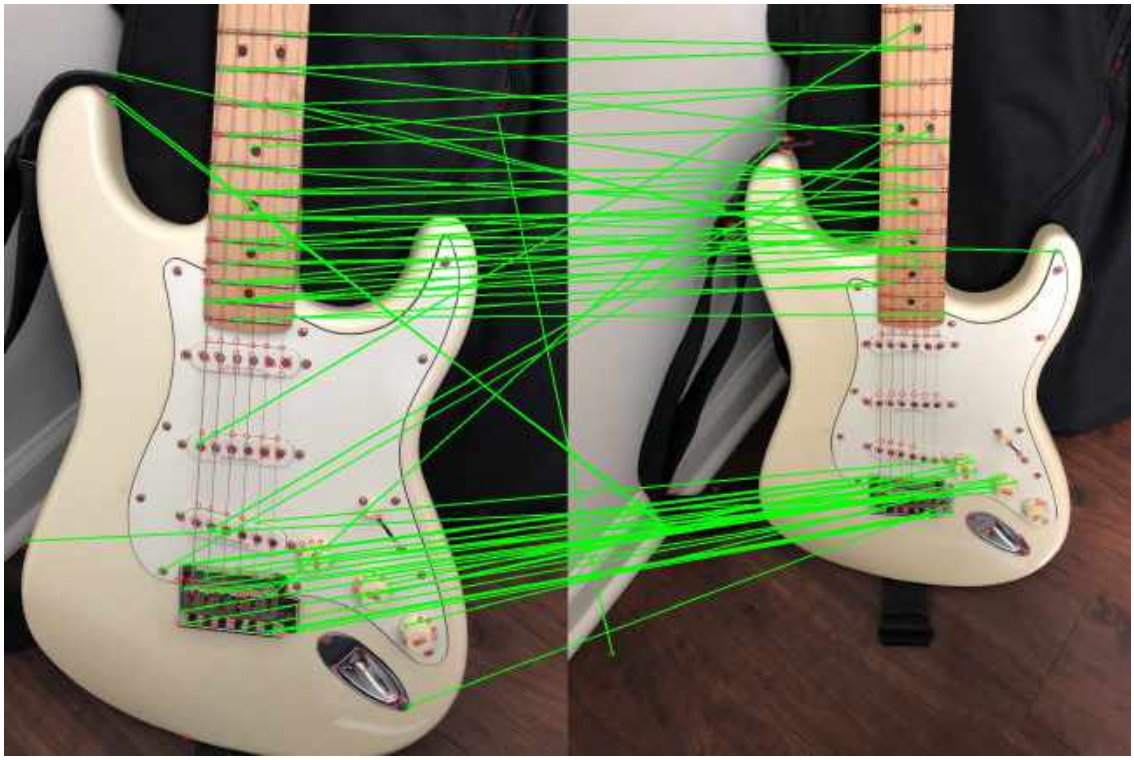

Figure 80: The matched correspondences using SSD on pair 5 images when $\sigma = 1$

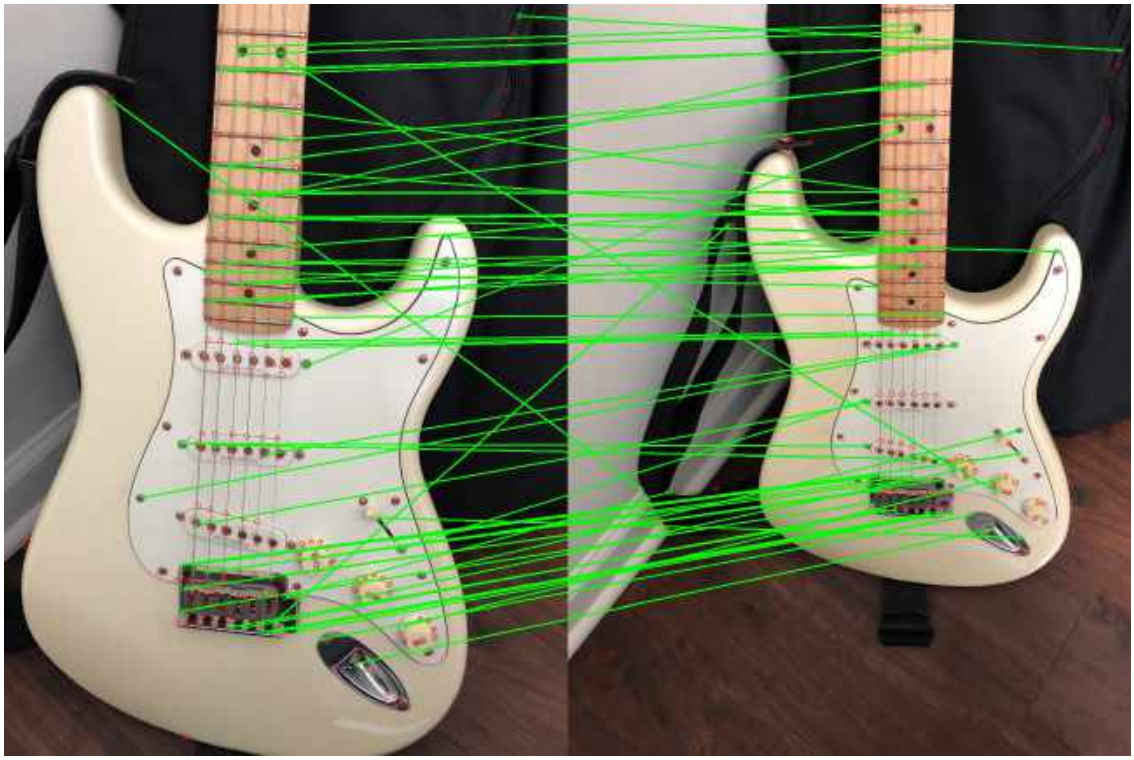

Figure 81: The matched correspondences using NCC on pair 5 images when $\sigma = 1$

Figure 82: The detected interest points on pair 5 images when $\sigma = 1.414$

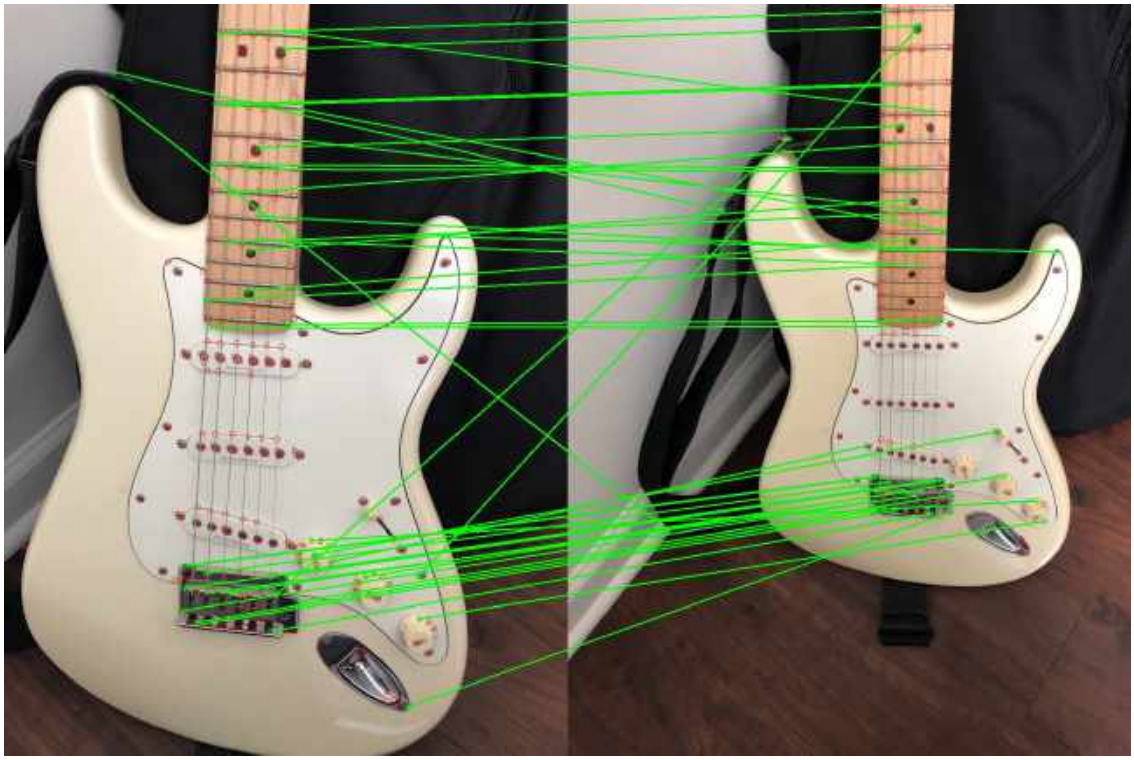

Figure 83: The matched correspondences using SSD on pair 5 images when $\sigma = 1.414$

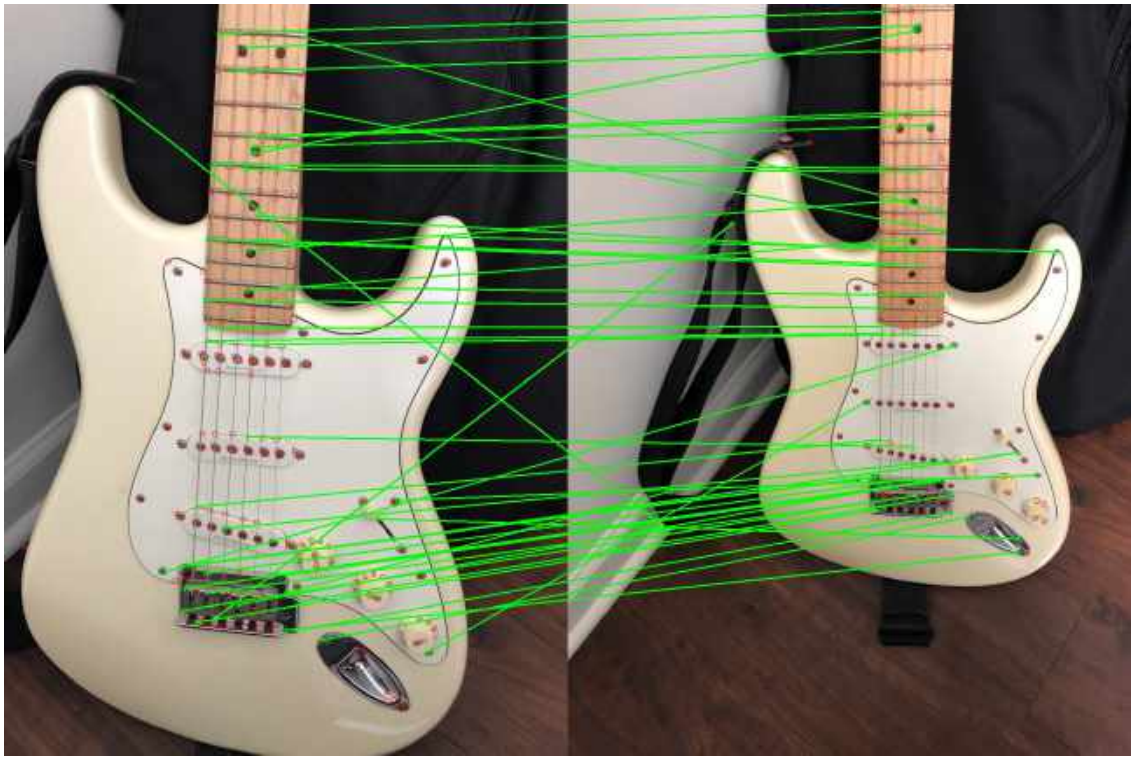

Figure 84: The matched correspondences using NCC on pair 5 images when $\sigma = 1.414$

Figure 85: The detected interest points on pair 5 images when $\sigma = 2$

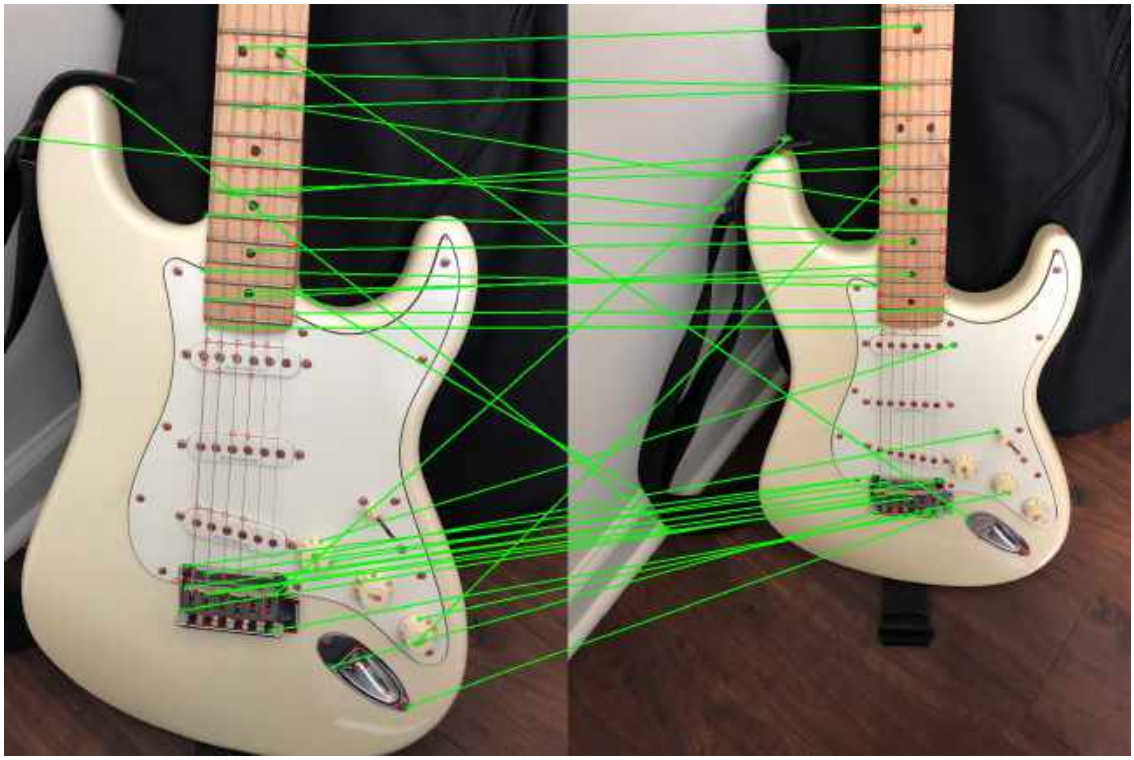

Figure 86: The matched correspondences using SSD on pair 5 images when $\sigma = 2$

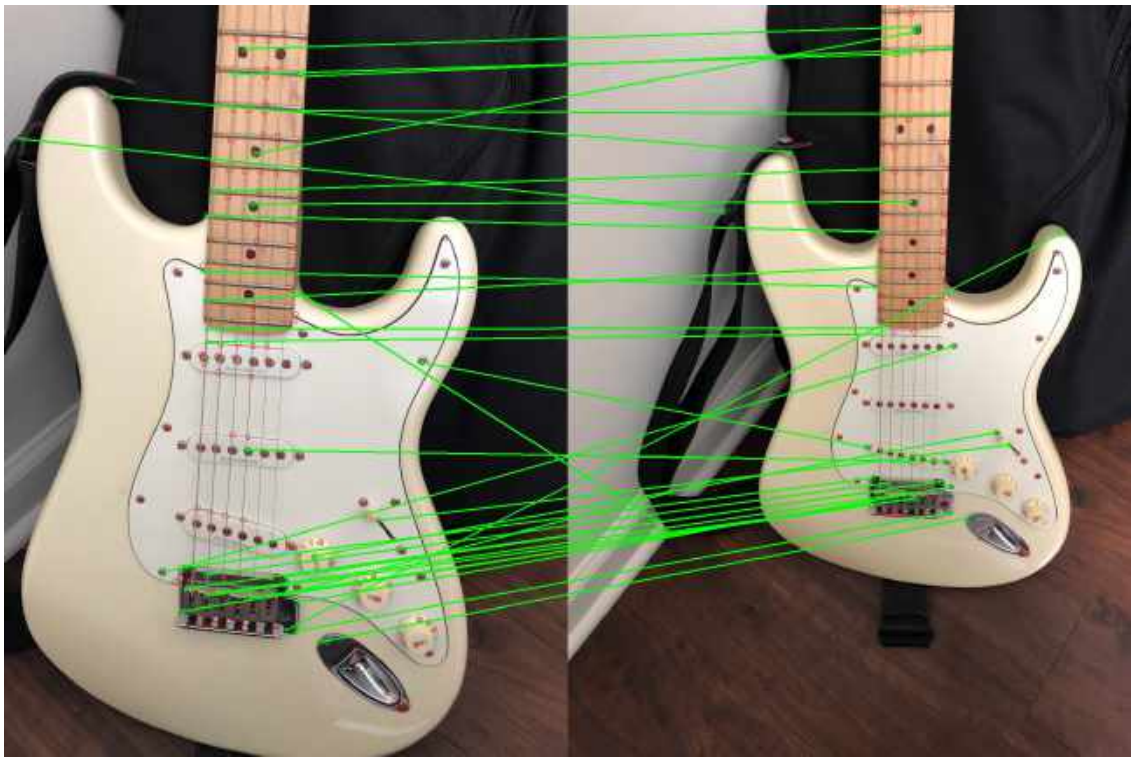

Figure 87: The matched correspondences using NCC on pair 5 images when $\sigma = 2$

2. SURF and SIFT

Figure 88: The detected interest points on pair 4 images using SIFT

Figure 89: The matched correspondences using SSD on pair 4 images with SIFT feature

Figure 90: The matched correspondences using NCC on pair 4 images with SIFT feature

Figure 91: The detected interest points on pair 4 images using SURF when Hessian threshold is set to 15000

Figure 92: The matched correspondences using SSD on pair 4 images with SURF features when Hessian threshold is set to 15000

Figure 93: The matched correspondences using NCC on pair 4 images with SURF features when Hessian threshold is set to 15000

Figure 94: The detected interest points on pair 5 images using SIFT

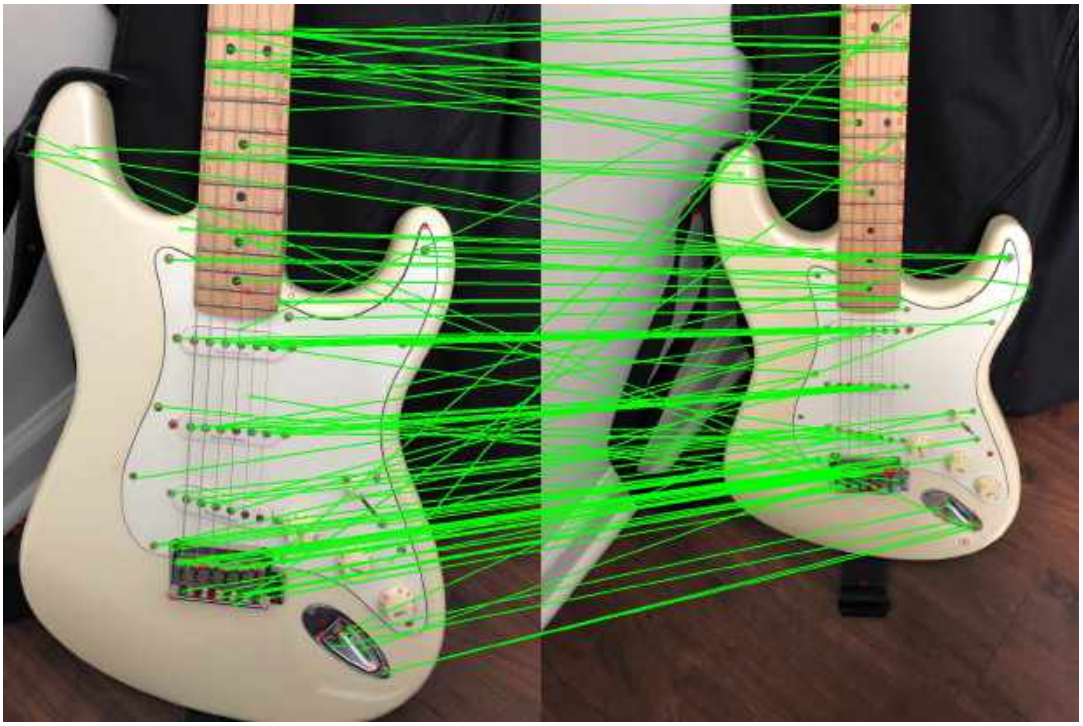

Figure 95: The matched correspondences using SSD on pair 5 images with SIFT feature

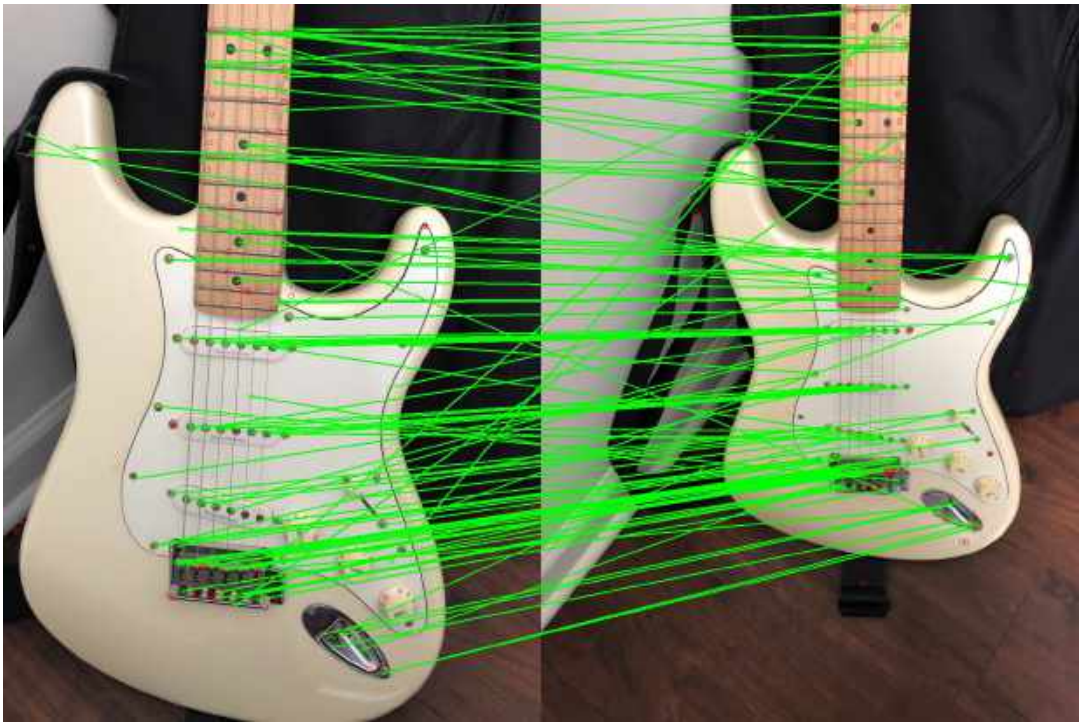

Figure 96: The matched correspondences using NCC on pair 5 images with SIFT feature

Figure 97: The detected interest points on pair 5 images using SURF when Hessian threshold is set to 5000

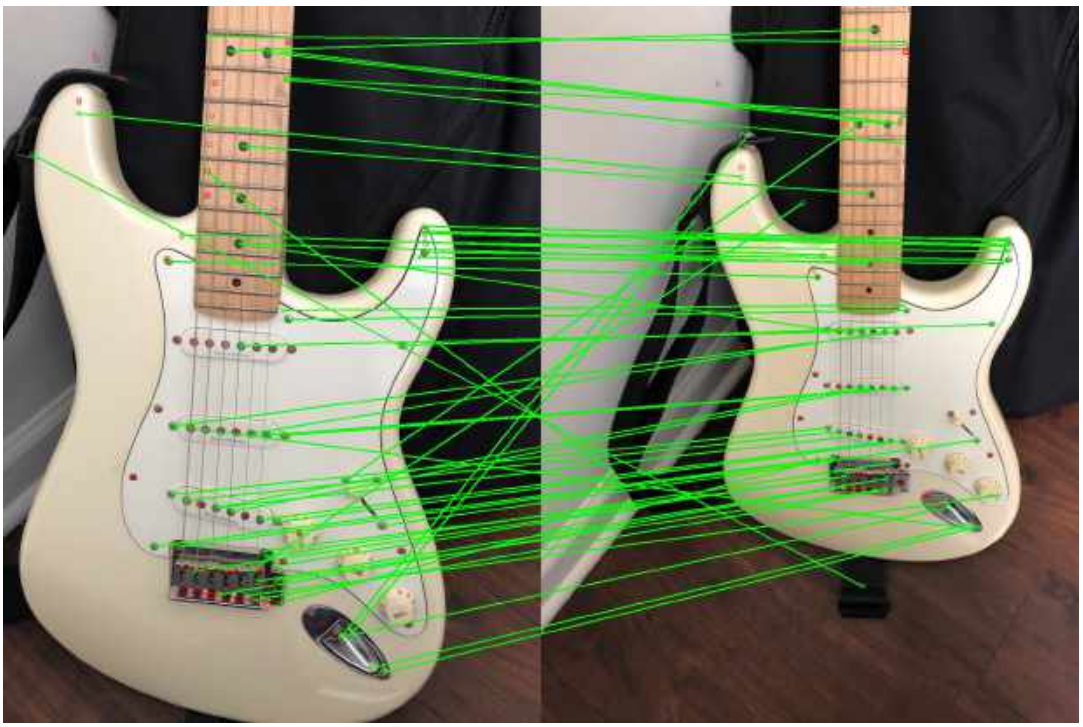

Figure 98: The matched correspondences using SSD on pair 5 images with SURF features when Hessian threshold is set to 5000

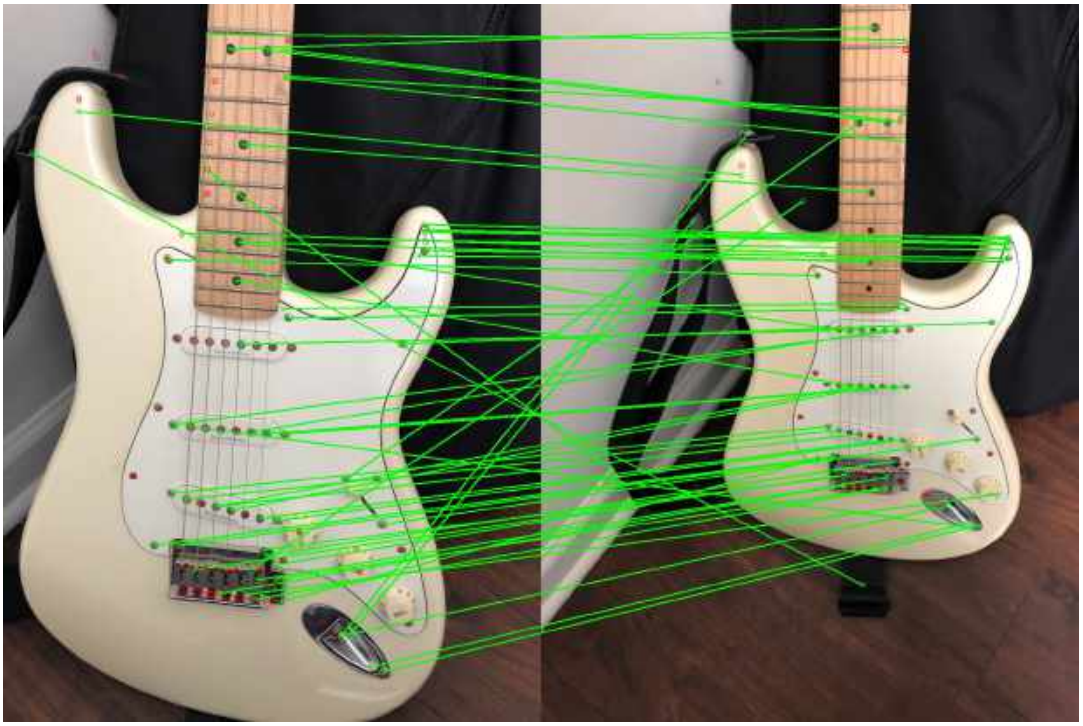

Figure 99: The matched correspondences using NCC on pair 5 images with SURF features when Hessian threshold is set to 5000

Code

```
import numpy as np
import cv2
import os
import scipy.signal as scisig


def HarrisCornerDet(img, sigma, local=1, k=0.05, filter_ratio=0.1):
    def HaarFilter(sigma=1):
        s = round(np.ceil(4 * sigma))
        if s % 2: s += 1
        Fx = np.ones((s, s))
        Fx[:, :int(s/2)] = -1
        Fy = -1 * np.copy(Fx.T)
        return Fx, Fy

    h, w = img.shape
    Fx, Fy = HaarFilter(sigma)
    Dx = scisig.correlate2d(img, Fx, mode="same")
    Dy = scisig.correlate2d(img, Fy, mode="same")
    Dx2 = Dx * Dx
    Dxy = Dx * Dy
    Dy2 = Dy * Dy
    sumSize = int(round(np.ceil(5*sigma)/2.0)*2+1)
    sumFilter = np.ones((sumSize, sumSize))

    Dx2 = scisig.correlate2d(Dx2, sumFilter, mode='same')
    Dxy = scisig.correlate2d(Dxy, sumFilter, mode='same')
```

```
    Dy2 = scisig.correlate2d(Dy2, sumFilter, mode='same')
    detC = Dx2 * Dy2 - Dxy * Dxy
    trC = Dx2 + Dy2
    R = detC - k * trC * trC
    thresh = filter_ratio * R.max()
    markerImg = np.zeros_like(R)
    for i in range(local, h-local):
        for j in range(local, w-local):
            neighbor = R[i-local:i+local+1, j-local:j+local+1]
            if R[i, j] == np.amax(neighbor) and R[i, j] >= 0:
                markerImg[i, j] = R[i, j]

    i, j = np.where(markerImg >= thresh)
    pList = [_ for _ in zip(j, i)]
    return markerImg, pList


def DrawMarker(Img, pList, color=(0,0,255)):
    result = np.copy(Img)
    for j, i in pList:
        result = cv2.circle(result, (j, i), 5, color, 1)
    return result


def HarrisFeatureMaking(img, pList, window_size):
    ws = window_size
    feature = np.zeros((len(pList), (2*ws+1)*(2*ws+1)))
    new_img = cv2.copyMakeBorder(img, ws, ws, ws, ws, cv2.BORDER_REPLICATE)
    for k , (j, i) in enumerate(pList):
        i, j = i+ws, j+ws
        neighbors = new_img[i-ws:i+ws+1, j-ws:j+ws+1]
        feature[k, :] = neighbors.reshape((1, -1))
    return feature


def cpMatching(f1, f2, method):
    d1 = f1.shape[0]
    d2 = f2.shape[0]
    if method == 'SSD':
        f11 = np.sum(f1*f1, axis=1).reshape((d1, 1))
        f12 = np.matmul(f1, f2.T)
        f22 = np.sum(f2*f2, axis=1).reshape((1, d2))
        cp = -f11.repeat(d2, axis=1) + 2 * f12 - f22.repeat(d1, axis=0)
    elif method == 'NCC':
        m1 = np.sum(f1, axis=1, keepdims=True)/f1.shape[1]
        m2 = np.sum(f2, axis=1, keepdims=True)/f2.shape[1]
        nf1 = f1 - m1
        nf2 = f2 - m2
        numerator = np.matmul(nf1, nf2.T)
        denom_1 = np.sqrt(np.sum(nf1*nf1, axis=1).reshape((d1, 1)))
        denom_2 = np.sqrt(np.sum(nf2*nf2, axis=1).reshape((1, d2)))
        denominator = np.matmul(denom_1, denom_2)
        cp = numerator / denominator
    else: print("\'Method\' has to be SSD or NCC")
    f1_Matching = np.argmax(cp, axis=1)
    f2_Matching = np.argmax(cp, axis=0)
    return cp, f1_Matching, f2_Matching
```

```
def matchedImg(img1, img2, pList1, pList2, f1_f2, f2_f1):
    def drawLines(img, p_start, p_end, color=(0,255,0)):
        image = np.copy(img)
        for i in range(len(p_start)):
            image = cv2.circle(image, p_start[i], 5, color, 1)
            image = cv2.circle(image, p_end[i], 5, color, 1)
            image = cv2.line(image, p_start[i], p_end[i], color, 2)
        return image

    im1 = np.copy(img1)
    im2 = np.copy(img2)

    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    if h1 > h2:
        im2 = cv2.copyMakeBorder(im2, 0, h1-h2, 0, 0, cv2.BORDER_REPLICATE)
        h = h1
    elif h1 < h2:
        im1 = cv2.copyMakeBorder(im1, 0, h2-h1, 0, 0, cv2.BORDER_REPLICATE)
        h = h2
    else: h = h1

    if w1 > w2:
        im2 = cv2.copyMakeBorder(im2, 0, 0, 0, w1-w2, cv2.BORDER_REPLICATE)
        w = w1
    elif w1 < w2:
        im1 = cv2.copyMakeBorder(im1, 0, 0, 0, w2-w1, cv2.BORDER_REPLICATE)
        w = w2
    else: w = w1

    concatIm = np.concatenate((im1, im2), axis=1)
    npList2 = [(j+w, i) for (j, i) in pList2]
    concatIm = DrawMarker(concatIm, pList1, color=(255,0,0))
    concatIm = DrawMarker(concatIm, npList2, color=(255,0,0))

    bestMatch_1 = []
    bestMatch_2 = []
    for i, idx2 in enumerate(f1_f2):
        if i == f2_f1[idx2]:
            bestMatch_1.append(pList1[i])
            bestMatch_2.append(npList2[idx2])
    print('detected '+str(len(pList1))+ ' corners in image 1')
    print('detected '+str(len(pList2))+ ' corners in image 2')
    print('found '+str(len(bestMatch_1))+ ' matched corners between image 1 and 2'
    )
    matchedIm = drawLines(concatIm, bestMatch_1, bestMatch_2, color=(0,255,0))
    return concatIm, matchedIm


def taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.05):
    g_1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
    g_2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
    for sigma in sigmas:
        _, responses_1 = HarrisCornerDet(g_1, sigma=sigma, local=7, k=0.05,
   filter_ratio=filter_ratio)
        _, responses_2 = HarrisCornerDet(g_2, sigma=sigma, local=7, k=0.05,
   filter_ratio=filter_ratio)
        f1 = HarrisFeatureMaking(g_1, pList=responses_1, window_size=20)
        f2 = HarrisFeatureMaking(g_2, pList=responses_2, window_size=20)
```

```
        for M in ['SSD', 'NCC']:
            print(M, ', sigma:', sigma)
            cp, f1_f2, f2_f1 = cpMatching(f1, f2, method=M)
            concatIm, matchedIm = matchedImg(img1, img2, responses_1, responses_2,
    f1_f2, f2_f1)
            concatIm = cv2.cvtColor(concatIm, cv2.COLOR_RGB2BGR)
            concat_out = os.path.join(folder_result, 'Harris_'+str(sigma)+'.jpg')
            matchedIm = cv2.cvtColor(matchedIm, cv2.COLOR_RGB2BGR)
            match_out = os.path.join(folder_result, 'HarrisMatch_'+str(sigma)+'_'+
   M+'.jpg')
            cv2.imwrite(match_out, matchedIm)
        cv2.imwrite(concat_out, concatIm)


def taskSIFT(img1, img2, nfeatures):
    g_1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
    g_2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)

    sift = cv2.xfeatures2d.SIFT_create(nfeatures)
    kp1, des1 = sift.detectAndCompute(g_1, None)
    kp2, des2 = sift.detectAndCompute(g_2, None)
    pList1 = [(int(kp1[i].pt[0]), int(kp1[i].pt[1])) for i in range(len(kp1))]
    pList2 = [(int(kp2[i].pt[0]), int(kp2[i].pt[1])) for i in range(len(kp2))]
    for M in ['SSD', 'NCC']:
        print(M)
        _, f1_f2, f2_f1 = cpMatching(des1, des2, method=M)
        concatIm, matchedIm = matchedImg(img1, img2, pList1, pList2, f1_f2, f2_f1)
        concatIm = cv2.cvtColor(concatIm, cv2.COLOR_RGB2BGR)
        concat_out = os.path.join(folder_result, 'SIFT.jpg')
        matchedIm = cv2.cvtColor(matchedIm, cv2.COLOR_RGB2BGR)
        match_out = os.path.join(folder_result, 'SIFTmatch_'+M+'.jpg')
        cv2.imwrite(match_out, matchedIm)
    cv2.imwrite(concat_out, concatIm)


def taskSURF(img1, img2, hessianThreshold):
    g_1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
    g_2 = cv2.cvtColor(img2, cv2.COLOR_RGB2GRAY)
    print(hessianThreshold)
    surf = cv2.xfeatures2d.SURF_create(hessianThreshold)
    kp1, des1 = surf.detectAndCompute(g_1, None)
    kp2, des2 = surf.detectAndCompute(g_2, None)
    pList1 = [(int(kp1[i].pt[0]), int(kp1[i].pt[1])) for i in range(len(kp1))]
    pList2 = [(int(kp2[i].pt[0]), int(kp2[i].pt[1])) for i in range(len(kp2))]
    for M in ['SSD', 'NCC']:
        print(M)
        _, f1_f2, f2_f1 = cpMatching(des1, des2, method=M)
        concatIm, matchedIm = matchedImg(img1, img2, pList1, pList2, f1_f2, f2_f1)
        concatIm = cv2.cvtColor(concatIm, cv2.COLOR_RGB2BGR)
        concat_out = os.path.join(folder_result, 'SURF'+str(hessianThreshold)+'.
   jpg')
        matchedIm = cv2.cvtColor(matchedIm, cv2.COLOR_RGB2BGR)
        match_out = os.path.join(folder_result, 'SURFmatch_'+M+str(
   hessianThreshold)+'.jpg')
        cv2.imwrite(match_out, matchedIm)
    cv2.imwrite(concat_out, concatIm)


folder_task = '/home/xingguang/Documents/ECE661/hw4/hw4_Task1_Images'
```

```
folder_task2 = '/home/xingguang/Documents/ECE661/hw4/hw4_Task2_Images'
path_1_1 = os.path.join(folder_task, 'pair1', '1.JPG')
path_1_2 = os.path.join(folder_task, 'pair1', '2.JPG')
path_2_1 = os.path.join(folder_task, 'pair2', '1.JPG')
path_2_2 = os.path.join(folder_task, 'pair2', '2.JPG')
path_3_1 = os.path.join(folder_task, 'pair3', '1.jpg')
path_3_2 = os.path.join(folder_task, 'pair3', '2.jpg')
path_4_1 = os.path.join(folder_task2, 'pair1', '1.jpg')
path_4_2 = os.path.join(folder_task2, 'pair1', '2.jpg')
path_5_1 = os.path.join(folder_task2, 'pair2', '1.jpg')
path_5_2 = os.path.join(folder_task2, 'pair2', '2.jpg')

img_1_1 = cv2.cvtColor(cv2.imread(path_1_1), cv2.COLOR_BGR2RGB)
img_1_2 = cv2.cvtColor(cv2.imread(path_1_2), cv2.COLOR_BGR2RGB)
img_2_1 = cv2.cvtColor(cv2.imread(path_2_1), cv2.COLOR_BGR2RGB)
img_2_2 = cv2.cvtColor(cv2.imread(path_2_2), cv2.COLOR_BGR2RGB)
img_3_1 = cv2.cvtColor(cv2.imread(path_3_1), cv2.COLOR_BGR2RGB)
img_3_2 = cv2.cvtColor(cv2.imread(path_3_2), cv2.COLOR_BGR2RGB)
img_4_1 = cv2.cvtColor(cv2.imread(path_4_1), cv2.COLOR_BGR2RGB)
img_4_2 = cv2.cvtColor(cv2.imread(path_4_2), cv2.COLOR_BGR2RGB)
img_5_1 = cv2.cvtColor(cv2.imread(path_5_1), cv2.COLOR_BGR2RGB)
img_5_2 = cv2.cvtColor(cv2.imread(path_5_2), cv2.COLOR_BGR2RGB)
sigmas = [0.5, 0.707, 1, 1.414, 2]

folder_result = os.path.join('/home/xingguang/Documents/ECE661/hw4/Results', '
    pair1')
img1 = img_1_1
img2 = img_1_2
taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.02)
taskSIFT(img1, img2, 300)
taskSURF(img1, img2, 2000)


folder_result = os.path.join('/home/xingguang/Documents/ECE661/hw4/Results', '
    pair2')
img1 = img_2_1
img2 = img_2_2
taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.15)
taskSIFT(img1, img2, 200)
taskSURF(img1, img2, 20000)


folder_result = os.path.join('/home/xingguang/Documents/ECE661/hw4/Results', '
    pair3')
img1 = img_3_1
img2 = img_3_2
taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.05)
taskSIFT(img1, img2, 200)
taskSURF(img1, img2, 7500)


folder_result = os.path.join('/home/xingguang/Documents/ECE661/hw4/Results', '
    pair4')
img1 = img_4_1
img2 = img_4_2
taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.02)
taskSIFT(img1, img2, 200)
taskSURF(img1, img2, 15000)
```

```
folder_result = os.path.join('/home/xingguang/Documents/ECE661/hw4/Results', '
    pair5')
img1 = img_5_1
img2 = img_5_2
taskHarris(img1, img2, sigmas, folder_result, filter_ratio=0.01)
taskSIFT(img1, img2, 200)
taskSURF(img1, img2, 5000)
```