Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Theory

Harris Corner Detector

Haar filters with a size of k x k, such that k is the smallest even integer greater than 4$\boldsymbol{\sigma}$ are used, where $\boldsymbol{\sigma}$ is a parameter to the Harris corner detector. For example, if $\boldsymbol{\sigma}$ = 1.2, the Haar filters would be of dimension 6 x 6.

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

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

The entire image is convolved with these filters to obtain the derivatives along the x and y directions separately. Next, for each pixel, a matrix C is computed using these derivative matrices. A window size of N x N is used, where N is an odd integer as close to 5 $\boldsymbol{\sigma}$ x 5 $\boldsymbol{\sigma}$ as possible.

$$C = \begin{bmatrix} \sum x^2 & \sum xy \\ \sum xy & \sum y2 \end{bmatrix}$$

An interest point is found when the ratio of the small eigenvalue lambda_1 to the ratio of the larger eigenvalue lambda_2 is above a certain threshold. Since calculating the eigenvalues is computationally intensive, $\frac{\det(C)}{trace(C)2} = \frac{r}{(1+r)\text{^}2}$ is used to achieve the same thresholding.

After this, the entire matrix of Cs is iterated through, checking if the ratio of that particular pixel value is the greatest in a window of a size determined as a hyperparameter (29 is used in my case). If its ratio is the greatest and that ratio is above the threshold, then that pixel is declared as a corner point. Corner points at the edges of the image are removed, because they may seem to be falsely high-contrast.

Scale Invariant Feature Transform (SIFT)

The actual implementation was done using the VLFeat library in MTALAB. To approximate the Laplacian of an image at scale $\boldsymbol{\sigma}$, the Difference of Gaussian (D0G) pyramid is calculated.

D(x,y, $\boldsymbol{\sigma}$) = ff(x,y, $\boldsymbol{\sigma}$1) – ff(x,y, $\boldsymbol{\sigma}$2),

Where ff(x,y, $\boldsymbol{\sigma}$n) is the image after it is convolved with the Gaussian filter:

$$G(x,y,\boldsymbol{\sigma}) = \frac{1}{2*pi*\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

$\boldsymbol{\sigma}1$ and $\boldsymbol{\sigma}2$ are two closely spaced values of $\boldsymbol{\sigma}$. SIFT interest points are points of local extrema in the D(x,y, $\boldsymbol{\sigma}$) scale space. Since DoG only gives us an approximation, Taylor series expansion is used to approximate the scale space near the extrema, for sub-pixel precision.

$D(x,y, \boldsymbol{\sigma}) = D(X_0) + J^T(X_0)X + 0.5*x^TH(X_0)X$

X is a varying step size from X_0. J is the Jacobian, and H is the Hessian. Points for which $|D(X)| < 0.03$ are removed to exclude low-contrast key points. Also, points at the edges of the image are removed because they may seem to be falsely high contrast because there is nothing around them.

The dominant local orientation for each of these interest points is found by using the gradient vector of the Gaussian-smoothed image, whose magnitude is :

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

And orientation is :

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

The 16 x 16 neighborhood of the pixel is divided into 4 x 4 cells, each cell having 4 x 4 points. An 8-bin orientation histogram is calculated for each cell, which when strung together, produce a 128-element descriptor, which is the output feature vector at the extemum point.

Establishing correspondences:

Once corner points of both scenes are found, correspondences between those corner points can be established using the following similarity metrics:

- Sum of Squared Differences (SSD):
  $SSD = \sum\sum((f1(i,j) - f2(i,j))^2)$
  Where fk(i, j) is the sum of the pixels within a window of the pixel being considered from image k. The smaller the SSD, the more accurate the correspondence is. This was used to find correspondences between corners detected using the Harris corner detector.

- Normalized Cross-Correlation (NCC):
  $NCC = \sum\sum((f1(i,j) - m1)(f2(i,j) - m2))/\sqrt{\sum\sum(f1(i,j) - m1)^2 * \sum\sum(f2(i,j) - m2)^2}$
  Where m1 and m2 are the means of the pixel values in that particular window in scenes 1 and 2 respectively. The larger the NCC, the more accurate the correspondence. This was also used to find correspondences between corners detected using the Harris corner detector.
- Euclidean distance:
  The Euclidean distance between the two feature vectors outputted by the SIFT algorithm is simply the l2-norm of the difference between corresponding elements. A smaller Euclidean distance means a more accurate correspondence.

For each corner point in the first scene, the metrics were calculated corresponding to each corner in the second scene. The best match for each corner point in the first image was taken only if, for each coordinate, the absolute difference between the two points was lesser than a threshold.

Results

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Input images

Pair 1

Pair 2

Pair 3

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

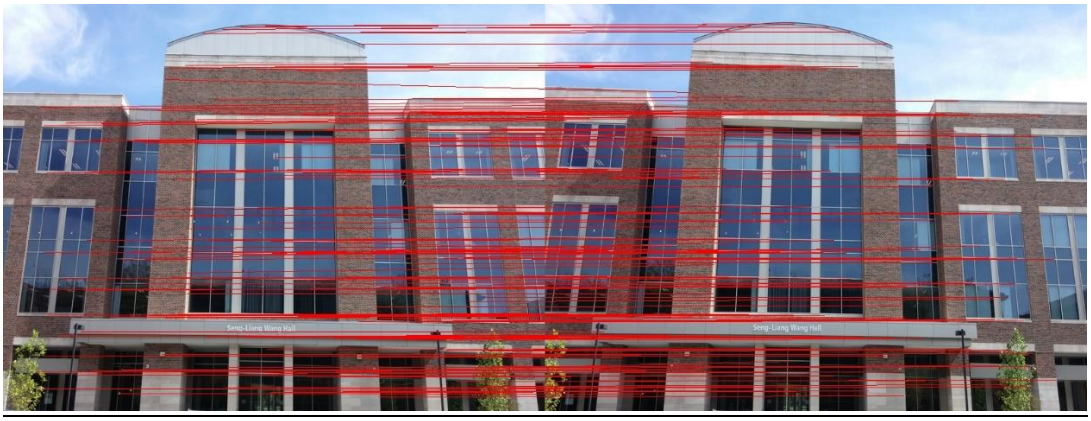
Pair 4

Output images

Pair 1:

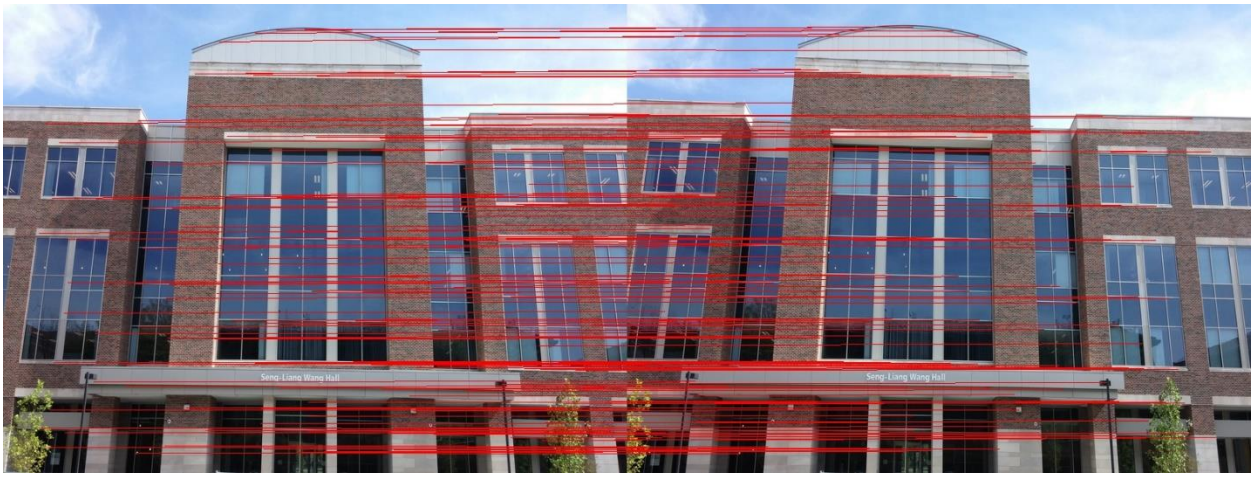
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with NCC, sigma = 1.2



Harris detector with NCC, sigma = 1.5

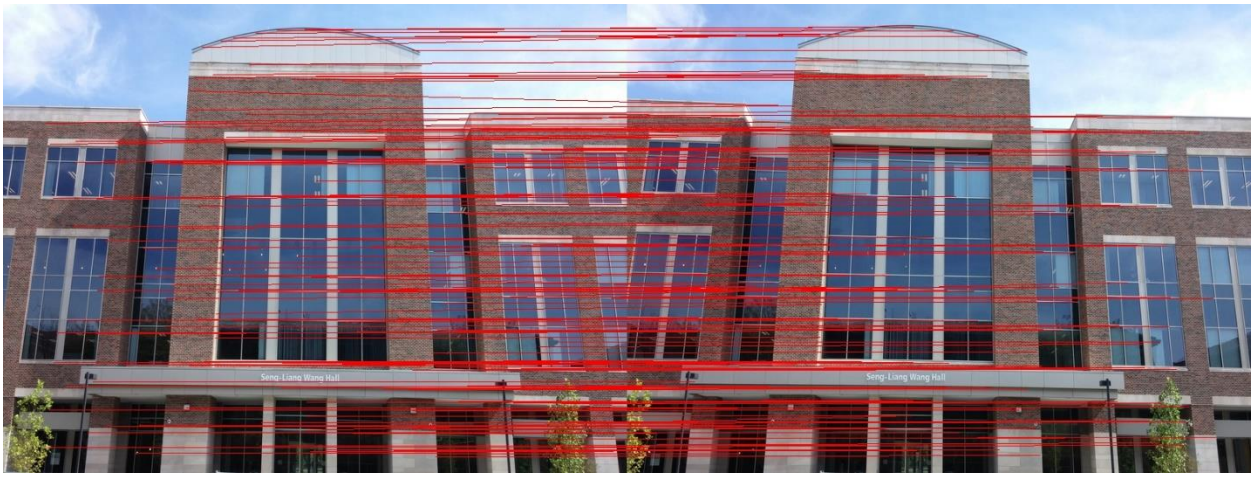
Harris detector with NCC, sigma = 1.8



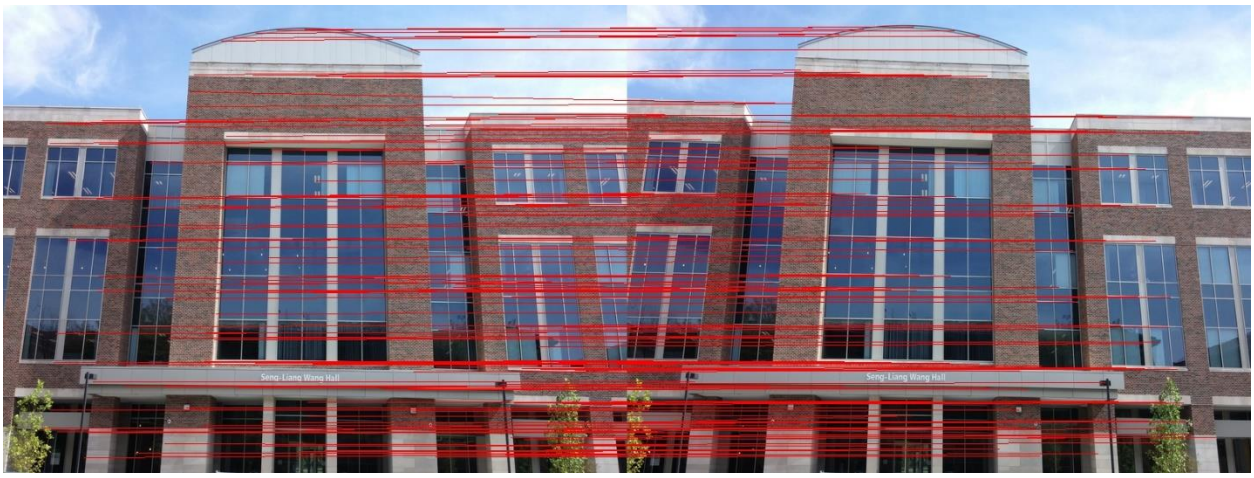
Harris detector with NCC, sigma = 2.1

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with SSD, sigma = 1.2

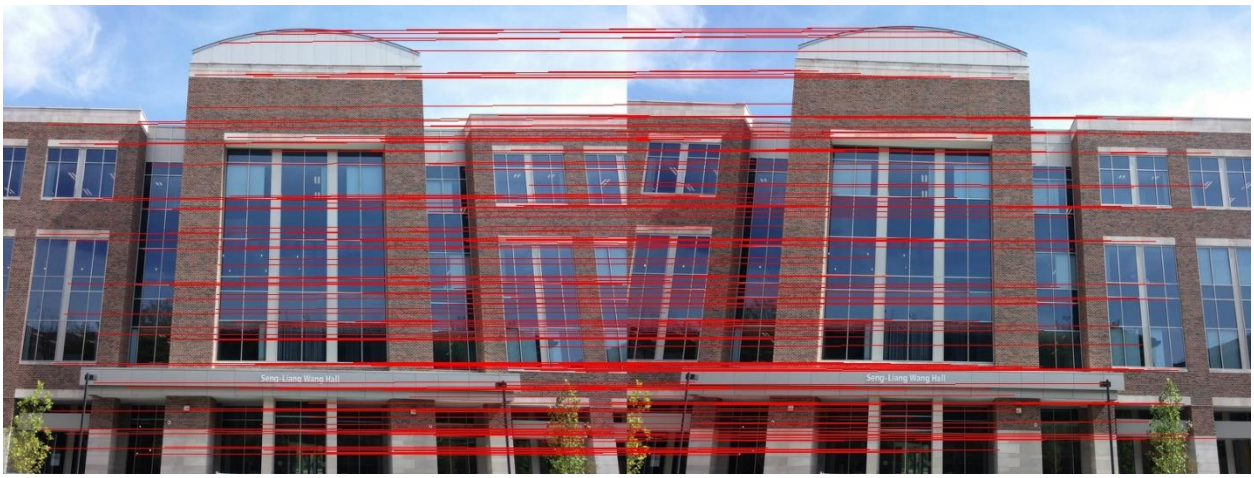


Harris detector with SSD, sigma = 1.5



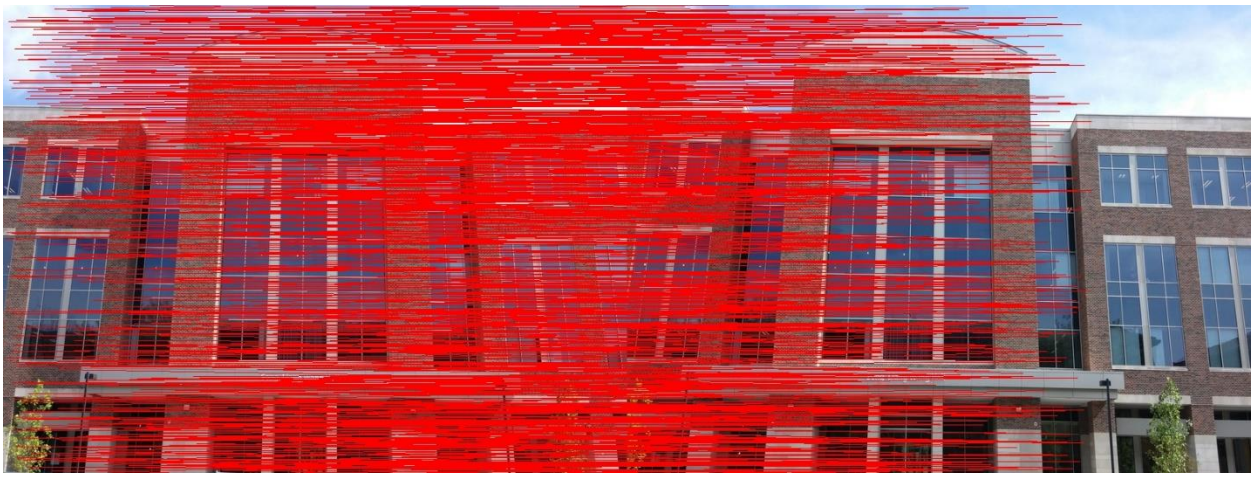
Harris detector with SSD, sigma = 1.8

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04



Harris detector with SSD, sigma = 2.1

SIFT points matched using Euclidean distance



Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Pair 2:

Harris detector with NCC, sigma = 1.2

Harris detector with NCC, sigma = 1.5

Harris detector with NCC, sigma = 1.8

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with NCC, sigma = 2.1

Harris detector with SSD, sigma = 1.2

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with SSD, sigma = 1.5

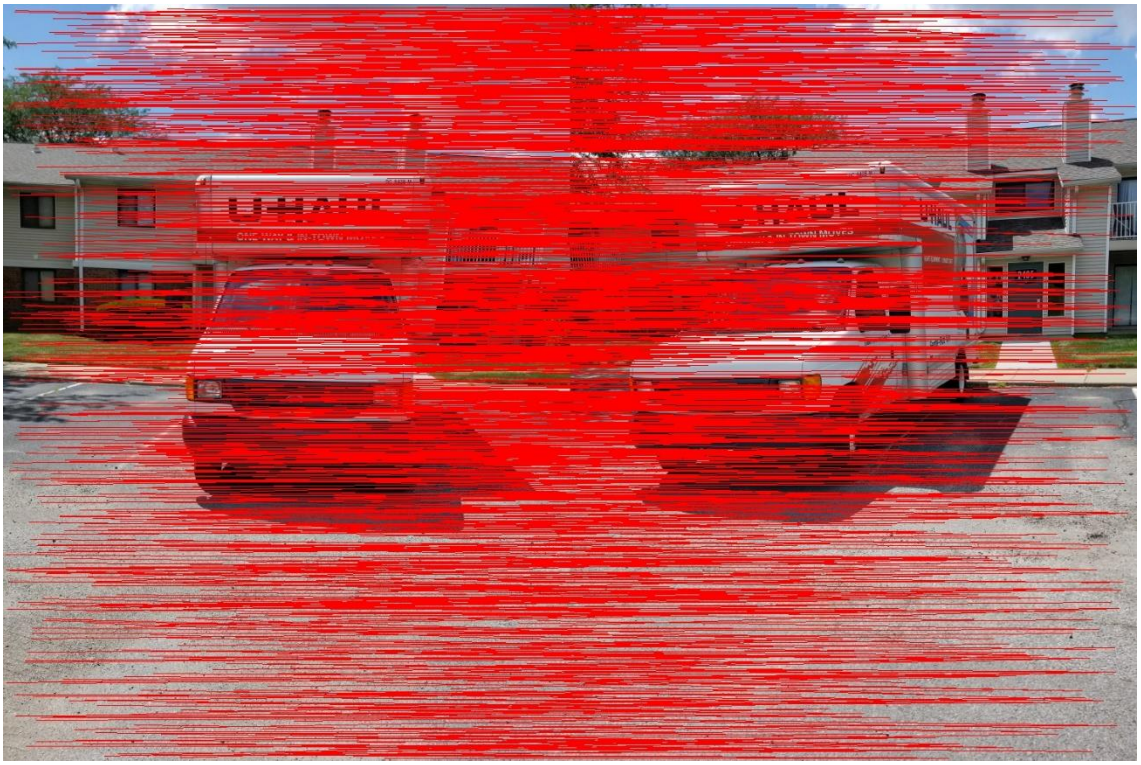
Harris detector with SSD, sigma = 1.8

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with SSD, sigma = 2.1

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

SIFT points matched using Euclidean distance



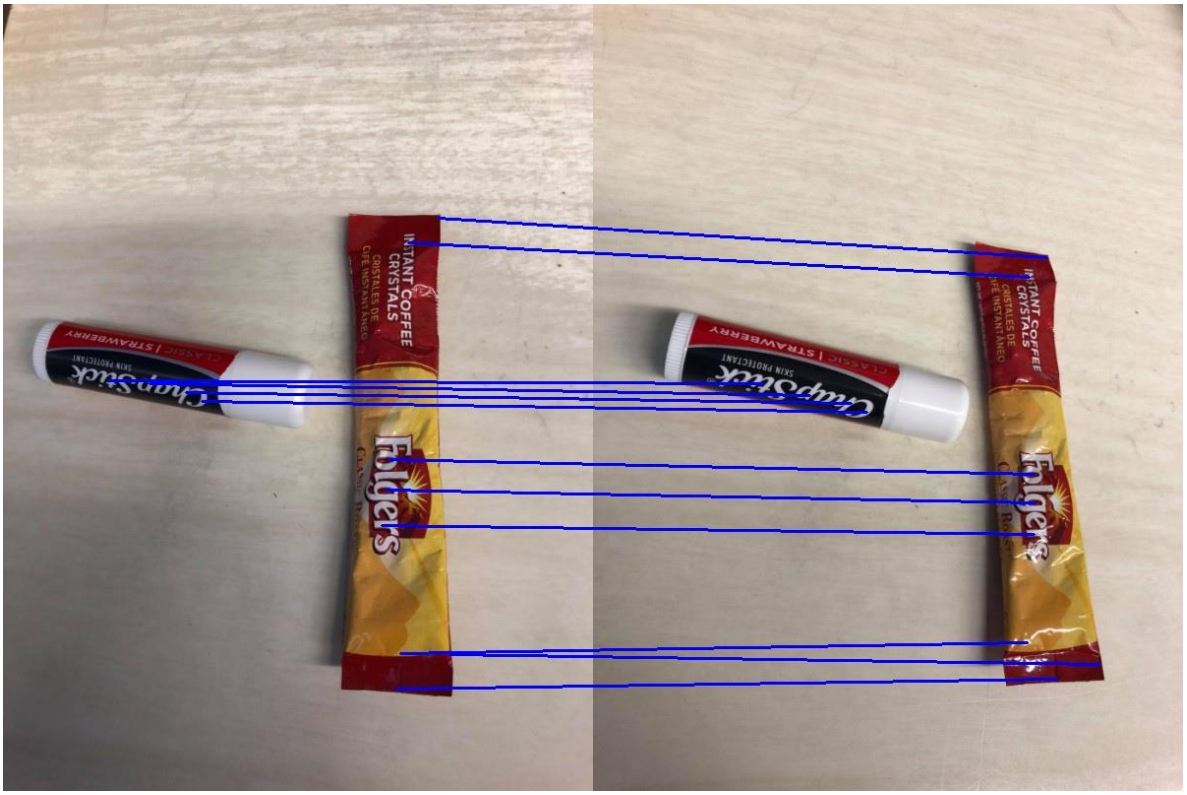
Pair 3:

Harris detector with NCC, sigma =1.2

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

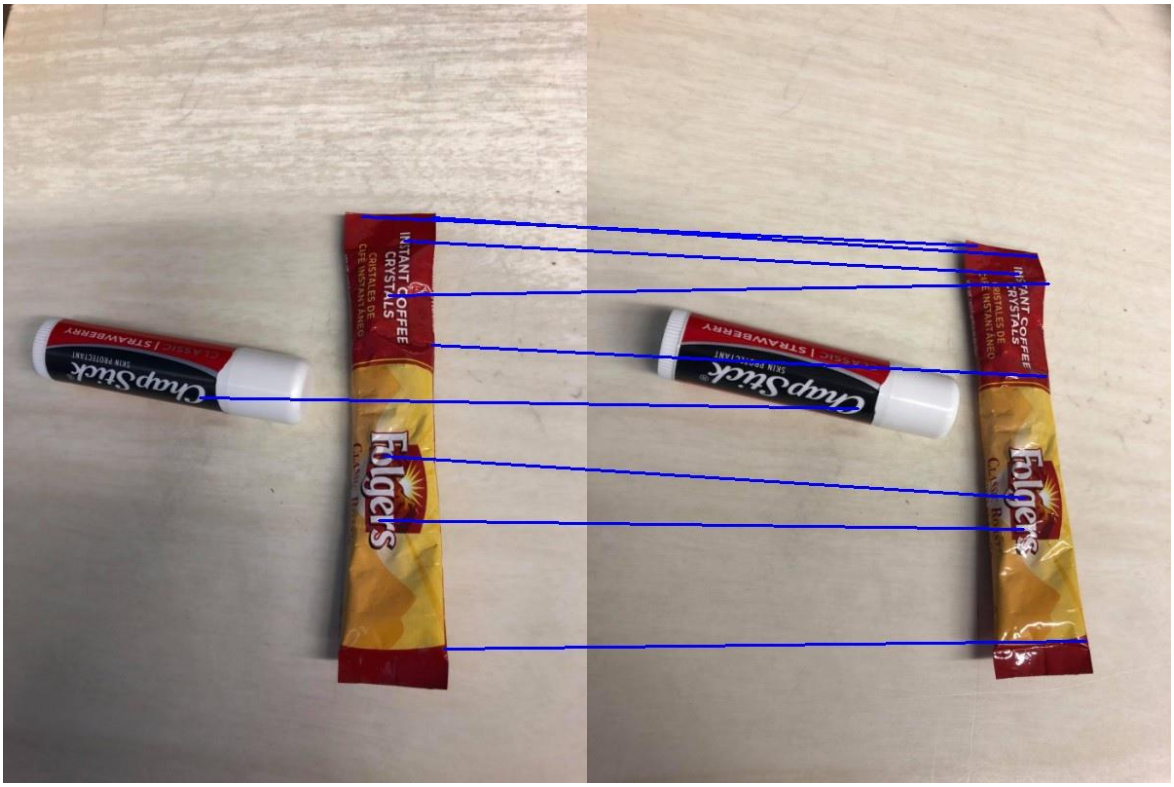
Harris detector with NCC, sigma = 1.5

Harris detector with NCC, sigma = 1.8

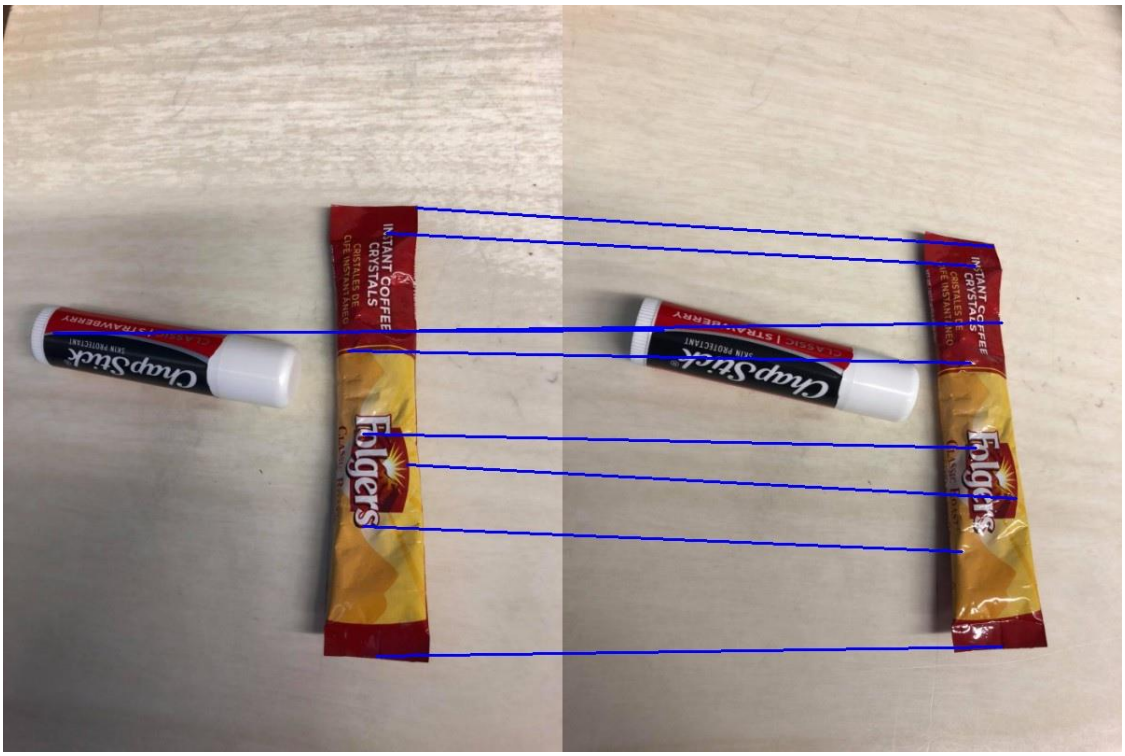


Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with NCC, sigma = 2.1



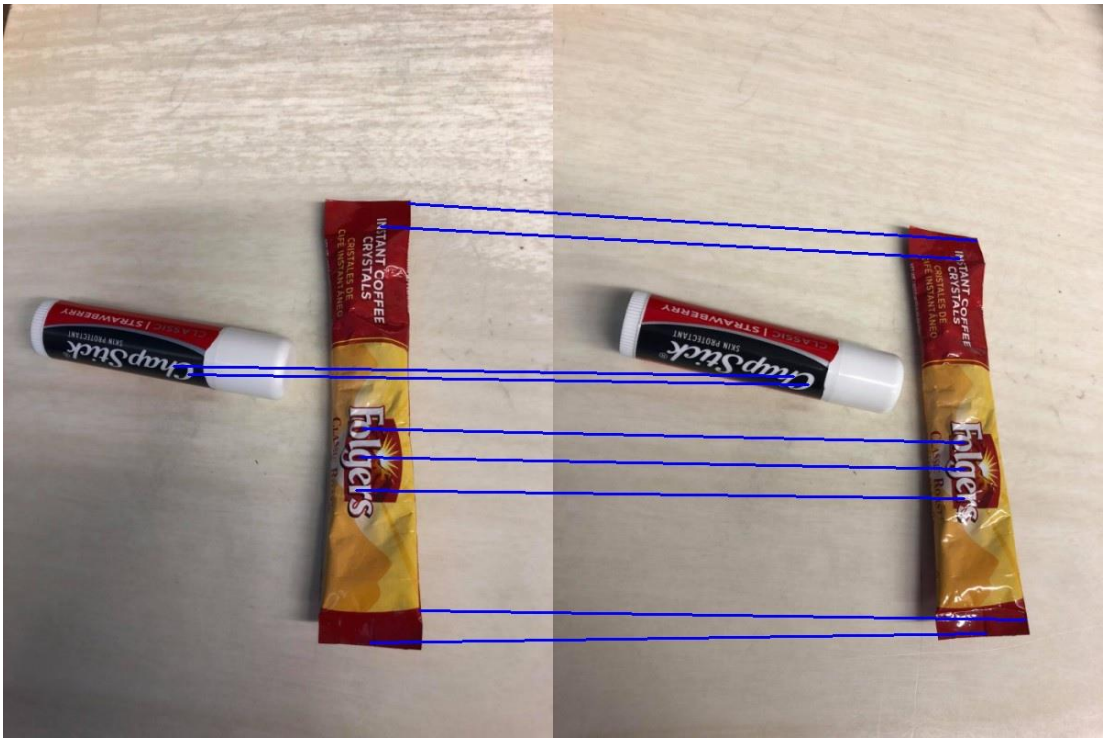
Harris detector with SSD, sigma =1.2



Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with SSD, sigma = 1.5

Harris detector with SSD, sigma = 1.8



Harris detector with SSD, sigma = 2.1

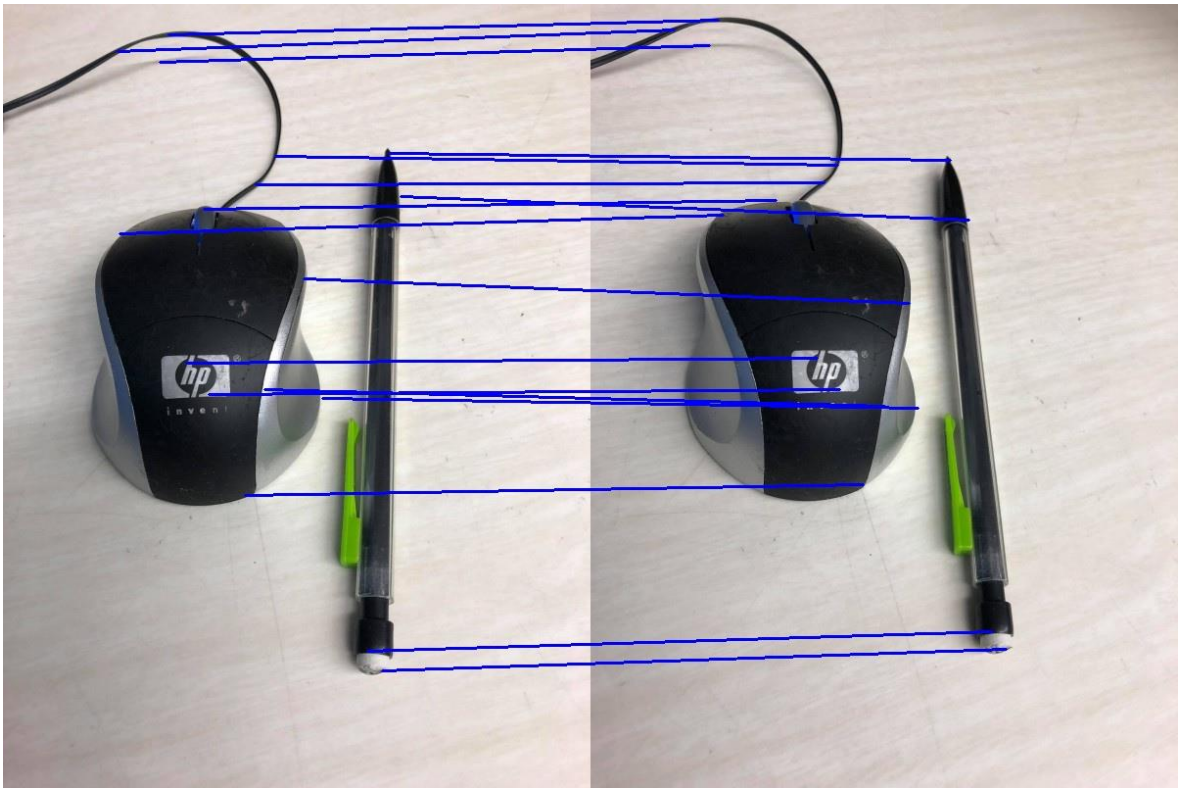
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

SIFT points matched using Euclidean distance

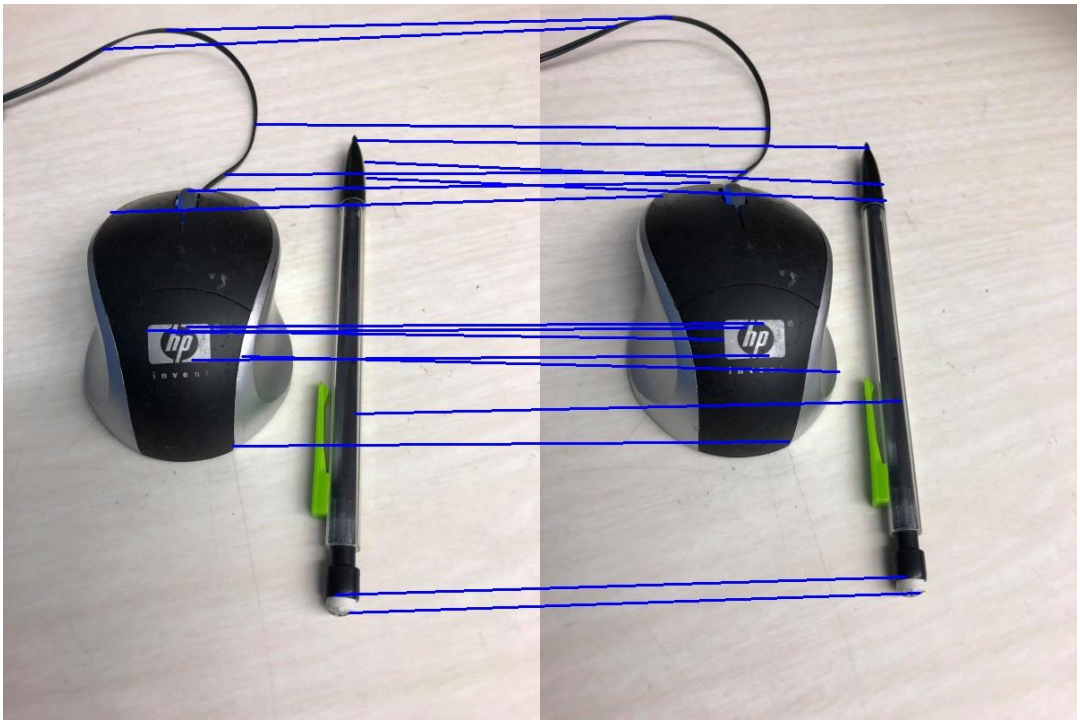
Pair 4:

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with NCC, sigma = 1.2

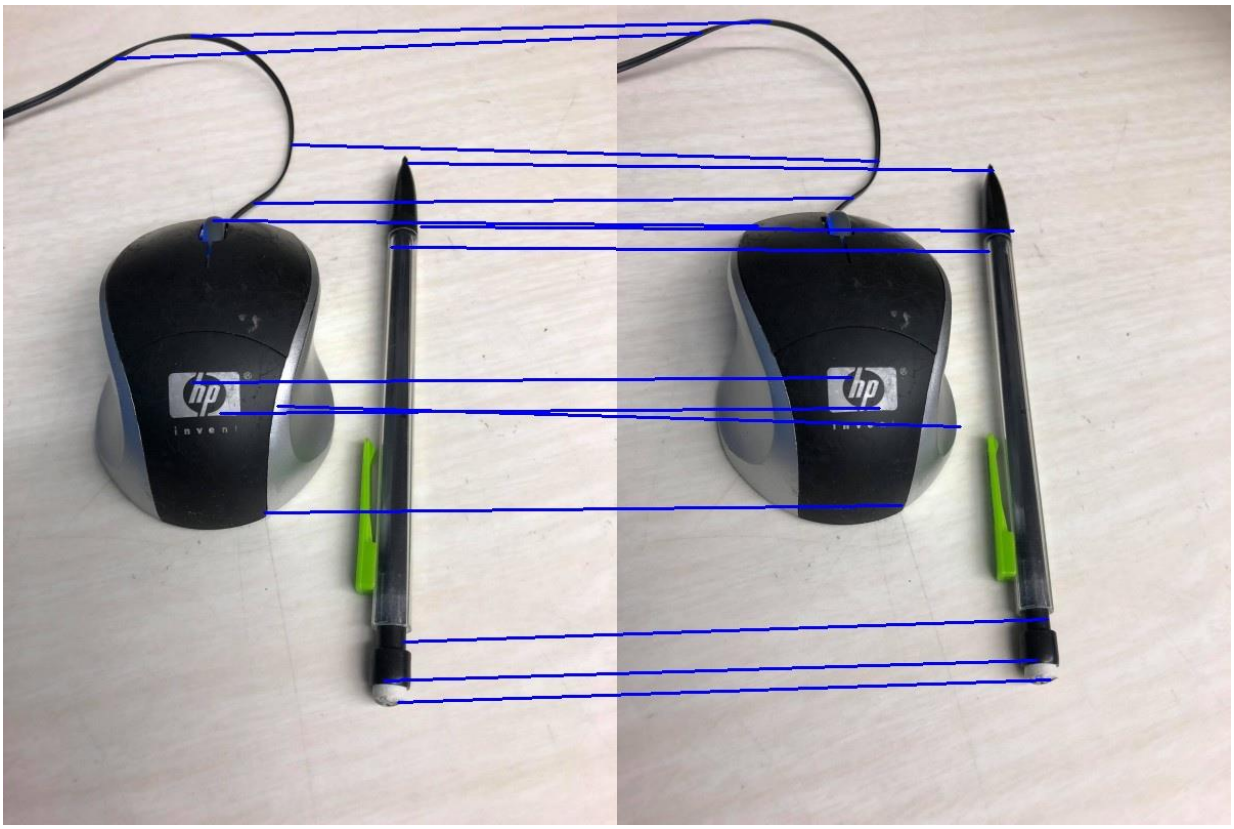


Harris detector with NCC, sigma = 1.5

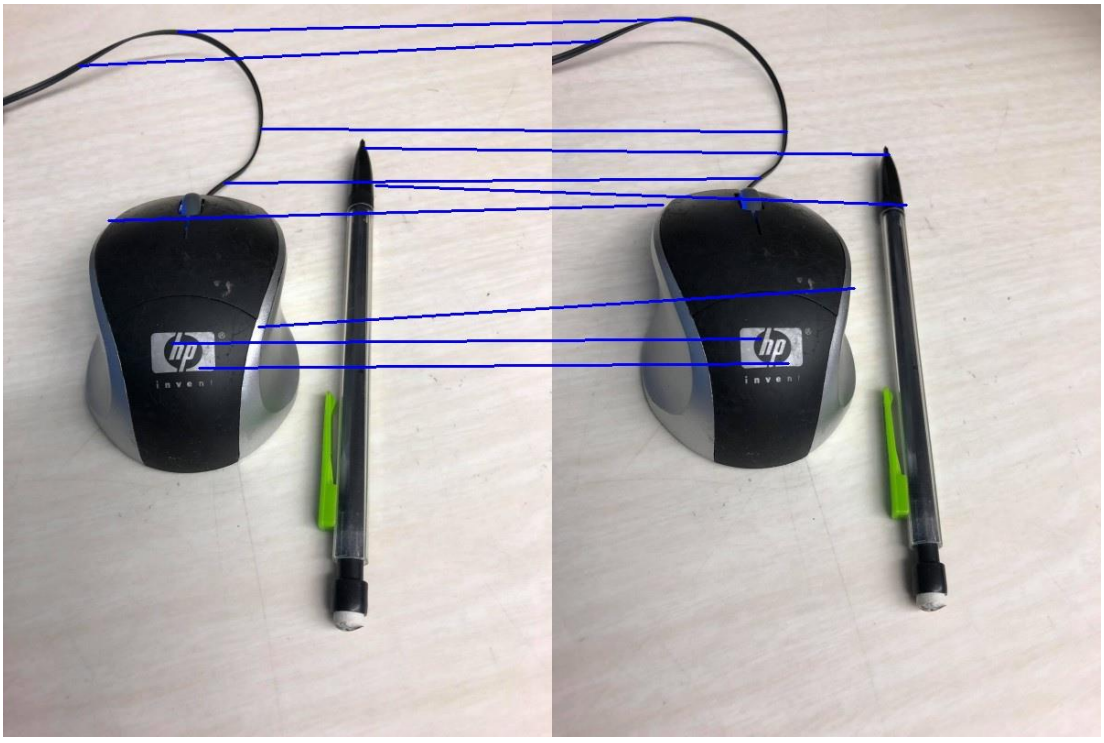


Harris detector with NCC, sigma = 1.8

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04



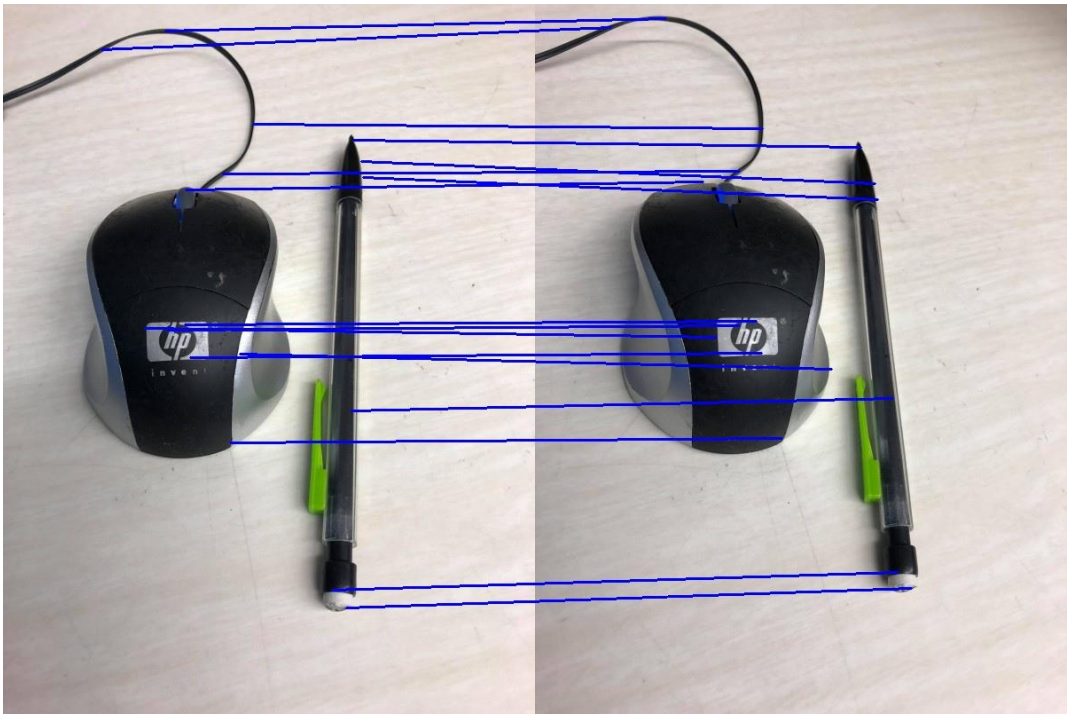
Harris detector with NCC, sigma = 2.1



Harris detector with SSD, sigma = 1.2

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

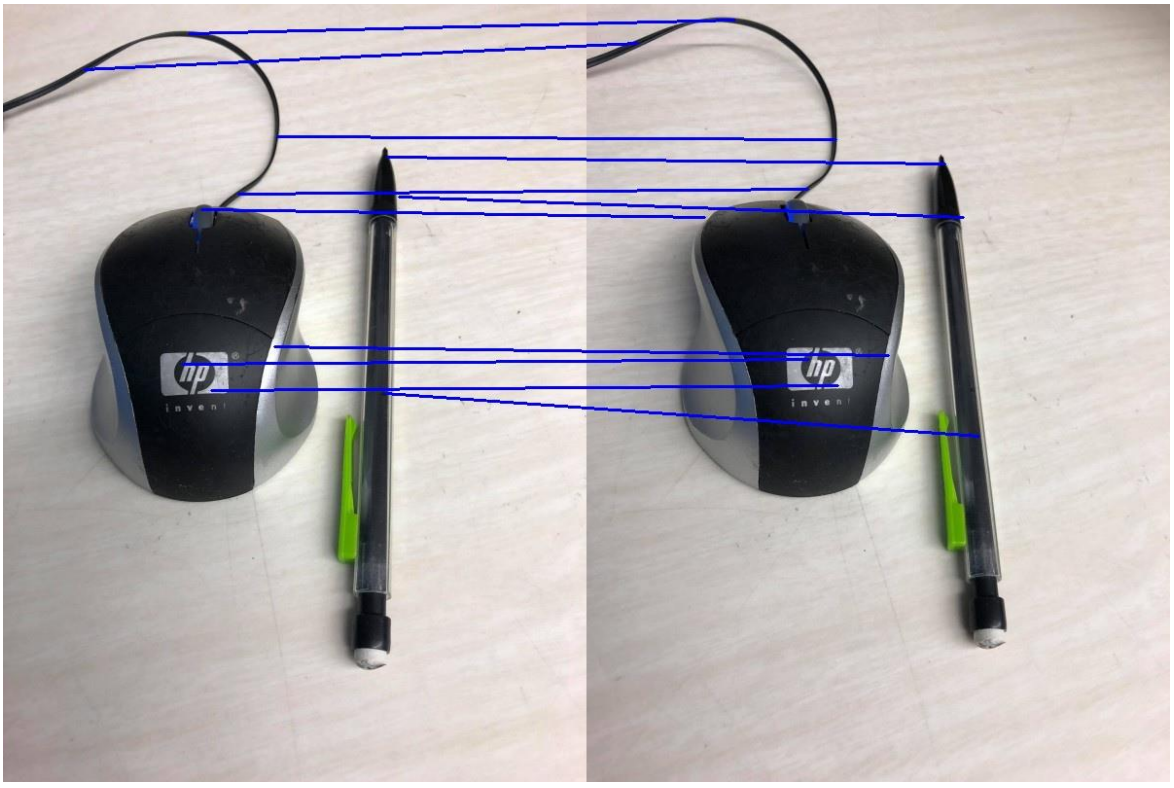
Harris detector with SSD, sigma = 1.5



Harris detector with SSD, sigma = 1.8

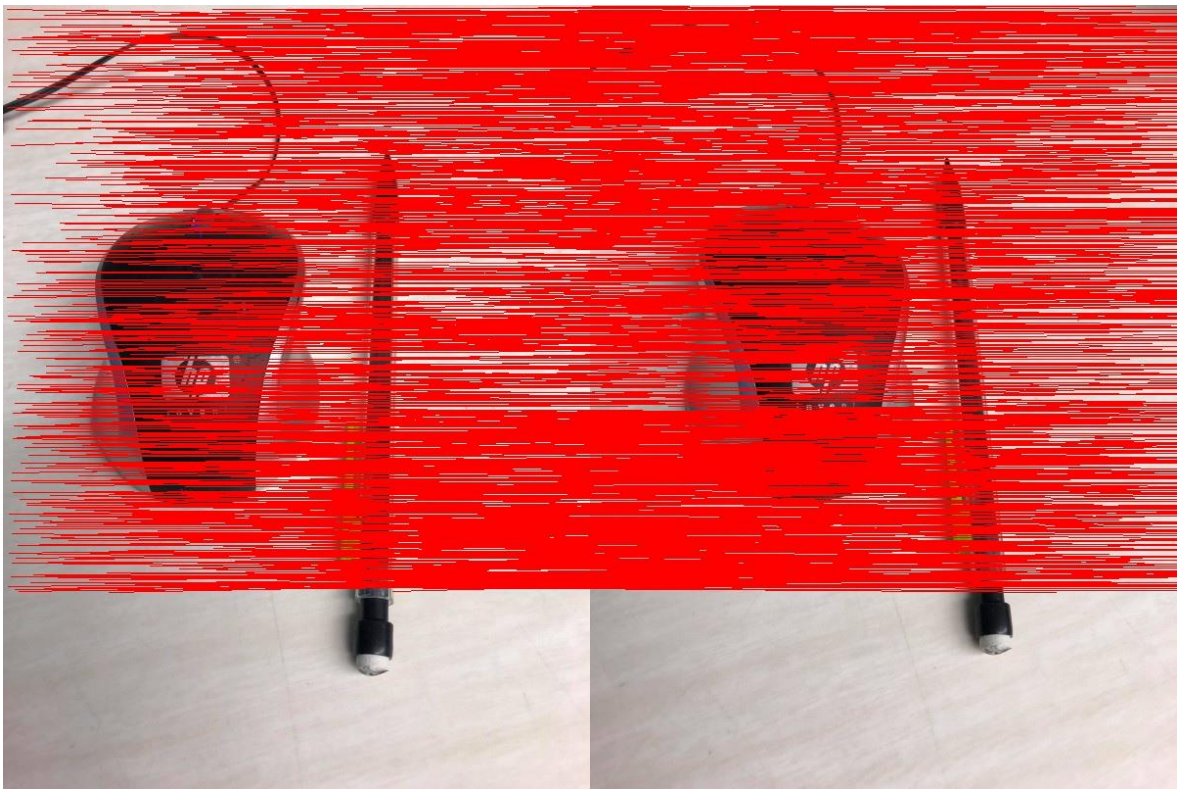
Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

Harris detector with SSD, sigma = 2.1



SIFT points matched using Euclidean distance

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04



Observation

Observation on interest points detected at various scales:

The lower the scale, the more interest points were detected with the Harris corner detector. As a result, a lower sigma gave us more correspondences.

SSD:

pair1:

sigma=1.2: correspondences: 170

sigma=1.5: correspondences: 157

sigma=1.8: correspondences: 154

sigma=2.1: correspondences: 108

pair2:

sigma=1.2: correspondences: 50

sigma=1.5: correspondences: 48

sigma=1.8: correspondences: 37

sigma=2.1: correspondences: 35

Rehana Mahfuz (rmahfuz@purdue.edu)
ECE 661 HW04

NCC:

pair1:

sigma=1.2: correspondences: 165

sigma=1.5: correspondences: 149

sigma=1.8: correspondences: 143

sigma=2.1: correspondences: 102

pair2:

sigma=1.2: correspondences: 44

sigma=1.5: correspondences: 41

sigma=1.8: correspondences: 28

sigma=2.1: correspondences: 30

Observation on relative performance of the two methods:

The Harris corner detector performs much better than SIFT. SIFT finds more corner points than Harris, and many of those corner points in the middle of the sky and in the middle of the ground. That can probably be corrected by adjusting the parameters, which I did not do using the readily available implementation from VLFeat.

It was also noticed that the Euclidean distance metric produced much more accurate correspondences for SIFT than NCC and SSD did for Harris corners.

Another point to note is that these methods fail when the camera captures the color of certain things as being different because of differences in lighting. For example, in the second picture in pair 2, the bottom left of the coffee packet has light shining on it, which is why it appears white. Some correspondences were found that map that non-white region from the first image to another non-white region in the second image, instead of to the corresponding white part, simply because the algorithm uses color variations to match corners.

Parameters chosen for feature extraction and matching:

For the Harris detector, I adjusted parameters to produce the desired results.

After calculating the C matrix for all pixels, I picked a window size of 29 for finding local maxima. When I thresholded the ratio values to be a minimum number, I used the mean of all the ratios as the threshold for pair 1. However, when that threshold seemed to not work for pair 2, I used a threshold of 0.01, as suggested in the lecture notes, and that worked well for pairs 2-4. Zero padding was used any time the window bounds exceeded the image bounds.

For filtering out the most accurate correspondences after calculating NCC and SSD, it was found that when I take the highest few NCCs and the lowest few SSDs, there were always inaccuracies. I resolved

this by eliminating correspondences where the absolute value of the difference between the two corner points along a certain coordinate was more than a certain threshold. Following is a table showing these thresholds which were estimated by looking at the deviations between the two scenes using GIMP, and further refined experimentally:

	Delta x	Delta y
Pair 1	80	35
Pair 2	210	50
Pair 3	40	90
Pair 4	35	80

Source code

Note: The entire implementation was done in Python except calculation of SIFT features, which was done in MATLAB. This is because I could not install the library that would do sift interest point detection in Python.

Harris.py

```
import numpy as np
import cv2, math, scipy.signal
import matplotlib.pyplot as plt


path = '../Users/rmahfuz/Desktop/661/HW04/'


def harris(img, sigma):
    #finding smallest even integer greater than 4*sigma (size of haar filter)
    size = int(4*sigma) + 1
    if size % 2 != 0:
        size += 1
    assert size % 2 == 0
    haar_x = np.hstack([-1*np.ones((size, int(size/2))), np.ones((size, int(size/2)))])
    haar_y = np.vstack([np.ones((int(size/2), size)), -1*np.ones((int(size/2), size))])
    img_x = scipy.signal.convolve2d(img, haar_x, mode = 'same')
    img_y = scipy.signal.convolve2d(img, haar_y, mode = 'same')
    img_x = np.divide(np.subtract(img_x, np.min(img_x)), np.subtract(np.max(img_x), np.min(img_x)))
    img_y = np.divide(np.subtract(img_y, np.min(img_y)), np.subtract(np.max(img_y), np.min(img_y)))
```

```
    print('img_x.shape = ', img_x.shape)
    d_x2 = np.square(img_x)
    d_y2 = np.square(img_y)
    d_xy = np.multiply(img_x, img_y)
    #print(img_x.shape)
    win_size = int(5*sigma)
    if win_size % 2 == 0: #ensuring that window size is even
        win_size += 1
    corner_pts = []
    ratio_store = np.zeros((img.shape[0], img.shape[1]))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            C = np.zeros((2,2))
            for k in range(i - int(win_size/2), i+int(win_size/2)+1):
                for l in range(j-int(win_size/2), j+int(win_size/2)+1):
                    if k < img.shape[0] and l < img.shape[1]:
                        C[0][0] += d_x2[k][l]
                        C[0][1] += d_xy[k][l]
                        C[1][0] += d_xy[k][l]
                        C[1][1] += d_y2[k][l]
            if np.linalg.matrix_rank(C) == 2:
                ratio_store[i][j] = np.linalg.det(C)/np.square(np.matrix.trace(C))
            else:
                ratio_store[i][j] = -1
    print('finished calculating Cs')
    fil_win = 29 #filter window size
    #thresh = 2*np.mean(ratio_store)
    thresh = 0.01
    #Filtering out non-local maxima
    for i in range(img.shape[0]):
```

```
        for j in range(img.shape[1]):
            x_min = max(0, i-int(fil_win/2))
            x_max = min(i+int(fil_win/2)+1, img.shape[0])
            y_min = max(0,j-int(fil_win/2))
            y_max = min(j+int(fil_win/2)+1, img.shape[1])
            max_ratio = np.max(ratio_store[x_min:x_max,y_min:y_max])
            # if local maximum and greater than threshold
            if ratio_store[i,j] == max_ratio and ratio_store[i,j] > thresh:
                pad_size = int(fil_win/2)
                if i >= pad_size and i < img.shape[0] - pad_size and j >= pad_size and j < img.shape[1] -
pad_size:
                    corner_pts.append((i,j))
    print('finished calculating corner points')
    return corner_pts


def ssd(cor_pts1, cor_pts2, img1, img2): #thresh: try 5000
    '''returns a list of point correspondences, where every element is of the form [(pt_x1, pt_y1), (pt_x2,
pt_y2)]'''
    fil_win = 31 #filter window length
    pad_len = int(fil_win/2)
    ssd_store = np.zeros((len(cor_pts1), len(cor_pts2))) #array to store all SSDs
    ssd_store = ssd_store - 1
    for i in range(len(cor_pts1)):
        for j in range(len(cor_pts2)):
            cor1 = cor_pts1[i]; cor2 = cor_pts2[j]


            x_lo1 = max(0, cor1[0]-pad_len)
            x_hi1 = min(cor1[0]+pad_len+1, img1.shape[0])
            y_lo1 = max(0,cor1[1]-pad_len)
            y_hi1 = min(cor1[1]+pad_len+1, img1.shape[1])
```

```
        x_lo2 = max(0, cor2[0]-pad_len)
        x_hi2 = min(cor2[0]+pad_len+1, img2.shape[0])
        y_lo2 = max(0,cor2[1]-pad_len)
        y_hi2 = min(cor2[1]+pad_len+1, img2.shape[1])

        if x_hi1 - x_lo1 == x_hi2 - x_lo2 and y_hi1 - y_lo1 == y_hi2 - y_lo2:
            ssd_store[i,j] = np.sum(np.square(np.subtract(img1[x_lo1:x_hi1,y_lo1:y_hi1], img2[x_lo2:x_hi2,y_lo2:y_hi2])))
        else:
            ssd_store[i,j] = -1


    to_ret = [];ssds = []; track = np.ones(len(cor_pts2))
    thresh = np.min(ssd_store[ssd_store > 0])
    for i in range(len(ssd_store)):
        cur = ssd_store[i]
        to_find = cur[cur>0]
        if len(to_find) > 0:
            j = np.argmin(to_find)
            #if min(to_find) < 2000000 and abs(cor_pts1[i][0] - cor_pts2[j][0]) < 20 and abs(cor_pts1[i][1] - cor_pts2[j][1]) < 35:
            if abs(cor_pts1[i][0] - cor_pts2[j][0]) < 35 and abs(cor_pts1[i][1] - cor_pts2[j][1]) < 80 and track[j] == 1:
                ssds.append(min(to_find))
                to_ret.append([cor_pts1[i], cor_pts2[j]])
                track[j] = 0
    return (to_ret, ssds)

#-----------------------------------------------------------------------------------------------------------------------

def ncc(cor_pts1, cor_pts2, img1, img2):
```

```
    #returns a list of point correspondences, where every element is of the form [(pt_x1, pt_y1), (pt_x2, pt_y2)]
    fil_win = 31 #filter window length
    pad_len = int(fil_win/2)
    ncc_store = np.zeros((len(cor_pts1), len(cor_pts2))) #array to store all NCCs
    ncc_store = ncc_store - 2
    for i in range(len(cor_pts1)):
        for j in range(len(cor_pts2)):
            cor1 = cor_pts1[i]; cor2 = cor_pts2[j]

            x_lo1 = max(0, cor1[0]-pad_len)
            x_hi1 = min(cor1[0]+pad_len+1, img1.shape[0])
            y_lo1 = max(0,cor1[1]-pad_len)
            y_hi1 = min(cor1[1]+pad_len+1, img1.shape[1])

            x_lo2 = max(0, cor2[0]-pad_len)
            x_hi2 = min(cor2[0]+pad_len+1, img2.shape[0])
            y_lo2 = max(0,cor2[1]-pad_len)
            y_hi2 = min(cor2[1]+pad_len+1, img2.shape[1])

            if x_hi1 - x_lo1 == x_hi2 - x_lo2 and y_hi1 - y_lo1 == y_hi2 - y_lo2:
                mean1 = np.mean(img1[x_lo1:x_hi1,y_lo1:y_hi1])
                mean2 = np.mean(img2[x_lo2:x_hi2,y_lo2:y_hi2])
                term1 = np.subtract(img1[x_lo1:x_hi1,y_lo1:y_hi1], mean1)
                term2 = np.subtract(img2[x_lo2:x_hi2,y_lo2:y_hi2], mean2)
                ncc_store[i,j] = np.divide(np.sum(np.multiply(term1, term2)),
                                           np.sqrt(np.multiply(np.sum(np.square(term1)), np.sum(np.square(term2)))))
                #np.sum(np.square(np.subtract(img1[x_lo1:x_hi1,y_lo1:y_hi1],
img2[x_lo2:x_hi2,y_lo2:y_hi2])))
```

```
    to_ret = [];nccs = []
    #thresh = np.min(ssd_store[ncc_store > 0])
    track = np.ones(len(cor_pts2))
    for i in range(len(ncc_store)):
        cur = ncc_store[i]
        to_find = cur[cur>=-1]
        if len(to_find) > 0:
            j = np.argmax(to_find)
            if abs(cor_pts1[i][0] - cor_pts2[j][0]) < 35 and abs(cor_pts1[i][1] - cor_pts2[j][1]) < 80 and
track[j] == 1:# and max(to_find) > 0.45:
                nccs.append(max(to_find))
                to_ret.append([cor_pts1[i], cor_pts2[j]])
                track[j] = 0

    return (to_ret, nccs)
#-----------------------------------------------------------------------------------------------------------------------
-
def make_gray(img):
    new_img = []
    for i in range(img.shape[0]): #Making the image grayscale
        new_img.append(list(map(lambda x: sum(x)/3, img[i])))
    img = np.array(new_img)
    return img
#-----------------------------------------------------------------------------------------------------------------------
-
def use_harris(pair, sigma):
    color_img1 = cv2.imread(path + 'pictures/pair' + str(pair) + '/1.jpg')
    color_img2 = cv2.imread(path + 'pictures/pair' + str(pair) + '/2.jpg')

    img1 = make_gray(color_img1)
    img2 = make_gray(color_img2)
```

```
'''img1_corners = harris(img1, sigma = sigma)
img2_corners = harris(img2, sigma = sigma)

# To write the corners points in files:

print('number of corner points of image 1 are: ', len(img1_corners))
with open(path + 'pictures/pair' + str(pair) + '/points1.txt', 'w') as fh:
    for i in range(len(img1_corners)):
        fh.write(str(img1_corners[i][0]) + ', ' + str(img1_corners[i][1]) + '\n' )

#img2_corners = harris(img2, sigma = sigma)
print('number of corner points of image 2 are: ', len(img2_corners))

with open(path + 'pictures/pair' + str(pair) + '/points2.txt', 'w') as fh:
    for i in range(len(img2_corners)):
        fh.write(str(img2_corners[i][0]) + ', ' + str(img2_corners[i][1]) + '\n')'''

# To read corner points from file
img1_corners = []; img2_corners = []

with open(path + 'pictures/pair' + str(pair) + '/points1.txt', 'r') as fh:
    lines = fh.readlines()
    for line in lines:
        img1_corners.append(list(map(lambda x: int(x.strip()), line.split(','))))
print('len(img1_corners) = ', len(img1_corners))

with open(path + 'pictures/pair' + str(pair) + '/points2.txt', 'r') as fh:
    lines = fh.readlines()
    for line in lines:
```

```
            img2_corners.append(list(map(lambda x: int(x.strip()), line.split(','))))
    print('len(img2_corners) = ', len(img2_corners))


    '''#Just indicating corners:
    color_img1 = cv2.imread(path + 'pictures/pair' + str(pair) + '/1.jpg')


    for corner in img1_corners:
        for i in range(corner[0] - 2, corner[0] + 3):
            for j in range(corner[1] - 2, corner[1] + 3):
                if i < img1.shape[0] and j < img1.shape[1]:
                    color_img1[i,j] = (0,0,255)
    cv2.imwrite(path + 'pictures/pair' + str(pair) + '/corners.jpg', color_img1)'''



    corresp, ssds = ncc(img1_corners, img2_corners, img1, img2) #correspondences. either ncc or ssd
    print('found correspondences of size ', len(corresp))



    stacked_img = np.hstack((color_img1, color_img2))
    i = 0
    for pt in corresp:
        cv2.line(stacked_img, (pt[0][1], pt[0][0]), (pt[1][1] + img1.shape[1], pt[1][0]), color = (255,0,0),
thickness = 2)
        #cv2.line(stacked_img, (pt[0][1], pt[0][0]), (pt[1][1] + img1.shape[1], pt[1][0]), color =
(0,0,ssds[i]*max(ssds)/255), thickness = 1)
        #cv2.putText(img = stacked_img, text = str(i), org = (pt[0][1], pt[0][0]), fontFace =
cv2.FONT_HERSHEY_SIMPLEX, fontScale = 0.5, color = (0,0,ssds[i]*max(ssds)/255), thickness = 1)
        #cv2.putText(img = stacked_img, text = str(i), org = (pt[1][1]+img1.shape[1], pt[1][0]), fontFace =
cv2.FONT_HERSHEY_SIMPLEX, fontScale = 0.5, color = (0,0,ssds[i]*max(ssds)/255), thickness = 1)
        i+=1
    cv2.imwrite(path + 'pictures/pair' + str(pair) + '/nccsigma'+str(sigma)+'.jpg', stacked_img)
```

```
    '''for corner in img1_corners:
        for i in range(corner[0] - 2, corner[0] + 3):
            for j in range(corner[1] - 2, corner[1] + 3):
                if i < img1.shape[0] and j < img1.shape[1]:
                    color_img1[i,j] = (0,0,255)
    cv2.imwrite(path + 'pictures/pair1/corners.jpg', color_img1)'''
    #plotting image1
    '''plt.imshow(img1)
    for i in range(len(corresp)):
        x = corresp[i][0][0]; y = corresp[i][0][1]
        plt.scatter(x, y, s = 11, c = 'r', marker = 'x')
        plt.text(x+0.3, y+0.3, str(i), fontsize=15)
    plt.savefig(path + 'pictures/pair' + str(pair) + '/corner1.png')

    plt.clf()

    #plotting image2
    plt.imshow(img2)
    for i in range(len(corresp)):
        x = corresp[i][1][0]; y = corresp[i][1][1]
        plt.scatter(x, y, s = 11, c = 'r', marker = 'x')
        plt.text(x+0.3, y+0.3, str(i), fontsize=15)
    plt.savefig(path + 'pictures/pair' + str(pair) + '/corner2.png')'''

if __name__ == '__main__':
    use_harris(pair = 4, sigma = 1.2)
```

sift.m

```
%run vlfeat-0.9.20/toolbox/vlsetup
```

```
img1 = double(rgb2gray(imread('pictures/pair4/1.jpg')));
img2 = double(rgb2gray(imread('pictures/pair4/2.jpg')));
%vl_version verbose
[f1,d1] = vl_sift(single(img1));
[f2,d2] = vl_sift(single(img2)) ;

%Writing to file:
fh = fopen('pictures/pair4/sift1.txt','w');
for i = 1:size(f1,2)
   fprintf(fh, '%d, %d, %d, %d, ', f1(1,i), f1(2,i), f1(3,i), f1(4,i));
   %fprintf(fh, '\n');
end
fclose(fh);

fh = fopen('pictures/pair4/sift2.txt','w');
for i = 1:size(f2,2)
   fprintf(fh, '%d, %d, %d, %d, ', f2(1,i), f2(2,i), f2(3,i), f2(4,i));
   %fprintf(fh, '\n');
end
fclose(fh);
```

sift.py

```
import numpy as np
import cv2

path = '../Users/rmahfuz/Desktop/661/HW04/'

def euclid(x,y):
   assert x.shape == y.shape
   return np.sqrt(np.sum(np.square(np.subtract(x,y))))
```

```
def sift(pair):

    color_img1 = cv2.imread(path + 'pictures/pair' + str(pair) + '/1.jpg')
    color_img2 = cv2.imread(path + 'pictures/pair' + str(pair) + '/2.jpg')
    print(color_img1.shape)

    with open(path + 'pictures/pair' + str(pair) + '//sift1.txt', 'r') as fh:
        txt = fh.read()
        old_pts1 = np.array(list(map(lambda x: float(x.strip()), txt.split(',')[:-1])))
        pts1 = old_pts1.reshape((int(len(old_pts1)/4), 4))

        with open(path + 'pictures/pair' + str(pair) + '//sift2.txt', 'r') as fh:
            txt = fh.read()
        old_pts2 = np.array(list(map(lambda x: float(x.strip()), txt.split(',')[:-1])))
        pts2 = old_pts2.reshape((int(len(old_pts2)/4), 4))

        #print(pts1.shape, pts2.shape)
        #assert pts1.shape == pts2.shape

        corresp = []
        for pt in pts1:
            min_dist = euclid(pt,pts2[0]); cor = pts2[0]
            for i in range(1,len(pts2)):
                dist = euclid(pt,pts2[i])
                if dist < min_dist:
                    min_dist = dist
                    cor = pts2[i]
            corresp.append([[int(pt[0]), int(pt[1])], [int(cor[0]), int(cor[1])]])
```

```
    stacked_img = np.hstack((color_img1, color_img2))

    i = 0

    for pt in corresp:

        cv2.line(stacked_img, (pt[0][1], pt[0][0]), (pt[1][1] + color_img1.shape[1], pt[1][0]), color =
(0,0,255), thickness = 1)

        #cv2.line(stacked_img, (pt[0][1], pt[0][0]), (pt[1][1] + img1.shape[1], pt[1][0]), color =
(0,0,ssds[i]*max(ssds)/255), thickness = 1)

        #cv2.putText(img = stacked_img, text = str(i), org = (pt[0][1], pt[0][0]), fontFace =
cv2.FONT_HERSHEY_SIMPLEX, fontScale = 0.5, color = (0,0,255), thickness = 1)

        #cv2.putText(img = stacked_img, text = str(i), org = (pt[1][1]+color_img1.shape[1], pt[1][0]),
fontFace = cv2.FONT_HERSHEY_SIMPLEX, fontScale = 0.5, color = (0,0,255), thickness = 1)

        i+=1

    cv2.imwrite(path + 'pictures/pair' + str(pair) + '/sift.jpg', stacked_img)


if __name__ == '__main__':

  sift(pair = 4)
```