ECE 661 – Homework 4

Ran Xu

xu943@purdue.edu

09/27/2018

1.1 Harris Corner Detector

I first determine a list of scales to use. The base scale is $\sigma = 1.2$ for a 9-elemnt operator in SURF algorithm. The size of operator increases to 15x15, 21x21 and then 27x27. Then the scale for each operator are 2.0, 2.8 and 3.6. In Harris Corner Detector

I then use Haar filter in even shape to compute the derivative along x and y axis. The shape is $\lceil 4\sigma \rceil$.

I thirdly use calculate the local cumulative matrix C. The size of the neighborhood is $\lceil 5\sigma \rceil$.

I then calculate the ratio of the two eigenvalues and set 1/4.5 for this metric – DET over the square of the trace.

I finally perform the elimination in a 7x7 neighborhood to select local maximum to reduce the number of detected corners.

1.2 SIFT

In SIFT, we first found the local extrema in the 3x3x3 neighborhood in DoG. Then we calculate the sub-pixel level extrema using derivative vector and Hessian matrix. Thirdly, we reject the much too small extrema. Finally, we construct the 128-element descriptor for each interest point we detected.

1.3 NCC, SSD, and Feature matching algorithm

Given a pair of feature vector $f_1(i)$ and $f_2(i)$ extracted from image 1 and 2 respectively, the Normalized cross-correlation (NCC) is defined as

$$NCC = \frac{\sum(f_1(i) - m_1)(f_2(i) - m_2)}{\sqrt{\sum(f_1(i) - m_1)^2 \cdot \sum(f_2(i) - m_2)^2}}$$

The Sum of Squared Differences (SSD) is defined as

$$SSD = \sum \left(f_1(i) - f_2(i)\right)^2$$

To get the correspondence between two images, I first extract feature vectors from the two images using either SIFT or Harris corner detector. Then I use a greedy algorithm that for each feature vector in the first image, search for the matched feature vector in the second image that minimize the SSD metric or maximize the NCC metric. To eliminate the much too correspondence, I selected 100 minimum SSD and maximum NCC as correspondence between the two images.

2.1 Observations for Harris Corner Detector

Interest points detected in each scales show different features in the image. Generally speaking interest points in smaller scale tend to show very detailed corners like walls and trees. However, interest points in larger scale can show large corners like the cloud.

2.2 Observations comparing Harris Corner Detector and SIFT

Harris Corner Detector is not so good as SIFT. Firstly, by looking at the correspondence built based on the feature vector. We can almost get horizontal lines of correspondence using SIFT features. However, we got very messy correspondence due to the poor descriptor in Harris Corner Detector, a.k.a. the gray values in the neighborhood and so on. Secondly, the execution time of SIFT is relatively lower than Harris Cornet Detector. This could also motivate more usage cases based on SIFT detector.

2.3 Parameters used

The parameters are summarized in the following Table.

Section of results	Input images	Interest point detector	Correspondence metrics	Parameters
3.1	Building and truck	Harris Corner Detector	NCC, SSD	$\sigma \in \{1.2, 2.0, 2.8, 3.6\}$ Non-max suppression uses 7x7 kernel. Choose the 100 most correspondent pairs
3.2	Building and truck	SIFT	NCC, SSD	Choose the 100 most correspondent pairs
3.3	Fountain and tower	Harris Corner Detector	NCC, SSD	$\sigma \in \{1.2, 2.0, 2.8, 3.6\}$ Non-max suppression uses 7x7 kernel. Choose the 100 most correspondent pairs
3.4	Fountain and tower	SIFT	NCC, SSD	Choose the 100 most correspondent pairs

3.1 Harris Corner Detector on Given Images

Two pairs of input images are shown as follows,

The detected corners on output images and the correspondences in four scales ([1.2, 2.0, 2.8, 3.6]) are shown as follows, I use NCC (for building image) and SSD (for truck image) metric to establish the correspondences.

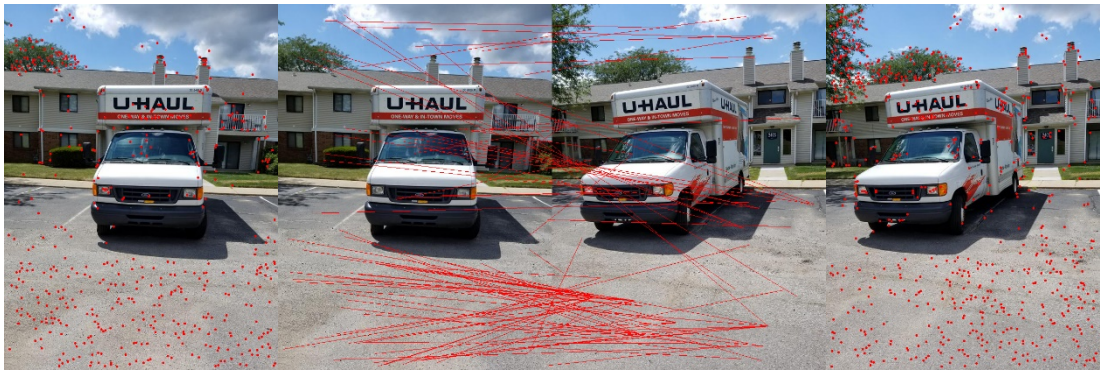
$\sigma = 1.2$, building image

$\sigma = 2.0$, building image

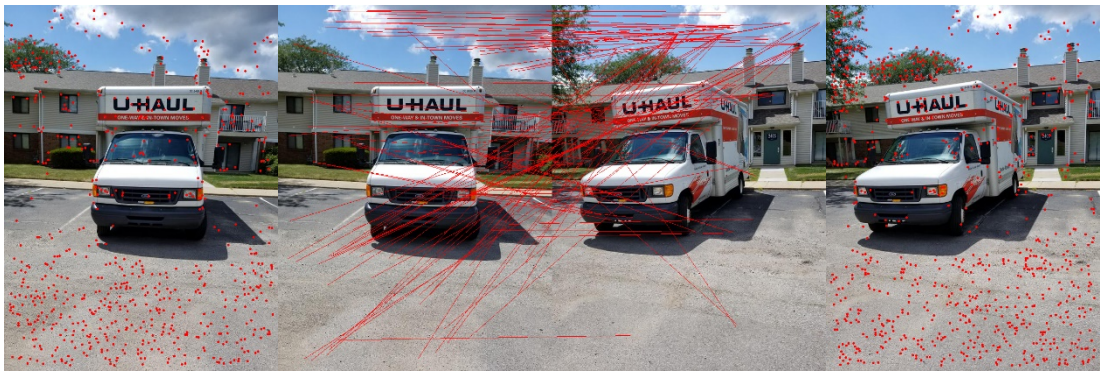
$\sigma = 2.8$, building image

$\sigma = 3.6$, building image

$\sigma = 1.2$, truck image



$\sigma = 2.0$, truck image



$\sigma = 2.8$, truck image

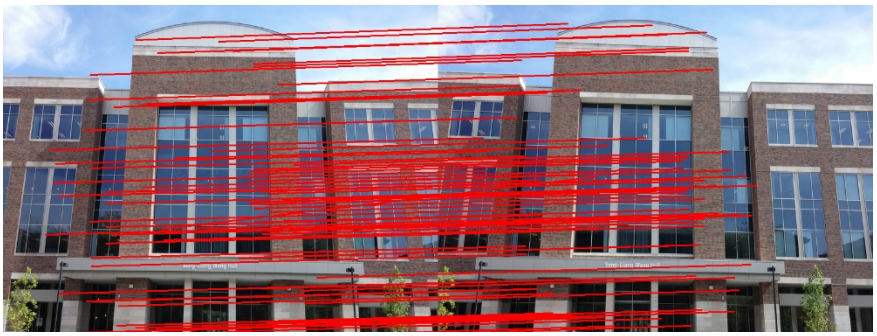
$\sigma = 3.6$, truck image

3.2 SIFT on Given Images

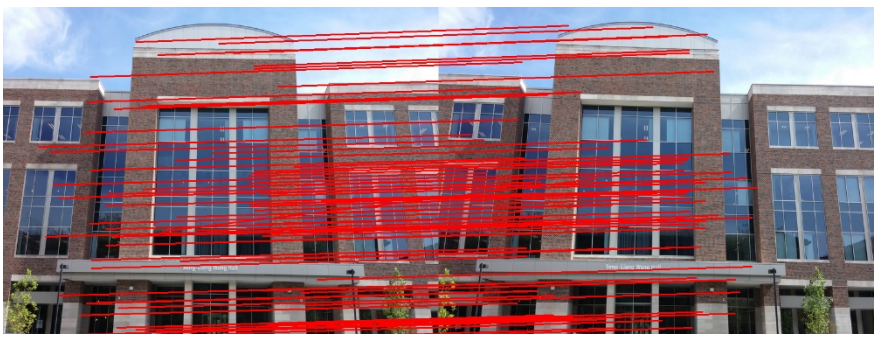
Input and Output images of SIFT on first image of building pair are shown as follows,

Input and Output images of SIFT on second image of building pair are shown as follows,

The correspondence built on NCC metric is shown as follows,

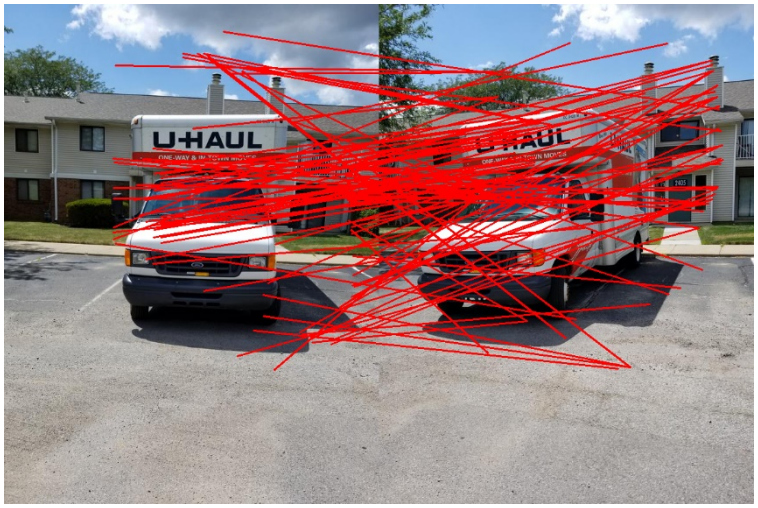


The correspondence built on SSD metric is shown as follows,

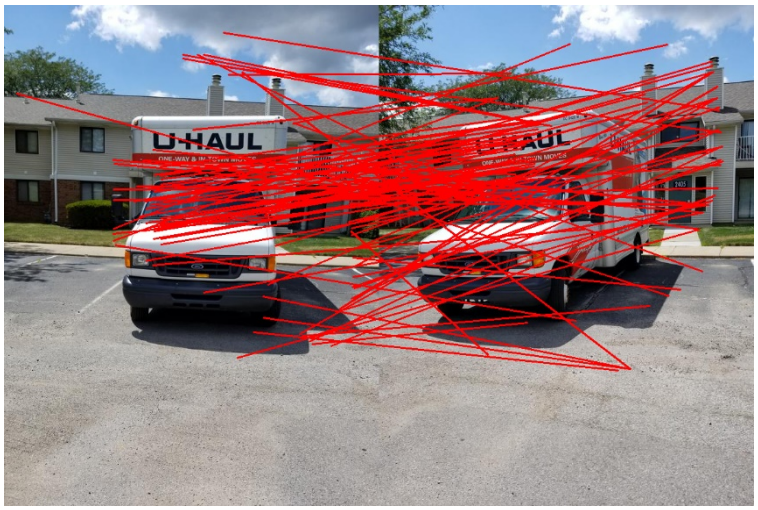


Input (2 on the left) and output (2 on the right) images of SIFT on truck pair are shown as follows,

The correspondence built on NCC metric is shown as follows,



The correspondence built on SSD metric is shown as follows,

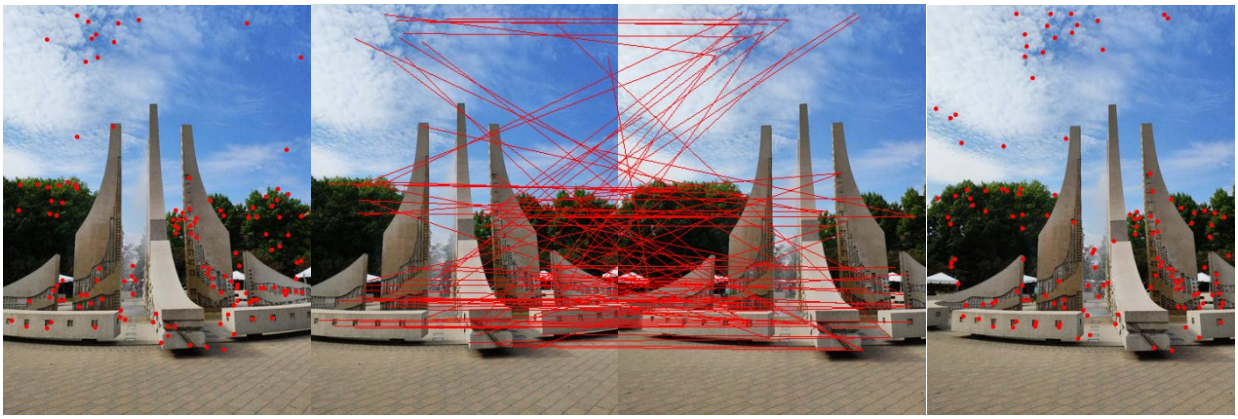


3.3 Harris Corner Detector on Own Images

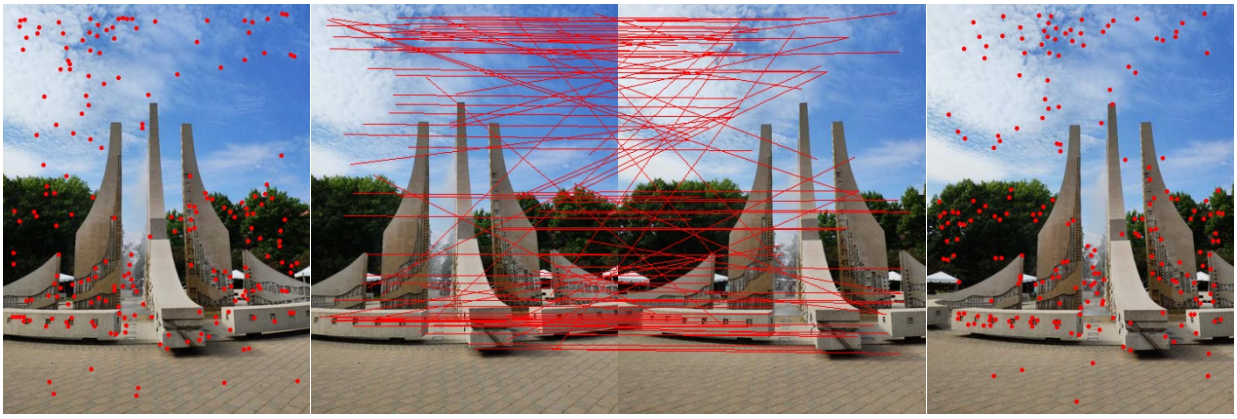
Two pairs of input images are shown as follows,

The detected corners on output images and the correspondences in four scales ([1.2, 2.0, 2.8, 3.6]) are shown as follows, I use NCC metric to establish the correspondences.

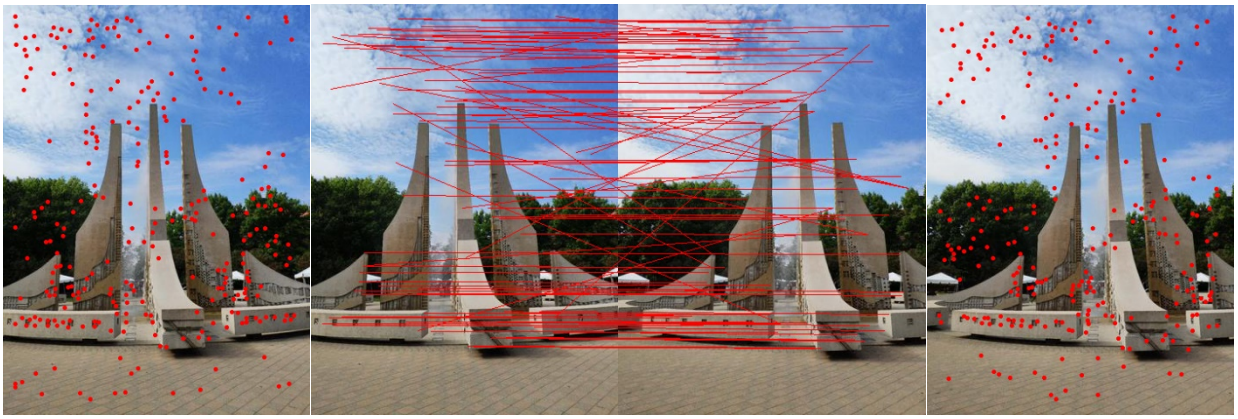
$\sigma = 1.2$, fountain image



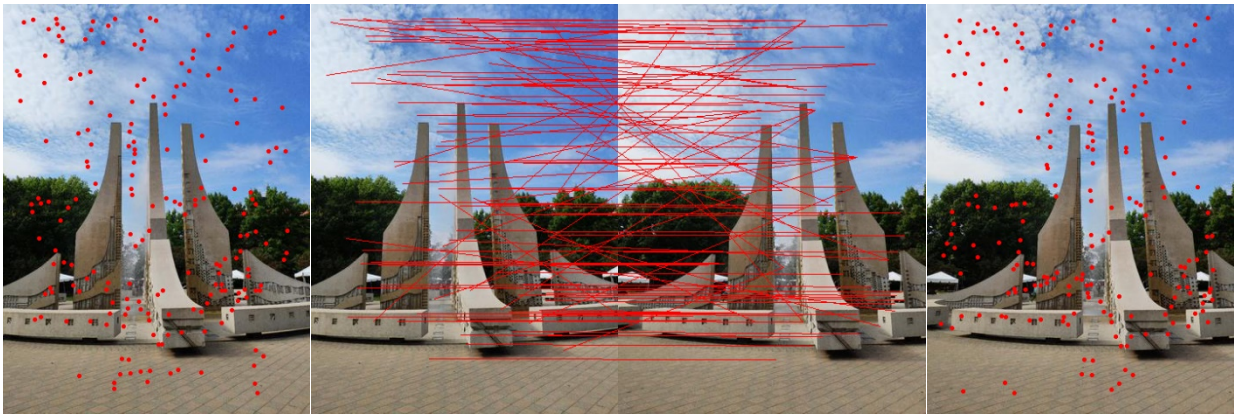
$\sigma = 2.0$, fountain image



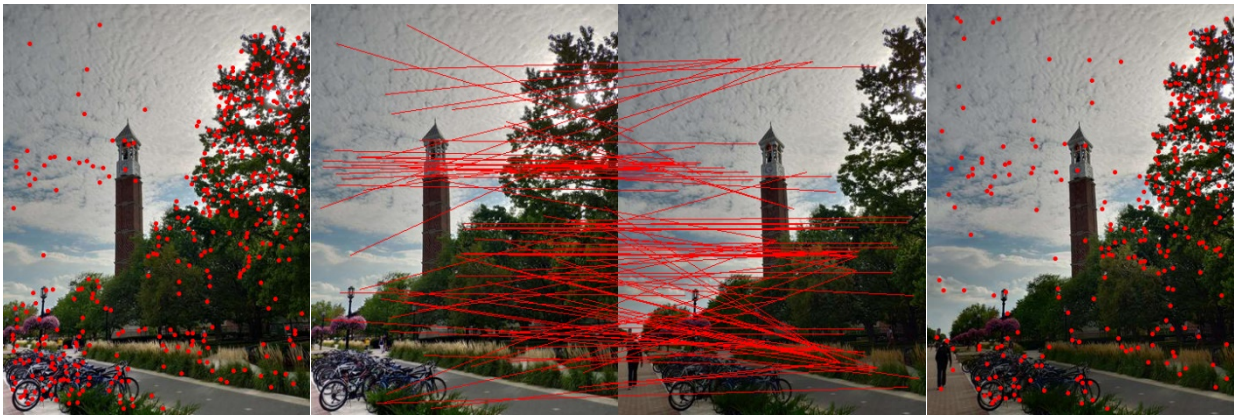
$\sigma = 2.8$, fountain image



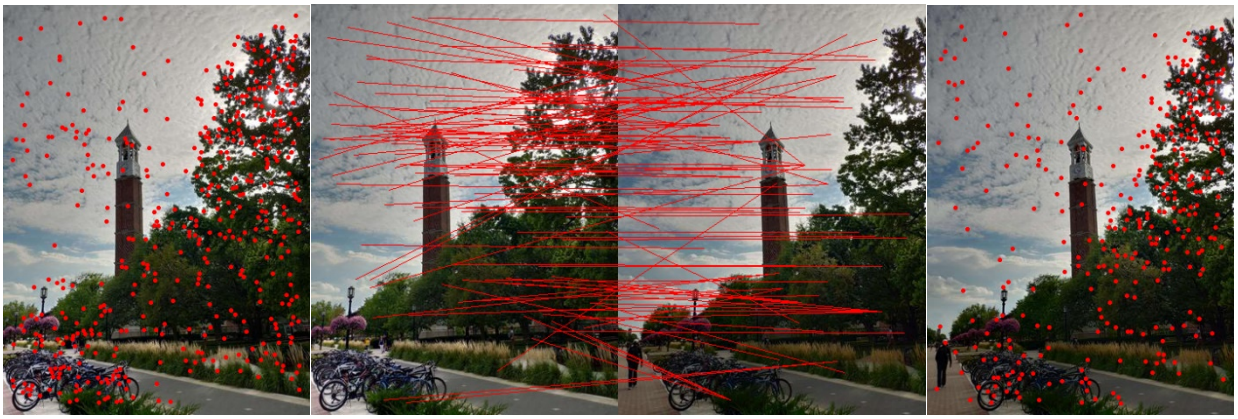
$\sigma = 3.6$, fountain image



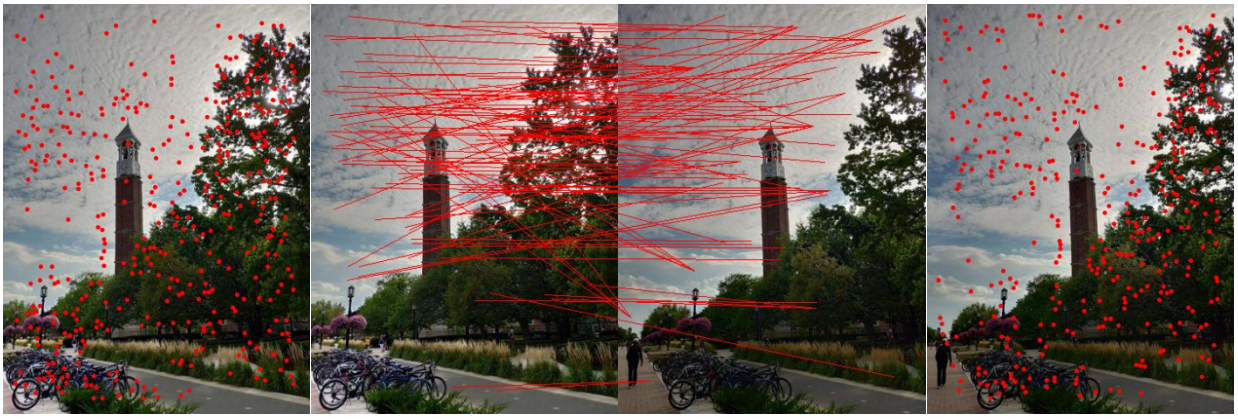
$\sigma = 1.2$, tower image



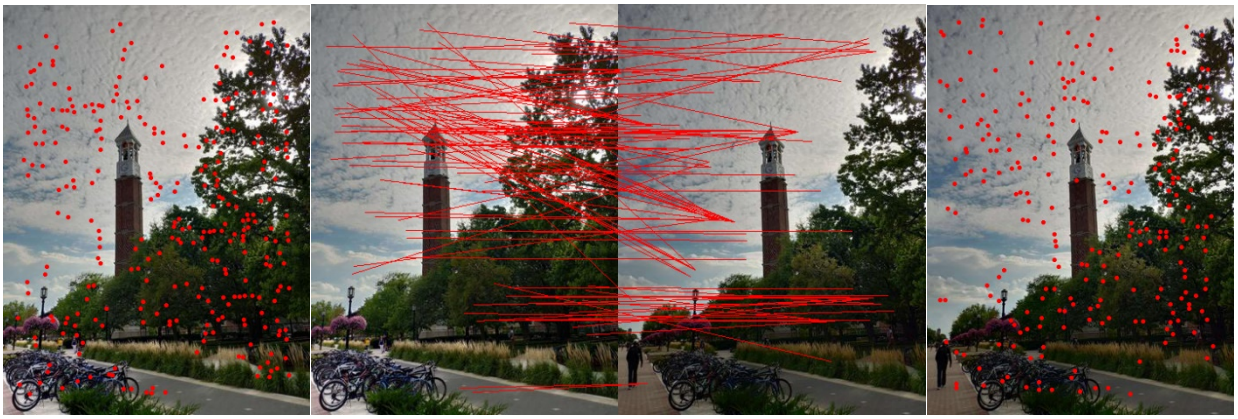
$\sigma = 2.0$, tower image



$\sigma = 2.8$, tower image



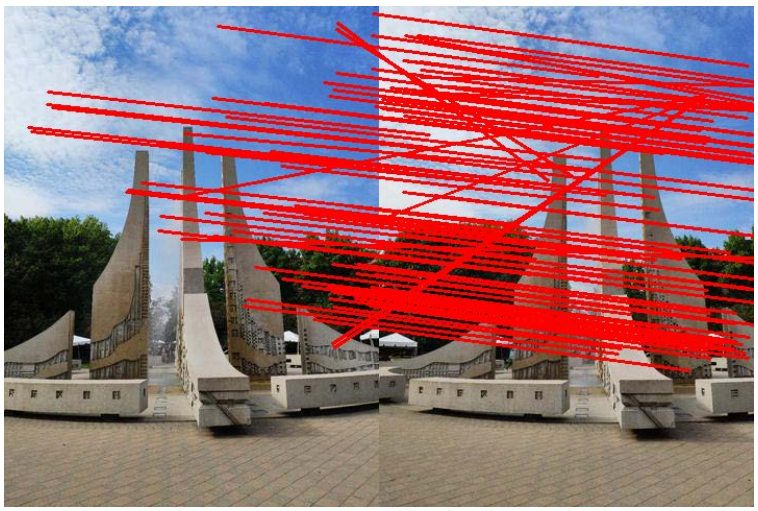
$\sigma = 3.6$, tower image



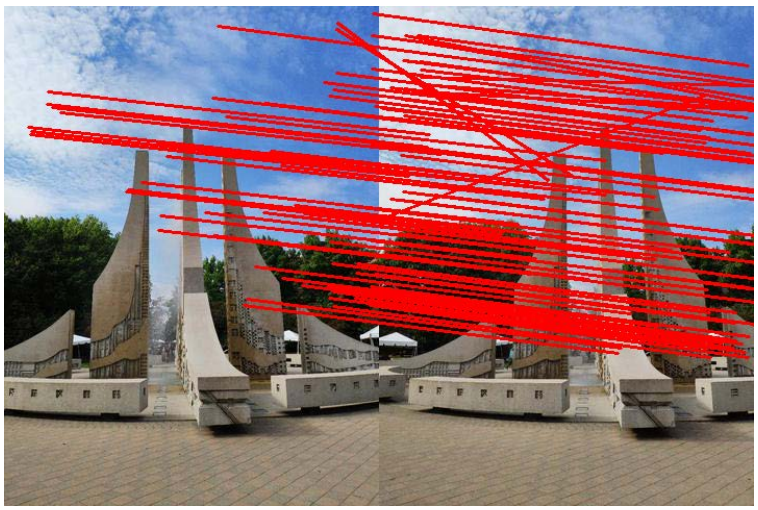
3.2 SIFT on Own Images

Input (2 on the left) and output (2 on the right) images of SIFT on truck pair are shown as follows,

The correspondence built on NCC metric is shown as follows,

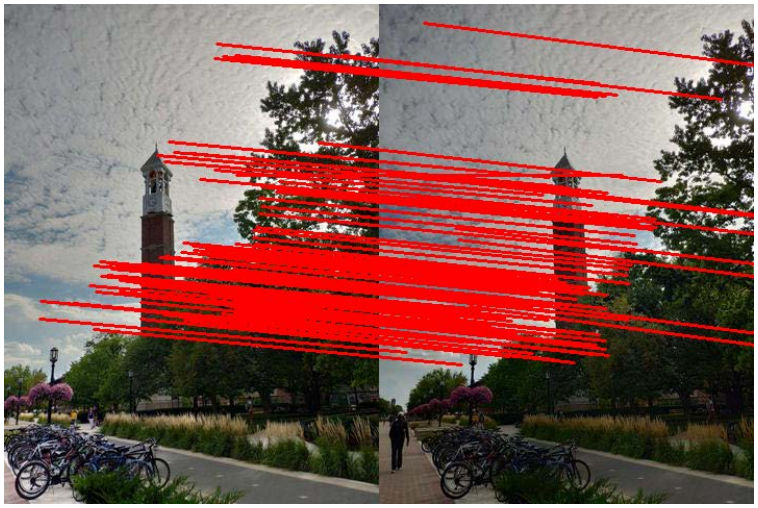


The correspondence built on SSD metric is shown as follows,

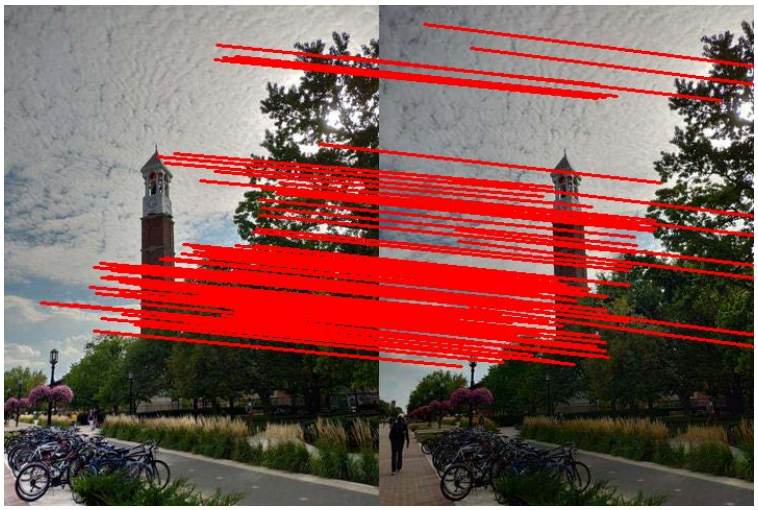


Input (2 on the left) and output (2 on the right) images of SIFT on truck pair are shown as follows,

The correspondence built on NCC metric is shown as follows,



The correspondence built on SSD metric is shown as follows,



4.1 Source code of Harris Corner Detector, NCC, SSD, and Feature matching algorithm

```
import numpy as np
from PIL import Image, ImageDraw
import time
import sys
import matplotlib.pyplot as plt
def Haar(scale = 1.2):  # scale in {1.2,2,2.8,3.6}
    ScaleToSize = {1.2:6, 2:8, 2.8:12, 3.6:16}
    HaarShape = ScaleToSize[scale]
    Haar_x = np.ones((HaarShape, HaarShape)) #[-1,1]
    Haar_y = np.ones((HaarShape, HaarShape)) # [1;-1]
    Haar_x[:, 0:int(HaarShape/2)] = -1
    Haar_y[int(HaarShape/2):HaarShape, :] = -1
    return (Haar_x, Haar_y, HaarShape)

def HCD_Kernel(img, scale = 1.2, NMS_Size = 7): # [1.2, 2, 2.8, 3.6]
    # HarrisDC, Get the Harris corner in img of scale
    (Haar_x, Haar_y, HaarShape) = Haar(scale)
    ScaleToNeighborSize = {1.2:7, 2:11, 2.8:15, 3.6:19}
    NeighborSize = ScaleToNeighborSize[scale]
    NMS_Side = int(NMS_Size/2)
    FinalSizeRed = HaarShape+NeighborSize+NMS_Size-3
    FinalShape = [img.shape[0]-FinalSizeRed, img.shape[1]-FinalSizeRed]

    # Step 0: Get final coordinate vector, vec_x, vec_y
    vec_x = np.zeros((FinalShape[0]*FinalShape[1],))
    vec_y = np.zeros((FinalShape[0]*FinalShape[1],))
    for idx in range(FinalShape[0]):
        for idy in range(FinalShape[1]):
            vec_x[idx*FinalShape[1]+idy] = idx
            vec_y[idx*FinalShape[1]+idy] = idy

    # step 1: perform convolution, get dx,dy,vec_dx,vec_dy
    dx = np.zeros((img.shape[0] - HaarShape + 1, img.shape[1] - HaarShape
+ 1))
    dy = np.zeros((img.shape[0] - HaarShape + 1, img.shape[1] - HaarShape
+ 1))
    for idx in range(HaarShape):
        for idy in range(HaarShape):
            dx = dx + img[idx:(idx + dx.shape[0]) , idy:(idy + dx.shape
[1])] * Haar_x[idx, idy]
            dy = dy + img[idx:(idx + dx.shape[0]) , idy:(idy + dx.shape
[1])] * Haar_y[idx, idy]
    Gap = int((NeighborSize+NMS_Size-2)/2)
    vec_dx = dx[Gap:-Gap,Gap:-Gap].flatten()
    vec_dy = dy[Gap:-Gap,Gap:-Gap].flatten()

    # Step 2: Construction C matrix, get det_trace,vec_det_trace
    c11 = np.zeros((dx.shape[0] - NeighborSize + 1, dx.shape[1] - Neighbor
Size + 1)) + 0.0001
    c12 = np.zeros((dx.shape[0] - NeighborSize + 1, dx.shape[1] - Neighbor
Size + 1)) + 0.0001
```

```
    c22 = np.zeros((dx.shape[0] - NeighborSize + 1, dx.shape[1] - Neighbor
Size + 1)) + 0.0001
    for idx in range(NeighborSize):
        for idy in range(NeighborSize):
            c11 = c11 + dx[idx:(idx + c11.shape[0]), idy:(idy + c11.shape
[1])] ** 2
            c12 = c12 + dx[idx:(idx + c11.shape[0]), idy:(idy + c11.shape
[1])] * \
                        dy[idx:(idx + c11.shape[0]), idy:(idy + c11.shape
[1])]
            c22 = c22 + dy[idx:(idx + c11.shape[0]), idy:(idy + c11.shape
[1])] ** 2
    # Find out whether r = lambda_2/lambda_1 >= 0.1 and <=10
    # Or r/(1+r)^2 = det_c/trace_c^2 in [0.25,1]
    trace_c = c11 + c22               # lambda_1 + lambda_2
    det_c = c11*c22 - c12*c12         # lambda_1 * lambda_2
    det_trace = det_c/(trace_c**2)
    Gap = int((NMS_Size-1)/2)
    vec_det_trace = det_trace[Gap:-Gap,Gap:-Gap].flatten()

    # Step 3: Non-maximum suppression over 3x3 region
    Center_Column = (NMS_Size*NMS_Size-1)/2
    Cpr = np.zeros((FinalShape[0] * FinalShape[1], NMS_Size*NMS_Size)) # A
 (H*W,9) matrix
    for idx in range(NMS_Size):
        for idy in range(NMS_Size):
            Cpr[:, idx*NMS_Size+idy] = det_trace[idx:(idx+FinalShape[0]),
                                                 idy:(idy+FinalShape[1])].
flatten()
    Cpr_rowmax = np.max(Cpr, axis = 1)
    Cpr_localmax = np.equal(Cpr_rowmax, Cpr[:, int(Center_Column)])
    return (vec_x, vec_y, np.abs(vec_dx), np.abs(vec_dy), vec_det_trace, C
pr_localmax, HaarShape)
def HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                          HaarShape,NeighborSize,NMS_Size,scale,
                          th_dxy = 100, th_r = 1/4.5, if_save = 1, input_
image = "1"):
    WhetherCorner = np.logical_and(vdx>th_dxy, vr>=th_r);
    WhetherCorner = np.logical_and(vdy>th_dxy, WhetherCorner);
    WhetherCorner = np.logical_and(localmax, WhetherCorner);
    print("Scale = %f, Number of corners: %d" %(scale, np.sum(WhetherCorne
r)))

    bias = int((HaarShape+NeighborSize+NMS_Size-3)/2)
    if if_save:  # Save to image or not
        img_color = Image.open("HW4Pics/%s.jpg" %input_image)
        img_draw = ImageDraw.Draw(img_color)
        for idx,_ in enumerate(WhetherCorner):
            if(WhetherCorner[idx]==1):
                img_draw.ellipse((vy[idx]+bias-2,
                                  vx[idx]+bias-2,
                                  vy[idx]+bias+2,
                                  vx[idx]+bias+2),
                                 fill = 'red', outline ='red')
```

```
                #img_draw.point((vec_x[idx]+FinalSizeRed/2,vec_y[idx]+Fina
lSizeRed/2), fill =(255,0,0))
        del img_draw
        img_color.save("save_%s_%.1f.png" %(input_image,scale))

    FeatureVector = np.zeros((NeighborSize**2, np.sum(WhetherCorner))) #
(Length, # Feature)
    Coord = np.zeros((2, np.sum(WhetherCorner))) # (2, # Feature)
    HalfBias = int(NeighborSize/2)
    Cnt = 0
    for idx, value in enumerate(WhetherCorner):
        if(value==1):
            cx = int(vx[idx]+bias)
            cy = int(vy[idx]+bias)
            FeatureVector[:, Cnt] = img[cx-HalfBias:cx+HalfBias+1, cy-Half
Bias:cy+HalfBias+1].flatten()
            Coord[0, Cnt] = cx
            Coord[1, Cnt] = cy
            Cnt = Cnt + 1
    return FeatureVector, Coord
def SSD(f1, f2):
    return np.sum((f1-f2)**2)
def NCC(f1, f2):
    m1 = np.mean(f1)
    m2 = np.mean(f2)
    top = np.sum((f1-m1)*(f2-m2))
    bottom = np.sqrt(np.sum((f1-m1)**2)*np.sum((f2-m2)**2))
    return top/bottom
def Correspondence(FM1, Coord1, FM2, Coord2, metric = "SSD"): #(L, #), (2,
 #), (L, ##), (2, ##)
    N1 = FM1.shape[1]
    N2 = FM2.shape[1]
    Pairs = []
    Scores = []
    if metric=="SSD":
        for idx1 in range(N1):
            Score = np.array([SSD(FM1[:,idx1], FM2[:,idx2]) for idx2 in ra
nge(N2)])
            idx2 = np.argmin(Score)
            Scores.append(np.min(Score))
            Pairs.append((idx1,idx2))
        # Select 100 pairs
        BestPairs = [x for _,x in sorted(zip(Scores,Pairs))]  # Incresing
order
    if metric=="NCC":
        for idx1 in range(N1):
            Score = np.array([NCC(FM1[:,idx1], FM2[:,idx2]) for idx2 in ra
nge(N2)])
            idx2 = np.argmax(Score)
            Scores.append(np.max(Score))
            Pairs.append((idx1,idx2))
        # Select 100 pairs
        BestPairs = [x for _,x in sorted(zip(Scores,Pairs), reverse = Tru
e)]  # Decresing order
```

```
    return BestPairs[0:int(np.min([100,len(BestPairs)]))]
def DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, scale, me
tric = "SSD"):
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image1)
    left = np.array(img_gray)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image2)
    right = np.array(img_gray)
    output = np.concatenate((left, right), axis = 1)
    img_color = Image.fromarray(output.astype("uint8"), mode = "RGB")
    img_draw = ImageDraw.Draw(img_color)
    for (idx1, idx2) in Pairs:
        x1 = Coord1[0,idx1]
        y1 = Coord1[1,idx1]
        x2 = Coord2[0,idx2]
        y2 = Coord2[1,idx2] + left.shape[1]
        img_draw.line([(y1,x1),(y2,x2)], fill =(255,0,0))
    del img_draw
    img_color.save("Coores_%s_%.1f_%s.png" %(input_image1, scale, metric))
```

4.2 Source code of main function for Harris Corner Detector

```
# Mapping in image 1
for scale in [1.2,2,2.8,3.6]:
    input_image1, scale0, NMS_Size, NeighborSize = ("1", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image1).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector1, Coord1 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image1)
    print("Saved output to save_%s_%.1f.png" % (input_image1,scale))

    input_image2, scale0, NMS_Size, NeighborSize = ("2", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image2).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector2, Coord2 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image2)
    print("Saved output to save_%s_%.1f.png" % (input_image2,scale))

    Pairs = Correspondence(FeatureVector1, Coord1, FeatureVector2, Coord2,
 metric = "NCC")
    DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, scale, me
tric = "NCC")
    print("Find %s pairs" %len(Pairs))
# Mapping in image 2
for scale in [1.2,2,2.8,3.6]:
    input_image1, scale0, NMS_Size, NeighborSize = ("Truck1", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image1).convert('L')
    img = np.array(img_gray).astype("float64")
```

```
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector1, Coord1 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image1)
    print("Saved output to save_%s_%.1f.png" % (input_image1,scale))

    input_image2, scale0, NMS_Size, NeighborSize = ("Truck2", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image2).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector2, Coord2 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image2)
    print("Saved output to save_%s_%.1f.png" % (input_image2,scale))

    Pairs = Correspondence(FeatureVector1, Coord1, FeatureVector2, Coord2,
 metric = "SSD")
    DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, scale, me
tric = "SSD")
    print("Find %s pairs" %len(Pairs))
# Mapping in image 3
for scale in [1.2,2,2.8,3.6]:
    input_image1, scale0, NMS_Size, NeighborSize = ("Fountain1", 1.2, 7, 1
1)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image1).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector1, Coord1 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image1)
    print("Saved output to save_%s_%.1f.png" % (input_image1,scale))

    input_image2, scale0, NMS_Size, NeighborSize = ("Fountain2", 1.2, 7, 1
1)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image2).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector2, Coord2 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image2)
    print("Saved output to save_%s_%.1f.png" % (input_image2,scale))

    Pairs = Correspondence(FeatureVector1, Coord1, FeatureVector2, Coord2,
 metric = "NCC")
    DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, scale, me
tric = "NCC")
```

```
    print("Find %s pairs" %len(Pairs))
# Mapping in image 4
for scale in [1.2,2,2.8,3.6]:
    input_image1, scale0, NMS_Size, NeighborSize = ("Tower1", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image1).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector1, Coord1 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image1)
    print("Saved output to save_%s_%.1f.png" % (input_image1,scale))

    input_image2, scale0, NMS_Size, NeighborSize = ("Tower2", 1.2, 7, 11)
    img_gray = Image.open("HW4Pics/%s.jpg" %input_image2).convert('L')
    img = np.array(img_gray).astype("float64")
    vx, vy, vdx, vdy, vr, localmax, HaarShape = HCD_Kernel(img, scale, NMS
_Size)
    FeatureVector2, Coord2 = HCD_Feaure(vx, vy, vdx, vdy, vr, localmax,
                                HaarShape,NeighborSize,NMS_Size,scale,
                                th_dxy = 100, th_r = 1/4.5, if_save = 1, in
put_image = input_image2)
    print("Saved output to save_%s_%.1f.png" % (input_image2,scale))

    Pairs = Correspondence(FeatureVector1, Coord1, FeatureVector2, Coord2,
 metric = "SSD")
    DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, scale, me
tric = "SSD")
    print("Find %s pairs" %len(Pairs))
```

4.3 Source code of SIFT

```
import numpy as np
import cv2 as cv
def SSD(f1, f2):
    return np.sum((f1-f2)**2)
def NCC(f1, f2):
    m1 = np.mean(f1)
    m2 = np.mean(f2)
    top = np.sum((f1-m1)*(f2-m2))
    bottom = np.sqrt(np.sum((f1-m1)**2)*np.sum((f2-m2)**2))
    return top/bottom
def Correspondence(FM1, Coord1, FM2, Coord2, metric = "SSD"):
    #(#,128), [i].pt=(2,), (##,128), [i].pt=(2,)
    N1 = FM1.shape[0]
    N2 = FM2.shape[0]
    Pairs = []
    Scores = []
    if metric=="SSD":
        for idx1 in range(N1):
            Score = np.array([SSD(FM1[idx1,:], FM2[idx2,:]) for idx2 in ra
nge(N2)])
            idx2 = np.argmin(Score)
```

```
            Scores.append(np.min(Score))
            Pairs.append((idx1,idx2))
        # Select 100 pairs
        BestPairs = [x for _,x in sorted(zip(Scores,Pairs))]  # Incresing
order
    if metric=="NCC":
        for idx1 in range(N1):
            Score = np.array([NCC(FM1[idx1,:], FM2[idx2,:]) for idx2 in ra
nge(N2)])
            idx2 = np.argmax(Score)
            Scores.append(np.max(Score))
            Pairs.append((idx1,idx2))
        # Select 100 pairs
        BestPairs = [x for _,x in sorted(zip(Scores,Pairs), reverse = Tru
e)]  # Decresing order
    return BestPairs[0:int(np.min([100,len(BestPairs)]))]
def DrawPairs(input_image1, input_image2, Coord1, Coord2, Pairs, metric =
"SSD"):
    left = cv.imread("HW4Pics/%s.jpg" %input_image1)
    right = cv.imread("HW4Pics/%s.jpg" %input_image2)
    output = np.concatenate((left, right), axis = 1)
    for (idx1, idx2) in Pairs:
        x1 = int(Coord1[idx1].pt[0])
        y1 = int(Coord1[idx1].pt[1])
        x2 = int(Coord2[idx2].pt[0])
        y2 = int(Coord2[idx2].pt[1]) + left.shape[1]
        cv.line(output, (y1,x1),(y2,x2), (0,0,255), 2)
    cv.imwrite("Coores_%s_%s.png" %(input_image1, metric), output)
# Input
input_image1, input_image2 = ("1", "2")
img = cv.imread('HW4Pics/%s.jpg' %input_image1)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp1,img)
cv.imwrite('save_%s.jpg' %input_image1, img)

# Input
img = cv.imread('HW4Pics/%s.jpg' %input_image2)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp2, des2 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp2,img)
cv.imwrite('save_%s.jpg' %input_image2, img)

Pairs = Correspondence(des1, kp1, des2, kp2, metric = "SSD")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "SSD")
Pairs = Correspondence(des1, kp1, des2, kp2, metric = "NCC")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "NCC")


# Input
```

```
input_image1, input_image2 = ("Truck1", "Truck2")
img = cv.imread('HW4Pics/%s.jpg' %input_image1)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp1,img)
cv.imwrite('save_%s.jpg' %input_image1, img)
print("Find %d key points" %len(kp1))

# Input
img = cv.imread('HW4Pics/%s.jpg' %input_image2)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp2, des2 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp2,img)
cv.imwrite('save_%s.jpg' %input_image2, img)
print("Find %d key points" %len(kp2))

Pairs = Correspondence(des1, kp1, des2, kp2, metric = "SSD")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "SSD")
Pairs = Correspondence(des1, kp1, des2, kp2, metric = "NCC")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "NCC")
```

```
# Input
input_image1, input_image2 = ("Fountain1", "Fountain2")
img = cv.imread('HW4Pics/%s.jpg' %input_image1)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp1,img)
cv.imwrite('save_%s.jpg' %input_image1, img)
print("Find %d key points" %len(kp1))

# Input
img = cv.imread('HW4Pics/%s.jpg' %input_image2)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp2, des2 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp2,img)
cv.imwrite('save_%s.jpg' %input_image2, img)
print("Find %d key points" %len(kp2))

Pairs = Correspondence(des1, kp1, des2, kp2, metric = "SSD")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "SSD")
Pairs = Correspondence(des1, kp1, des2, kp2, metric = "NCC")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "NCC")
```

```
# Input
```

```
input_image1, input_image2 = ("Tower1", "Tower2")
img = cv.imread('HW4Pics/%s.jpg' %input_image1)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp1,img)
cv.imwrite('save_%s.jpg' %input_image1, img)
print("Find %d key points" %len(kp1))

# Input
img = cv.imread('HW4Pics/%s.jpg' %input_image2)
# SIFT Feature
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)
sift = cv.xfeatures2d.SIFT_create()
kp2, des2 = sift.detectAndCompute(gray,None)
cv.drawKeypoints(img,kp2,img)
cv.imwrite('save_%s.jpg' %input_image2, img)
print("Find %d key points" %len(kp2))

Pairs = Correspondence(des1, kp1, des2, kp2, metric = "SSD")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "SSD")
Pairs = Correspondence(des1, kp1, des2, kp2, metric = "NCC")
DrawPairs(input_image1, input_image2, kp1, kp2, Pairs, metric = "NCC")
```