ECE 661 Computer Vision

Homework 4

Vineeth Ravi

September 25, 2018

PU ID : 0030019456

EMAIL ID : ravi24@purdue.edu

1 Introduction

In this homework, we detect and extract interest points from two different images of the same scene, using the Harris Corner Detector and SIFT algorithm, and we use techniques like NCC , SSD and euclidean distance metric to establish and map corresponding points between the two images. We display the correspondences, between interest points, by drawing a line between these points and showing the images together. We find Interest points in both images, that maybe invariant to changes in scale and rotation. We use the Harris Corner Detector algorithm and SIFT to detect these points, and use NCC and SSD to map the corresponding points, by using them as metrics of similarity. The algorithms and techniques used are described in detail in the sections below:-

Theory and Mathematical Background

2 Harris Corner Detector

We implement the Harris Corner Detector Algorithm to find interest points or corners. A corner is a point where the pixel has gray-scale levels have significant variation in at least two directions. The two orthogonal directions where there is maximum variation define the orientation of the corner. We implement the following steps :-

1) Once we read the images, we define a scale space representation for the images $\sigma = [1, 2, 3, 4, \dots]$ etc. and so on. The values of σ can be tuned for different image pairs to yeild best results. We use Haar wavelets (filters) of size 4σ, such that it is an even integer. We use the filters in x and y directions by convolving with the image. For eg:- when $\sigma = \sqrt{2}$, we have:-

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

Different scales, have different sizes of Haar filters based on the N value defined previously above. Once we convolve with the gray scale values of the pixels in images, we G_x and G_y respectively. The Harris corners can be detected properly at appropriate scales using properly sized Haar Filters.

2) As mentioned previously, we compute the gradients $G_x = I * h_x$ and $G_y = I * h_y$, by convolving the image with the Haar Filters, and we compute the squares and products of the gradients, which we use to construct the C matrix using the values $G_{x^2} = G_x^2$, $G_{y^2} = G_y^2$ and $G_{xy} = G_x G_y$. We use a window size of 5σ, such that it is an odd integer to compute the Harris Corner Detector Responses for every image Pixel using the C matrix, with values given by :-
$C_{11} = \Sigma G_{x^2}$, $C_{22} = \Sigma G_{y^2}$ and $C_{12} = \Sigma G_{xy}$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{12} & C_{22} \end{bmatrix}$$

3) We now, compute the Harris Corner responses given by $\frac{det(C}{trace(C)^2})$ for every pixel, and we use a threshold value to reject corners, whose value is less than the mean value of the entire image.

4) We use non-maximum suppression, and find the maximum corner response by using a pre-determined window size and we finalize the set of Harris Corners for a given image. The local maxima are considered to be strong corners, since they exhibit a relatively large amount of variation in gray levels along two orthogonal axis Only these final points with response greater than the threshold are considered.

5) Now that we have computed the Harris Corners for both images, we now employ similarity measures like Sum of Squared Distance (SSD) and Normalized Cross Correlation (NCC) metrics for mapping the corresponding interest points between images.

Point-to-Point correspondence

Once we have got the corners, we now establish correspondences between them, by directly comparing gray levels. We use the SSD and NCC metric.

2.1 SSD: Sum of Squared Differences

One method of finding correspondences, is to use the sum of squared differences. That is, for each pair of interest points in both input images I1 and I2 , we take a patch size of (S+1) x (S+1), and compute the following metric :-

$$SSD = \sum_{i=-S/2}^{i=S/2} \sum_{j=-S/2}^{j=S/2} |I_1(i,j) - I_2(i,j)|^2$$

Once, we have computed the SSD matrix, we find minimal value pairs, and add them as corresponding pairs of interest points.

2.2 NCC: Normalized Cross Correlation

Second Method of finding correspondences, is to use the normalized-cross-correlation That is, for each pair of interest points in both input images I1 and I2 , we take a patch size of (S+1) x (S+1), and compute the following metric :-

$$NCC = \frac{\sum_{i=-S/2}^{i=S/2} \sum_{j=-S/2}^{j=S/2} (I_1(i,j) - m_1)(I_2(i,j) - m_2)}{\sqrt{(\sum_{i=-S/2}^{i=S/2} \sum_{j=-S/2}^{j=S/2} (I_1(i,j) - m_1)^2)(\sum_{i=-S/2}^{i=S/2} \sum_{j=-S/2}^{j=S/2} (I_2(i,j) - m_2)^2)}}$$

where m_i, is the mean value of the patch taken for computing the metric in the individual image. A perfect match or score of 1 indicates similarity between two corresponding interests points. So, we find the maximum value pairs and add them as interest points to lists.

3 SIFT:Scale Invariant Feature Transform

The SIFT algorithm was described in detail during the lecture, So we would be mentioning the steps only briefly here as part of the Homework4.

1) Scale space extremum detection
SIFT is based entirely on the scale space analysis to identify interest points as discussed in class. Let $ff(x, y, \sigma))$ be the Laplacian of the Gaussian of Image I(x,y). The Gaussian is given by $\frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{2\pi\sigma^2}$ We use the Difference of Gaussian (DOG) technique to compute the Pyramid and final all the local extrema. This is computed by :-

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
$$D(x, y, \sigma) = (ff(x, y, k\sigma) - ff(x, y, \sigma))$$

We consider a 3x3x3 neighbourhood around 26 points for a given pixel in the image, and we find the local extremum, maxima or minima in a 3D-Space (x,y, σ)

2) Localization of KeyPoints and discarding weak extremum
We use different scales to build the DoG pyramid. While sampling this pyramid , we use second-order derivatives and localize the extremum with sub pixel accuracy using Taylor expansion such that :-

$$D(x) = D(x_0) + J^T(x_0) + 1/2x^T H(x_0)x$$

where

$$J(x_0) = (\frac{dD}{dx}, \frac{dD}{dy}, \frac{dD}{d\sigma})^T$$

and

$$H(x_0) = \begin{bmatrix} \frac{d^2D}{dx^2} & , \frac{d^2D}{dxdy} & \frac{d^2D}{dxd\sigma} \\ \frac{d^2D}{dydx} & , \frac{d^2D}{dy^2} & \frac{d^2D}{dyd\sigma} \\ \frac{d^2D}{d\sigma dx} & , \frac{d^2D}{d\sigma dy} & \frac{d^2D}{dx\sigma^2} \end{bmatrix}$$

Now SIFT can be interpolated near local extremum and find the accurate position of the original image.

$$X = -H^{-1}J$$

We now proceed to remove weak extrema using a threshold to remove points along edges. So, we eliminate the edges, as we do for the Harris corner detector using the determinant and trace, instead of finding the ratio of eigenvalues.

3) Computing Window Maxima along with Orientation and assignment
At each point in a SxS neighbourhood around the extremum, we compute the gradient magnitude m(x,y) and gradient orientation $\theta(x, y)$ such that :-

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

Hence we have found the dominant local orientation. by calculating gradient vector of the Gaussian smoothed image $ff(x, y, \sigma)$ at scale σ .
We now weight $\theta(x, y)$ with m(x,y) and create a histogram of 36 bins spanning 360 degrees.

4) 128 bit Descriptors
Each point is surrounded by 16x16 neighbourhood of points, which is divided into 4x4 cells. For each cell, we create an 8 bit orientation, as mentioned in the previous step. Repeating this for 16 cells, we get and 128-element descriptor in 128 dimensional space. The length is normalized to make it invariant to change in illumination.

5) Now that, we have found the 128 bit descriptors and pixel coordinates for interest points in both images, we employ the euclidean distance metric to find corresponding interest points. This is described in the section below:-

3.1 Point-to-Point correspondence for SIFT : Euclidean Distance Metric

When using SIFT feature descriptors, we compare the correspondences between image pairs, by using the Euclidean distance as a metric between the 128 bit descriptors for two interest points. The euclidean distance formula is given by:-

$$d = (\sqrt{\sum_{i=1}^{i=128} (a_i - b_i)^2})$$

where a and b are the two descriptors for the corresponding interest points, whose similarity we want to compare. Similar to SSD, we find the minimal value pair and add them as corresponding pairs of interest points.

4 Experimental Results, Observation and Comments

Quality of Correspondences :-
The correspondences got from the Harris Corner detector are of lower quality and not robust when compared to those got from SIFT. There are also a lot of more correspondences with better quality and accuracy got when using SIFT instead of Harris Corner Detector. So, it is better to use SIFT for locating and mapping interest points.

INTEREST POINTS DETECTED AT VARIOUS SCALE USING HARRIS CORNER DETECTOR :-
When we increase the scale, the number of detected interest points reduce. The interest points for the Harris corner detector are not robust, and depend a lot on the hyper-parameters which we used to implement the algorithm. The interest points located at corners have a high response to the Harris detector. But the number of points detected reduces as we increase scale. So, we are then forced to reduce the threshold dynamically to maintain the same number of correspondences and interest points. This particular feature of the Harris detector is shown for the TRUCK image. Here we tune the threshold and scale dynamically, to ensure more quality correspondences and mappings are made. The Harris detector also picks a lot less of false positives as we increase the scale. The interest points obtained for the Harris detector are dependent on the scale value, whereas SIFT is a very robust estimator, for finding interest points and mappings. For a corner point to remain detectable at all scales using HC detector, it must be a large, and sharply defined corner, with no noise in vicinity, that would smooth out and blur the corner at larger scales.

RELATIVE PERFORMANCE:-
As mentioned previously SIFT is very robust algorithm. It also finds and matches a lot of more interest points. It does not depend on a lot of parameters, So, yields better quality mappings. With SIFT, the number of interest points obtain do not vary for each scale in contrast to the Harris corner detector. Since, we have more key points, we can expect more correspondences of points using SIFT. Normalizing the descriptor makes it invariant to change in illumination. This was experimented on, using last years "stadium" image, where lights are different in both images. These descriptors are unique and offer more information regarding a corner. When compared to Harris, which depends on the type of similarity metric i.e. SSD, NCC used, the quality changes, Where as SIFT only depends on the similarity between the feature vectors of the L2 norm. Harris may perform poorly if the parameters are not chosen appropriately. SIFT auto generates the key points, and maps interest points with a better quality and more number of matches when compared to Harris, (fixing the set of parameter space).

The NCC metric is better when compared to SSD metric, in establishing point-to-point correspondences. SSD is more dependent on the parameters used, and it is not as robust as NCC, when we run both similarity measures and compare the interest point mappings obtained. It is also expected NCC should work better than SSD when there are changes in illumination. So, I would use the NCC metric, since it gives better results across different pairs of images keeping the other parameters fixed.

Parameters :-
The below hyper-parameters are used for tuning to get the results in mapping interest points. They are :-

harris parameter	Description	Value
σ	scale space values used across images (both i and $\sqrt{2} * i$ used)	[i=(1,2,3,4); (i and $\sqrt{2} * i$)]
N_{Haar}	Haar Filter Size	4σ (even)
N_{HCDR}	Harris Corner C matrix Computation	5σ (odd)
W_{winmax}	Windows size of non maxima suppression	29
SSD/NCC Parameters		
$N_{wsize-ssd}$	Window Size of SSD	21
$N_{wsize-ncc}$	Window Size of NCC	21
$SSD_{threshold}$	SSD threshold (tuned dynamically and statically) for image pairs)	min(SSD)*10
$NCC_{threshold}$	NCC threshold (tuned dynamically and statically) for image pairs)	0.95
SIFT Parameters		
$SIFT_{threshold}$	Sift threshold (tuned dynamically and statically) for image pairs)	min(L_2 norm)*2

The SSD and NCC threshold included in the table above work well, for most of the input images. For some of the images, if the number of correspondences were less or if they were of poor quality, Then some of the parameters were tuned, to give better results (i.e. more correspondences, with higher quality). This was done statically sometimes, But other wise using a dynamic threshold, which always generated 40-50 correspondence points also works as well.

The SSD threshold is set to be 10 * min(SSD), for it to be dynamically used across all images ; but this is changed in case it yields poor results for a specific scale value for a specific image pair, to (100-200)*min(SSD) .

Similarly the NCC threshold is set to 0.95, Since we want only the best correspondences to be retained. But, even this sometimes yields poor quality results, So we tune this as well depending on the need to increase the number of correspondences.

The Euclidean threshold is set to 2*min(distance), But sometimes even this is relaxed to 3, to ensure SIFT yields good quality and increased number of corresponding invariant point mappings.

The values mentioned in the table, are slightly varied for some image pairs, to get better quality and increased number of correspondences across scales. These are used for feature extraction and matching.

5 Results

Only the required results performed are included in this report. That is the 4 sets of scale space values are included, though 8 sets were carried out, to tune the parameters.
The parameters used, are those mentioned in the table.
However, the parameters are tuned for some of the images to different values to yield higher number and better quality of corresponding points, by using different thresholds and parameters than those mentioned in the table. Set of scale space values are :-

$\sigma = [1, 2, 4.2, 5.6]$

5.1 set1

Input Images

Figure 1: Input Image 1

Figure 2: Input Image 2

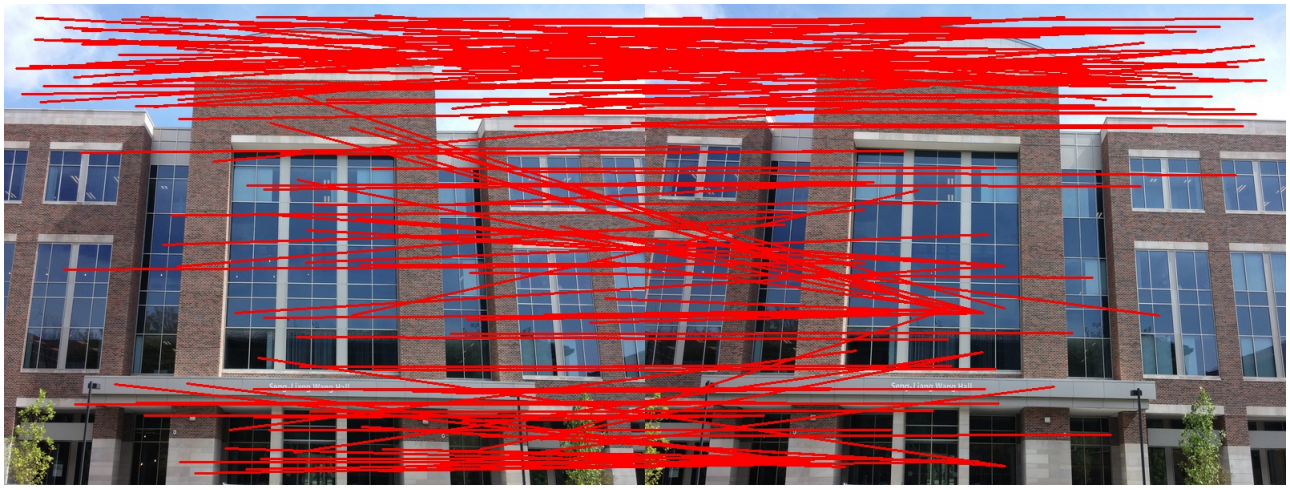

Figure 3: Image pair correspondences using SSD with scale = 1

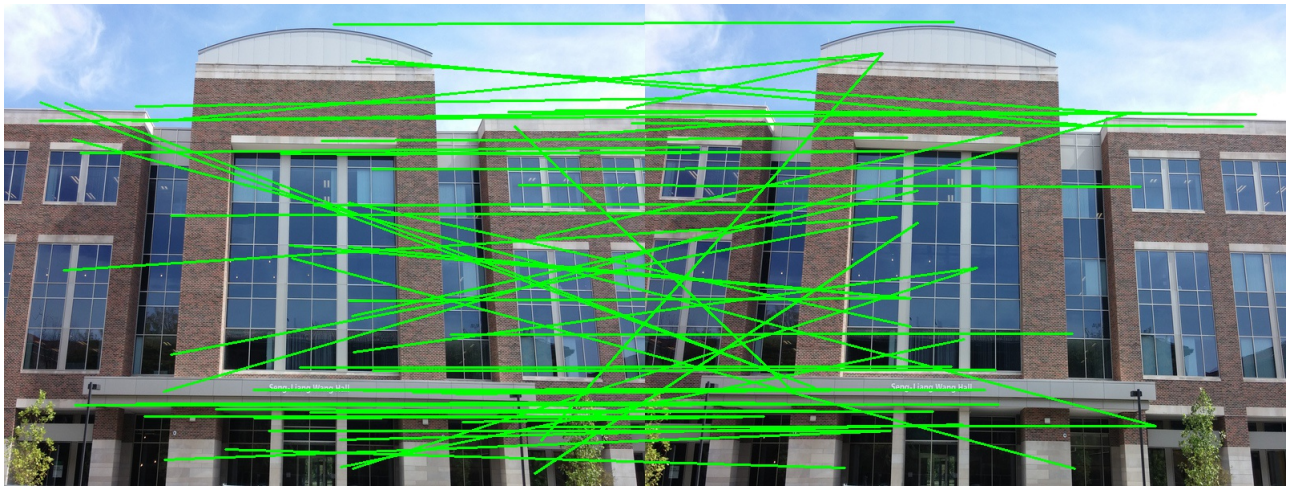

Figure 4: Image pair correspondences using NCC with scale = 1

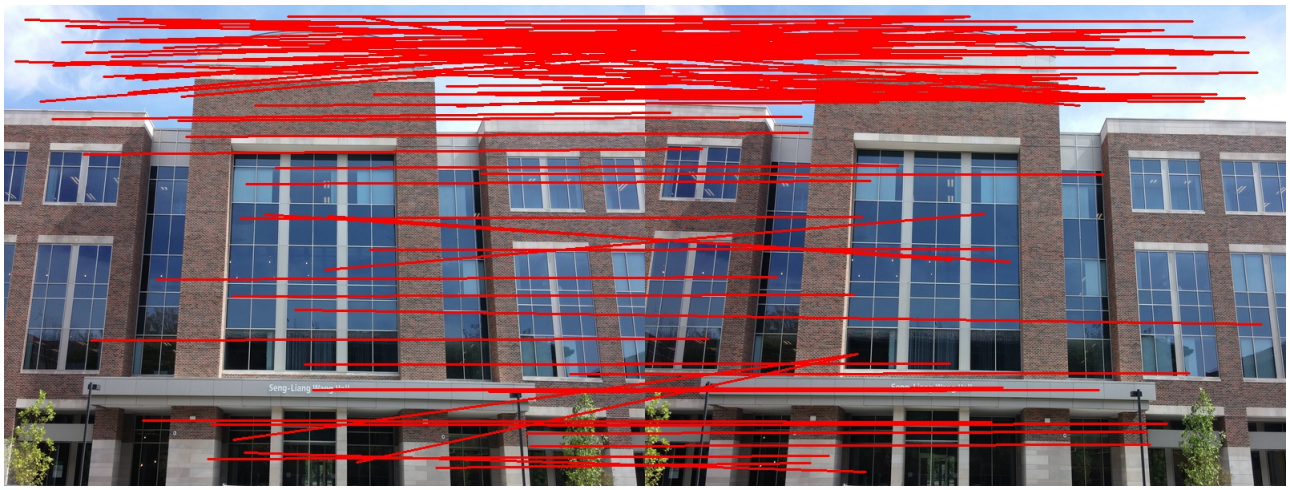

Figure 5: Image pair correspondences using SSD with scale = 2

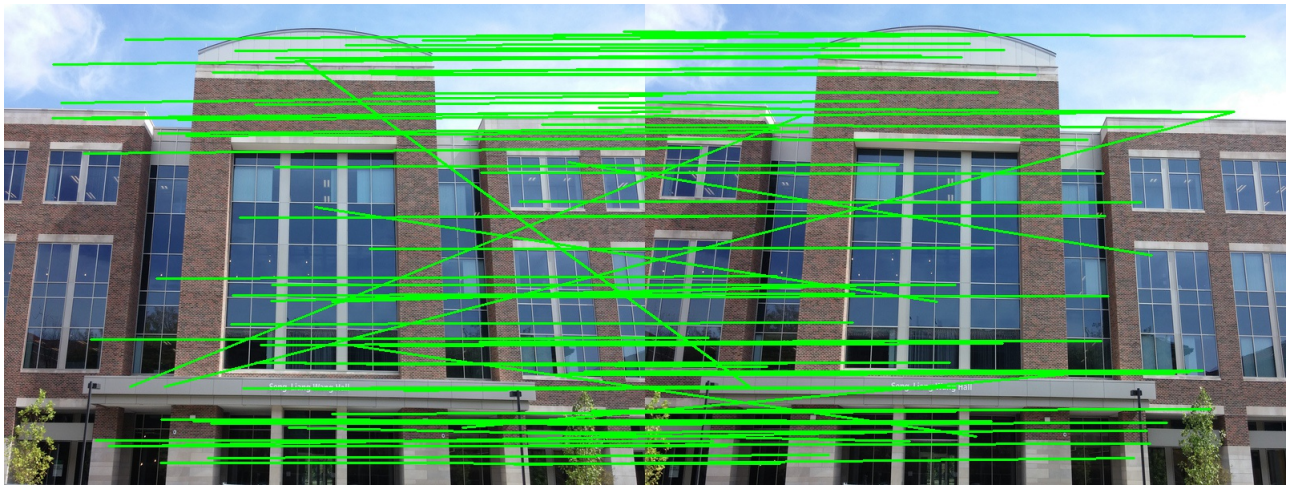

Figure 6: Image pair correspondences using NCC with scale = 2

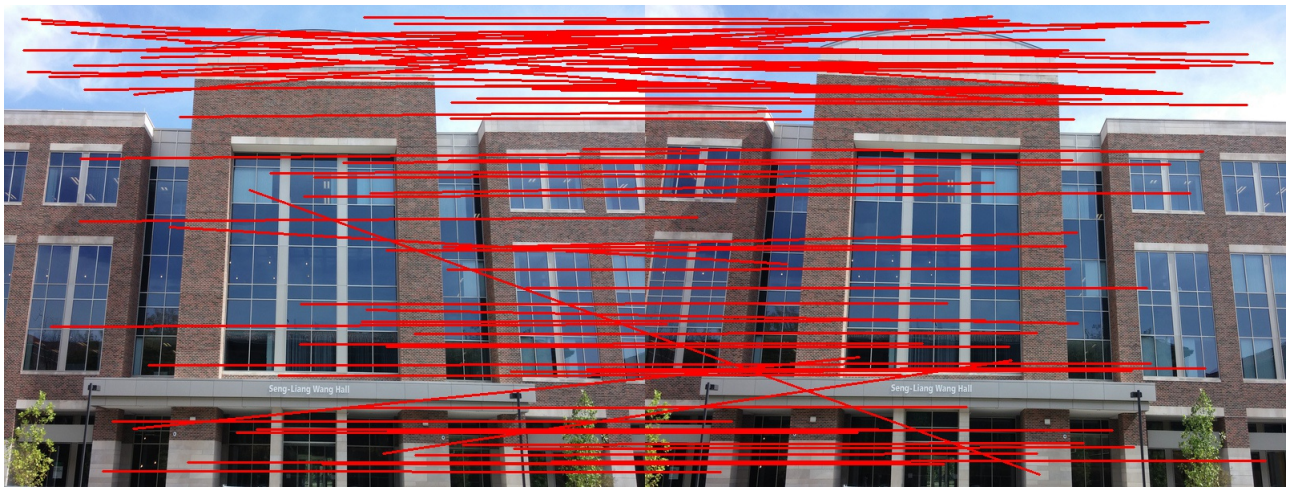

Figure 7: Image pair correspondences using SSD with scale = 4.2

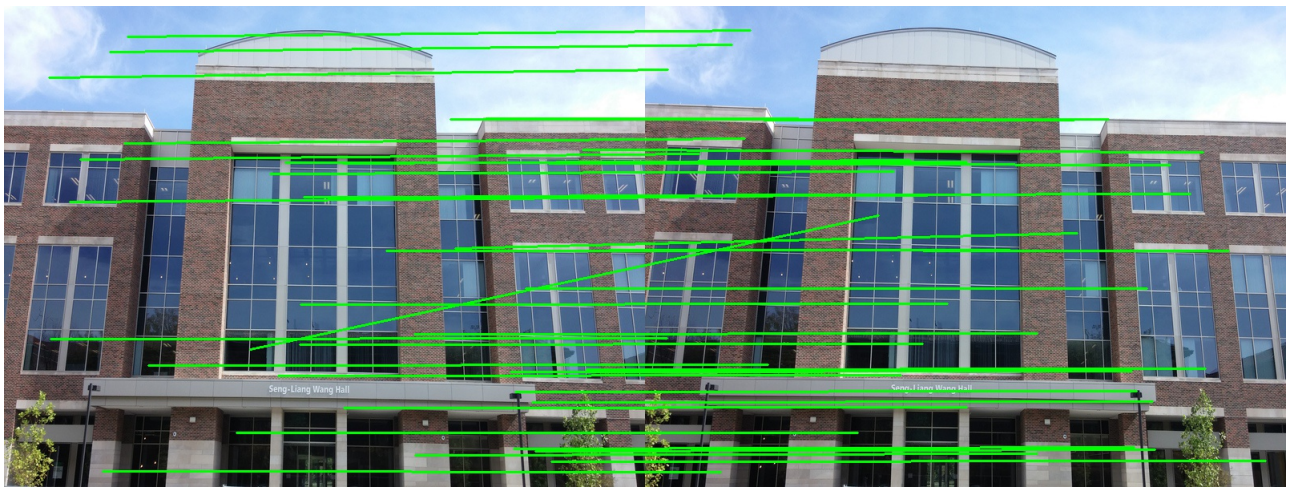

Figure 8: Image pair correspondences using NCC with scale = 4.2

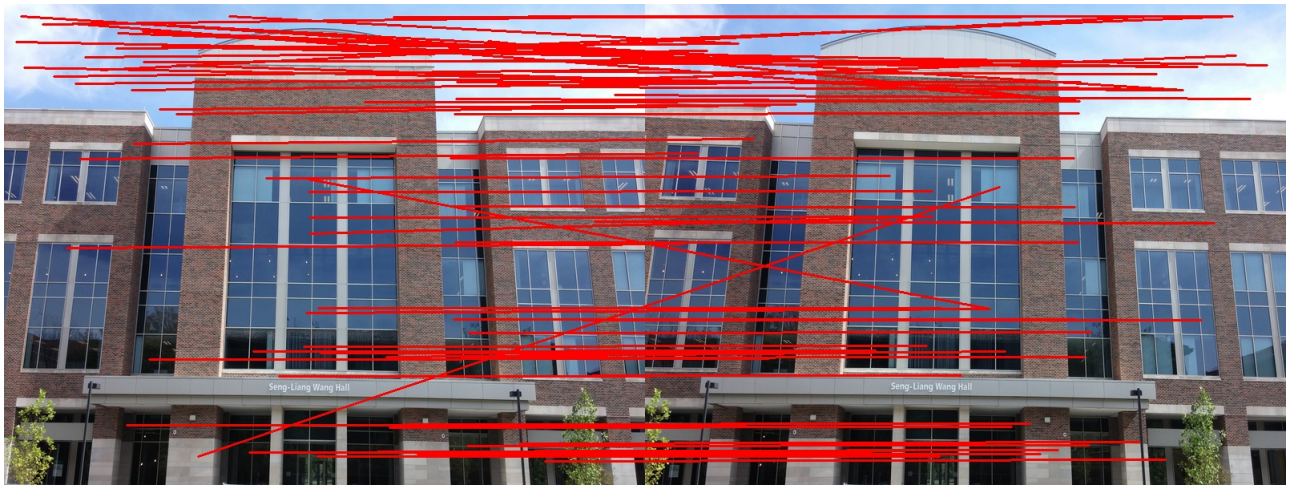

Figure 9: Image pair correspondences using SSD with scale = 5.6

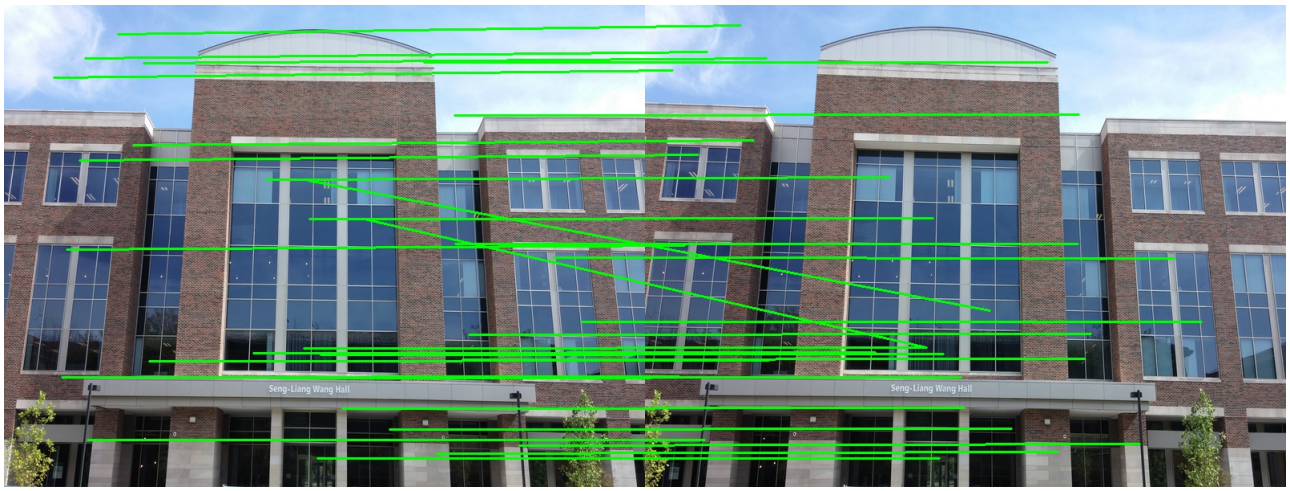

Figure 10: Image pair correspondences using NCC with scale = 5.6

SIFT

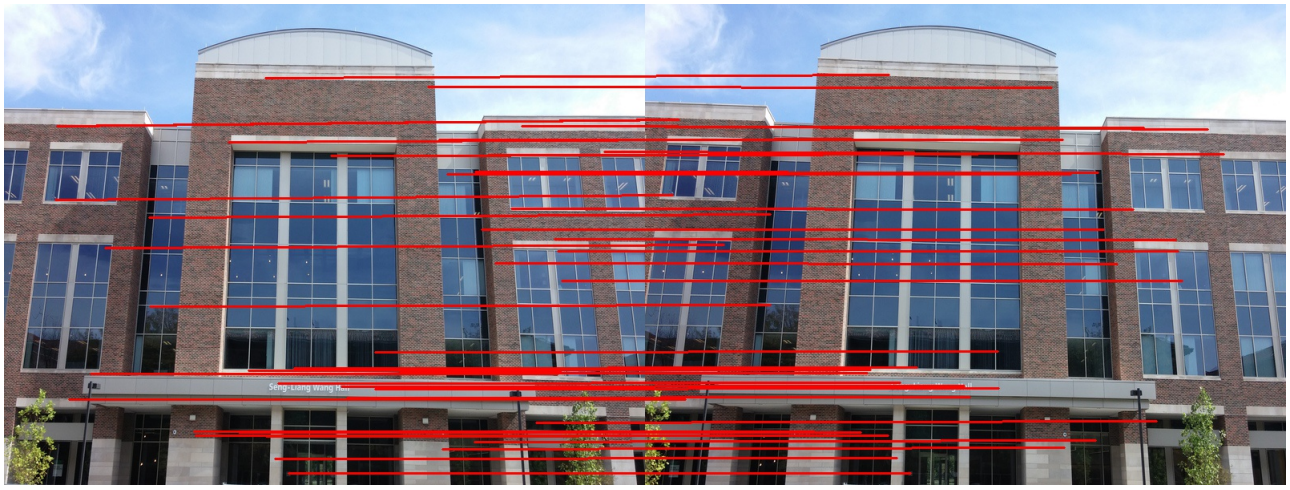

Figure 11: Image pair correspondences using SIFT

5.2 set2

Input Images

Figure 12: Input Image 1

Figure 13: Input Image 2

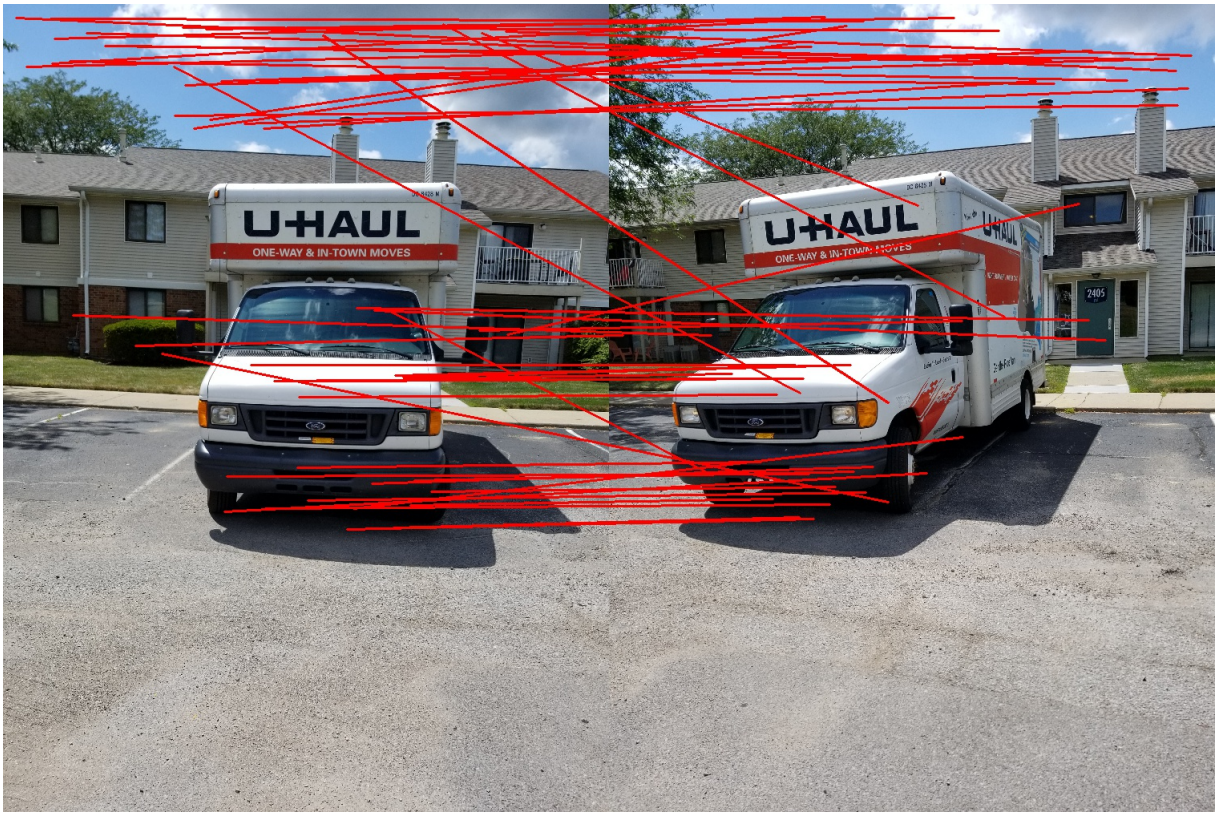

Figure 14: Image pair correspondences using SSD with scale = 1

Figure 15: Image pair correspondences using NCC with scale = 1

Figure 16: Image pair correspondences using SSD with scale = 2

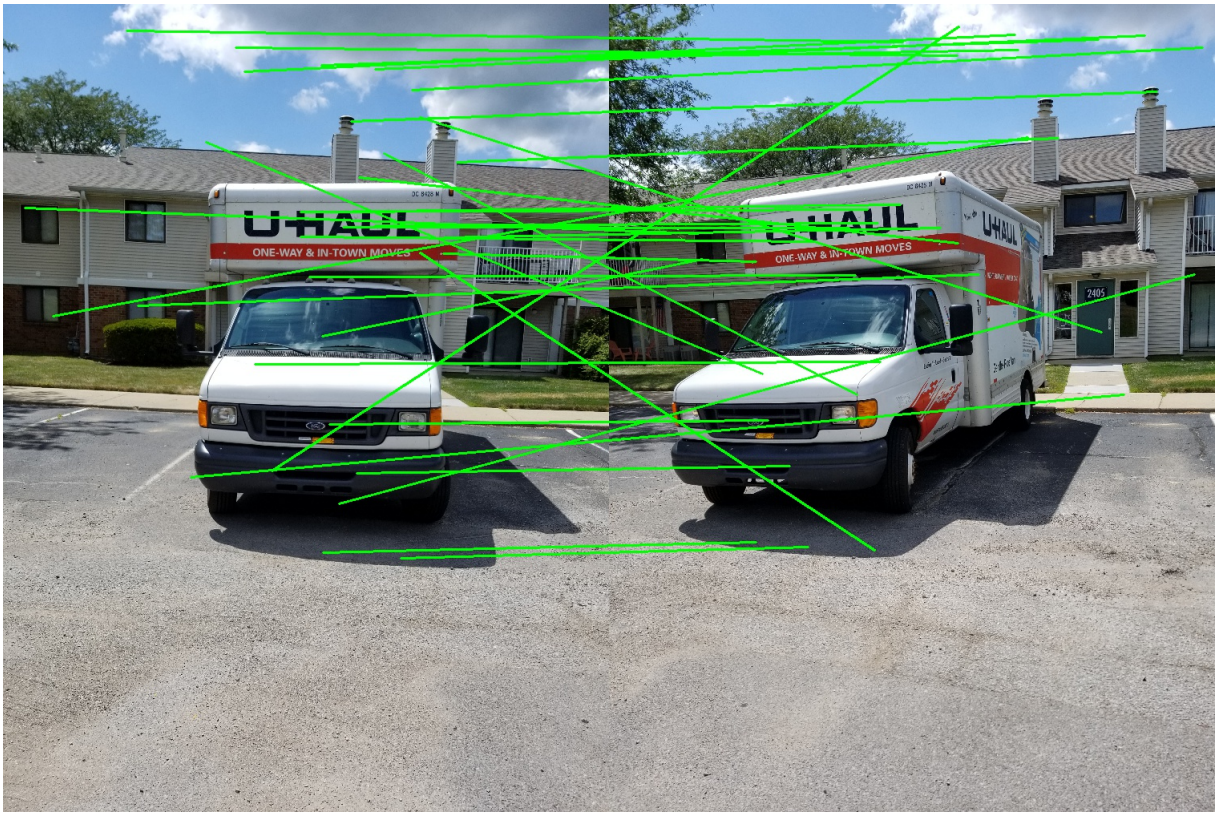

Figure 17: Image pair correspondences using NCC with scale = 2

Figure 18: Image pair correspondences using SSD with scale = 4.2

Figure 19: Image pair correspondences using NCC with scale = 4.2

Figure 20: Image pair correspondences using SSD with scale = 5.6

Figure 21: Image pair correspondences using NCC with scale = 5.6

SIFT

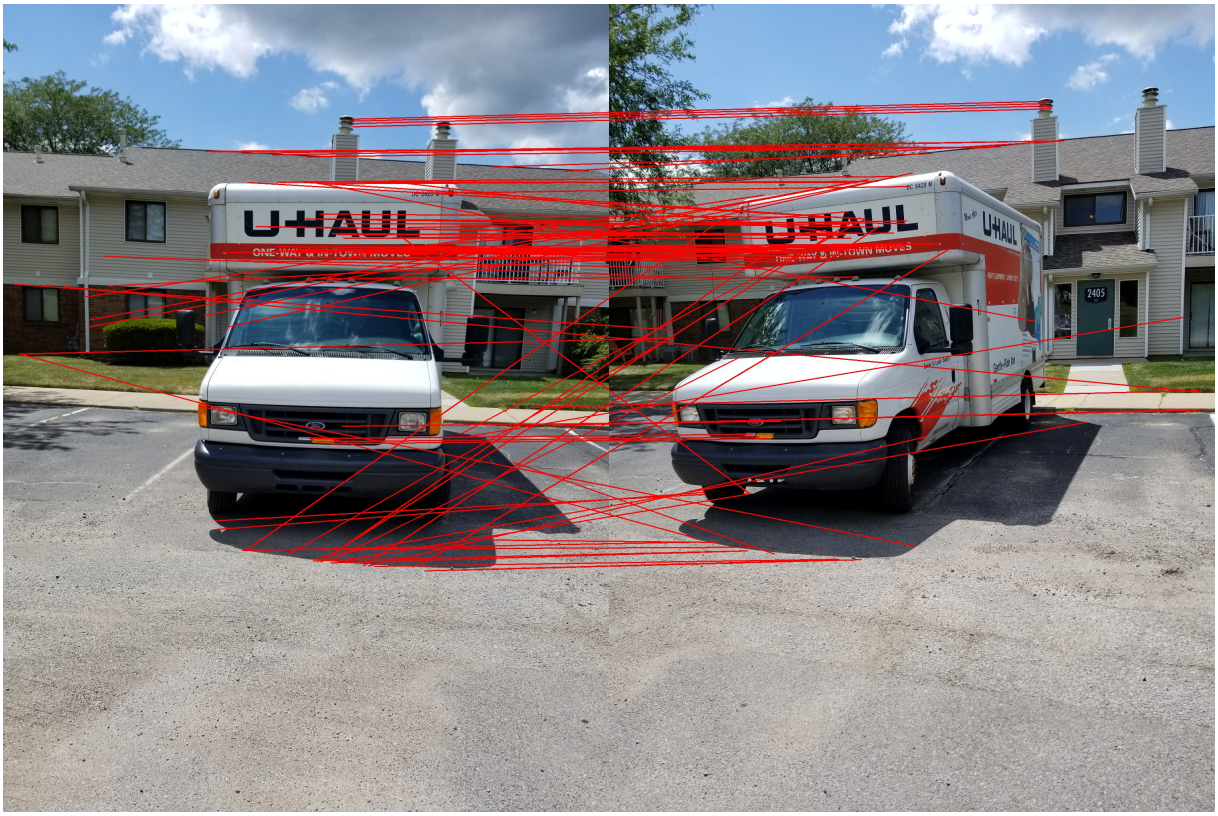

Figure 22: Image pair correspondences using SIFT

5.3 Custom Images Input : set3 : From previous years data set

Input Images

Figure 23: Input Image 1

Figure 24: Input Image 2

Harris Corner Detector

Figure 25: Image pair correspondences using SSD with scale = 1

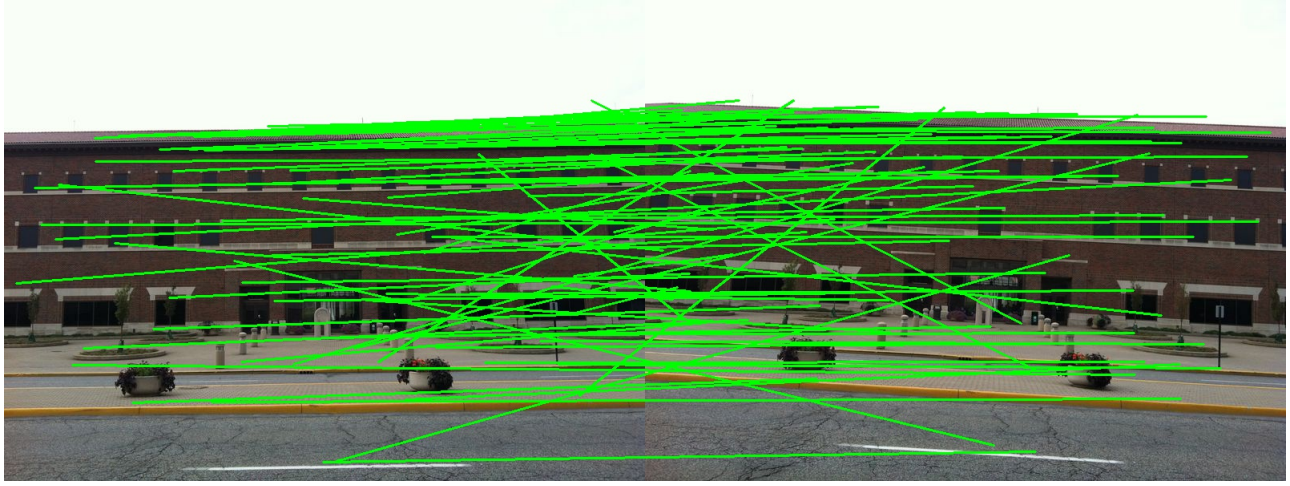

Figure 26: Image pair correspondences using NCC with scale = 1

Figure 27: Image pair correspondences using SSD with scale = 2

Figure 28: Image pair correspondences using NCC with scale = 2

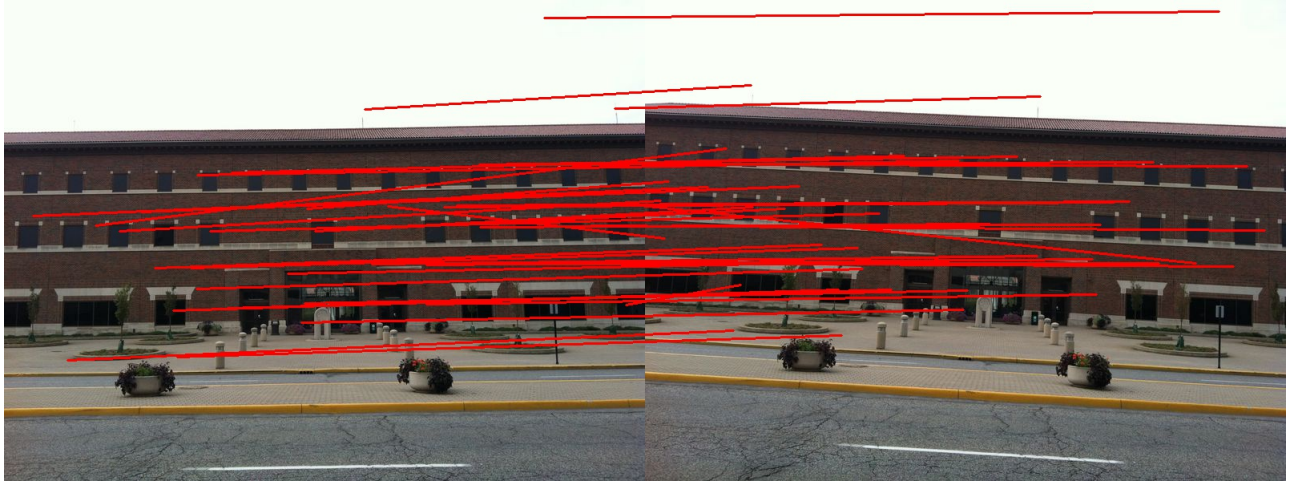

Figure 29: Image pair correspondences using SSD with scale = 4.2

Figure 30: Image pair correspondences using NCC with scale = 4.2

Figure 31: Image pair correspondences using SSD with scale = 5.6

Figure 32: Image pair correspondences using NCC with scale = 5.6

SIFT

Figure 33: Image pair correspondences using SIFT

5.4 Custom Images Input : set4 : From previous years data set

Input Images

Figure 34: Input Image 1

Figure 35: Input Image 2

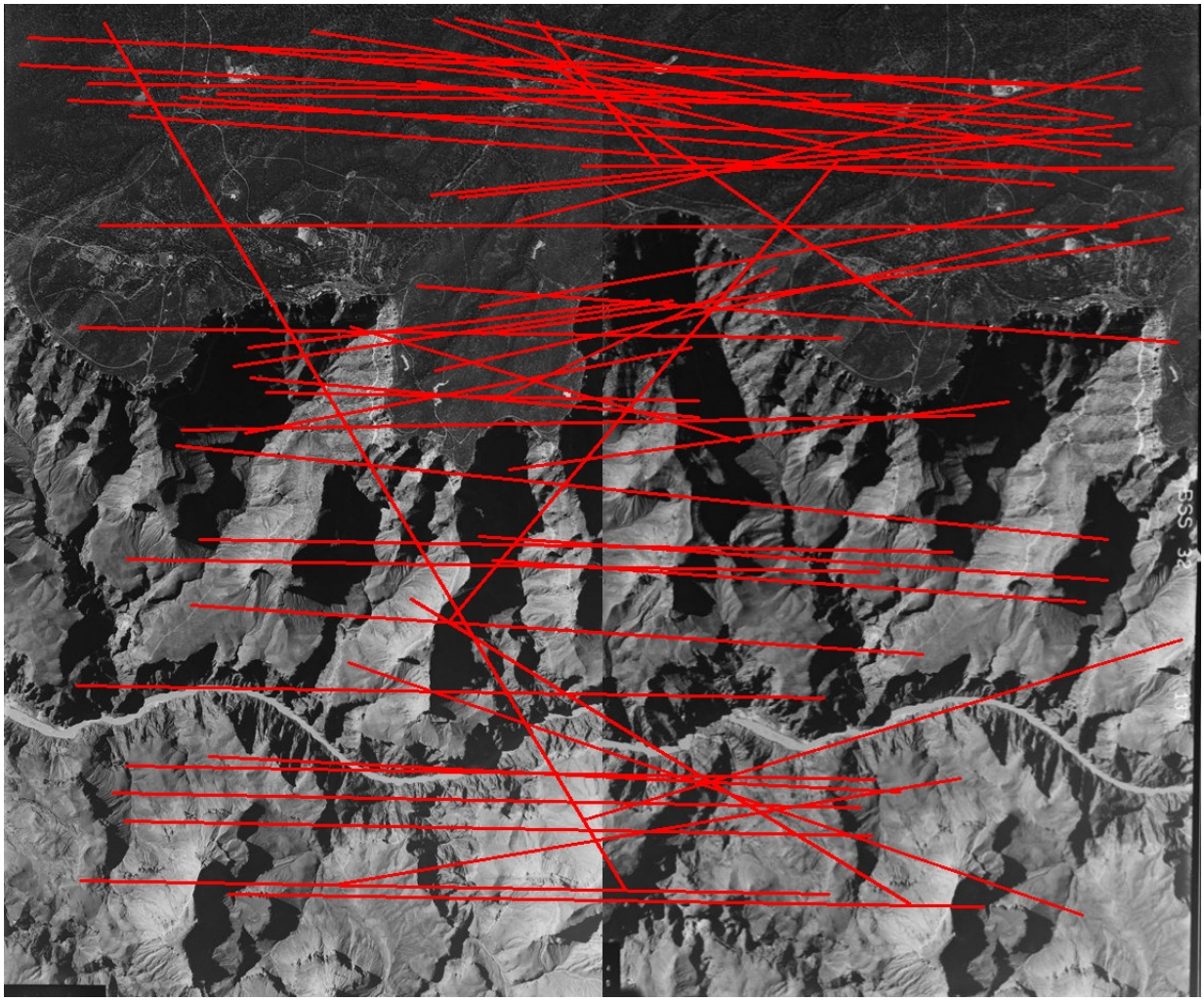

Figure 36: Image pair correspondences using SSD with scale = 1

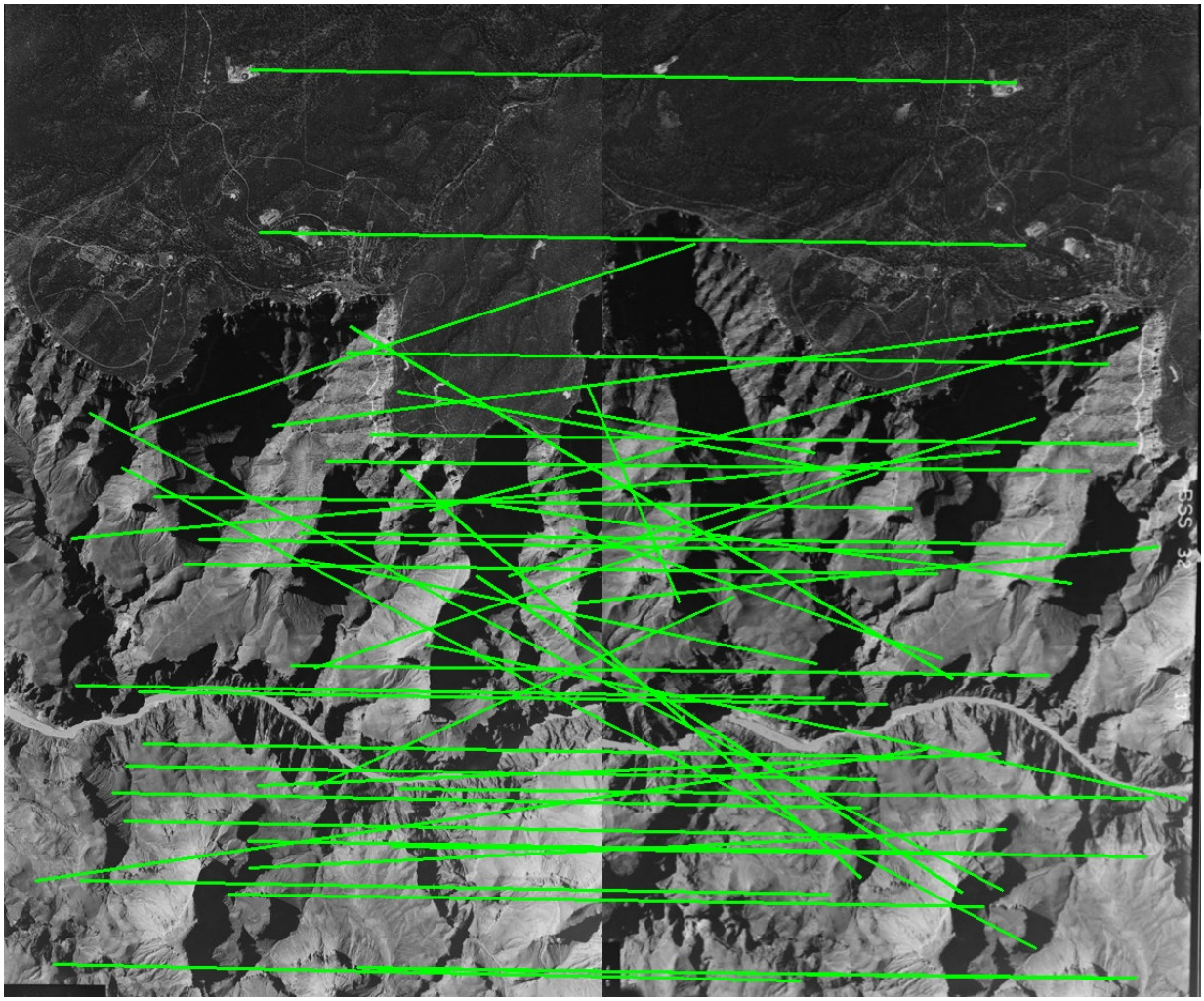

Figure 37: Image pair correspondences using NCC with scale = 1

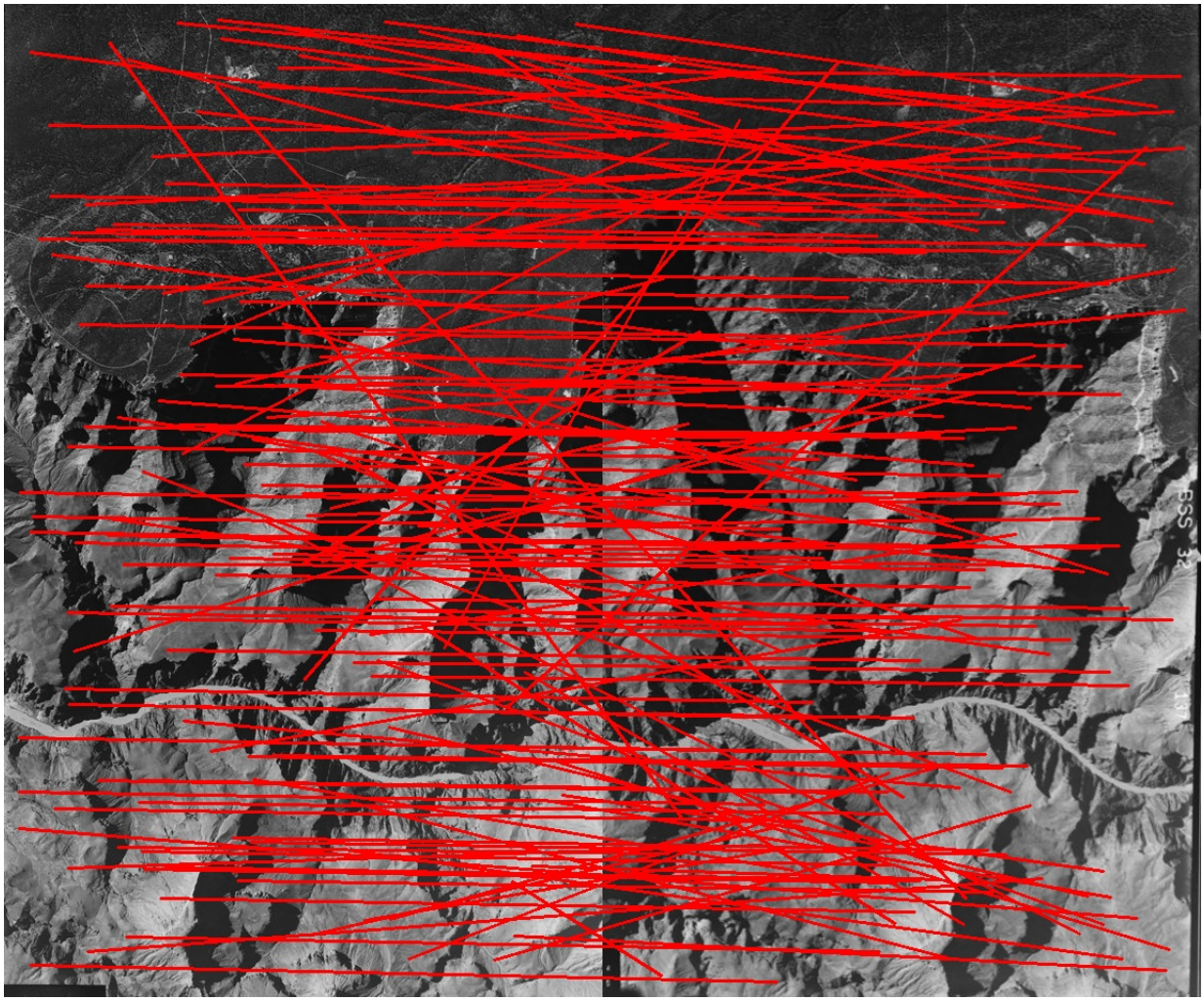

Figure 38: Image pair correspondences using SSD with scale = 2

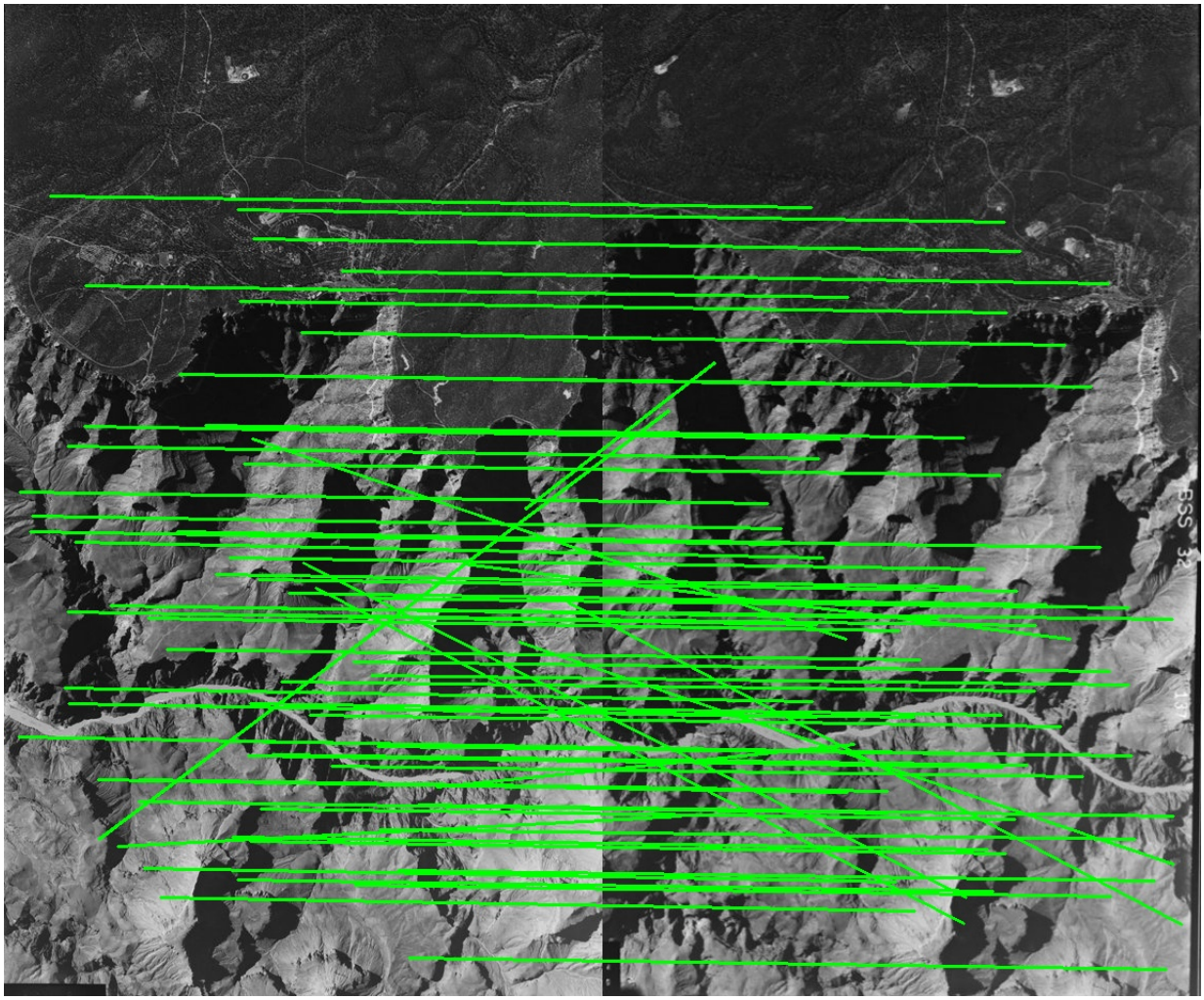

Figure 39: Image pair correspondences using NCC with scale = 2

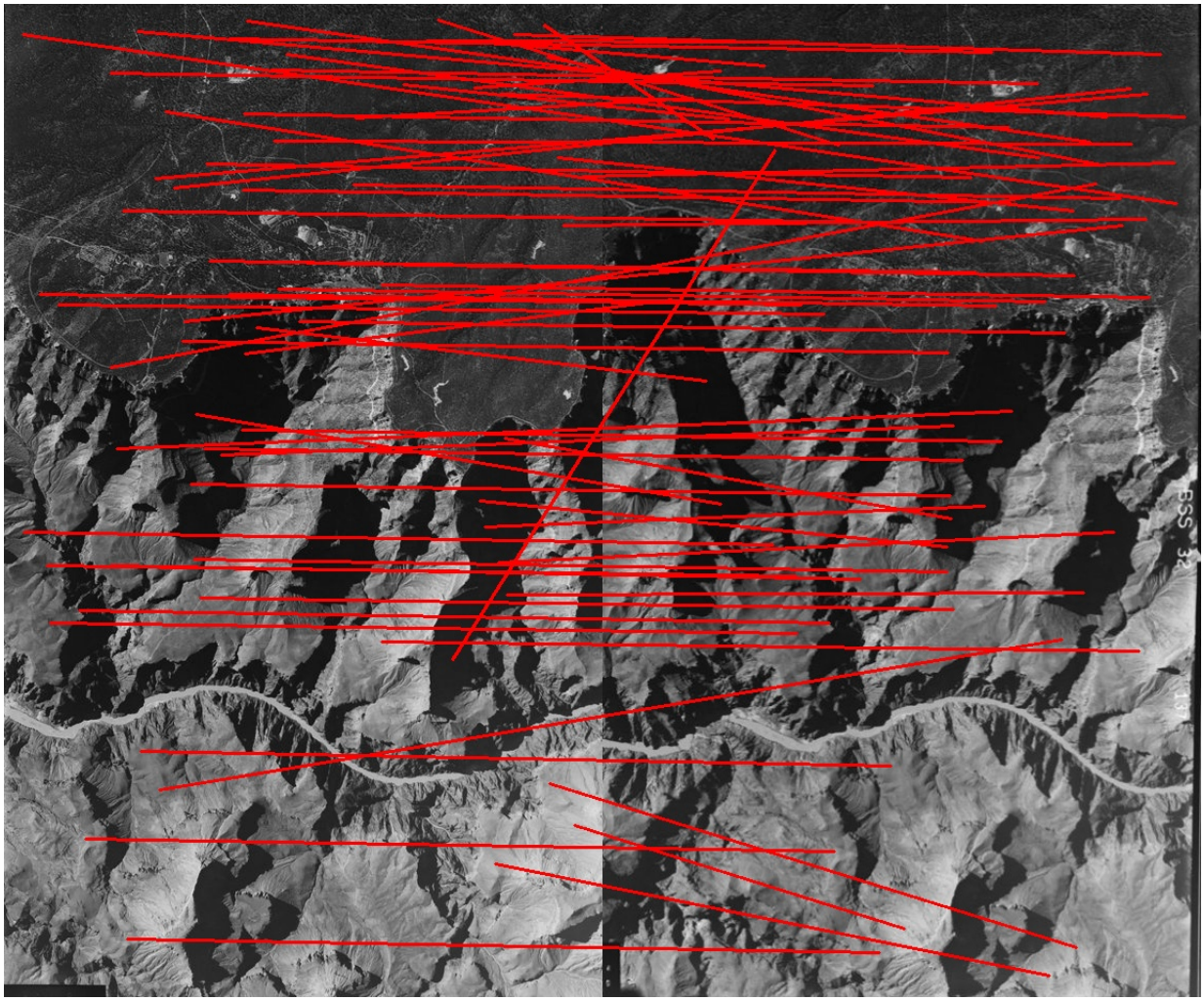

Figure 40: Image pair correspondences using SSD with scale = 4.2

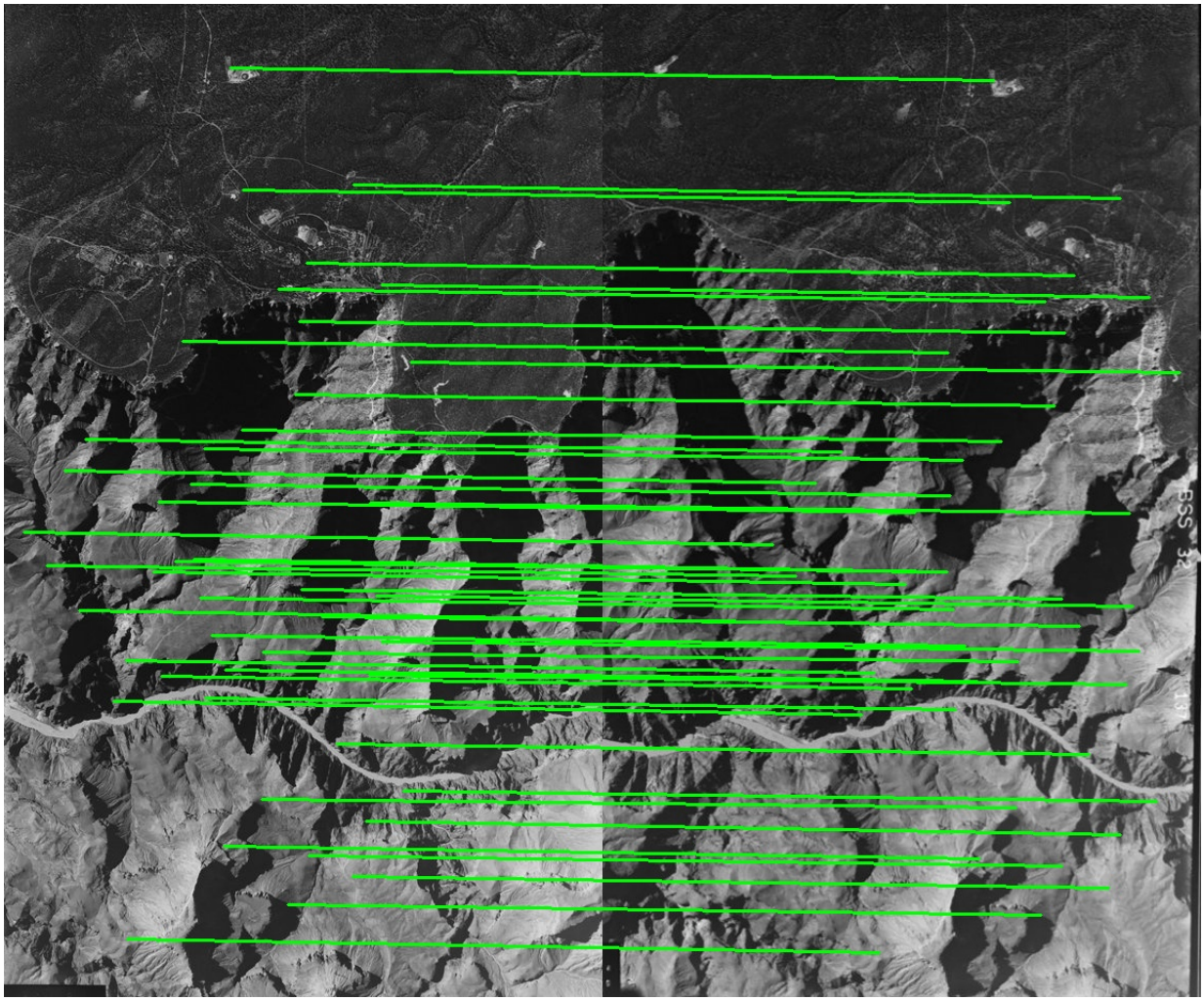

Figure 41: Image pair correspondences using NCC with scale = 4.2

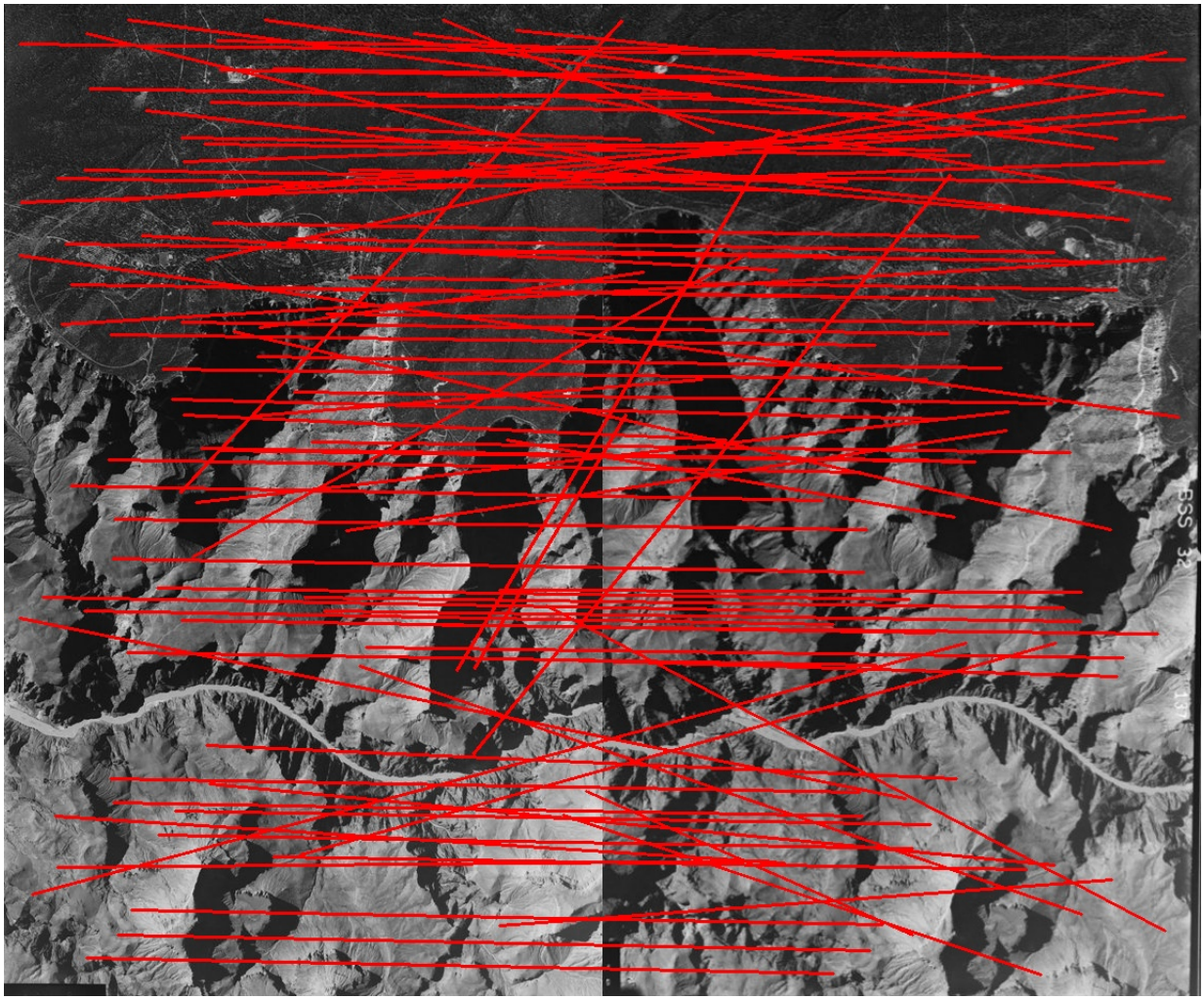

Figure 42: Image pair correspondences using SSD with scale = 5.6

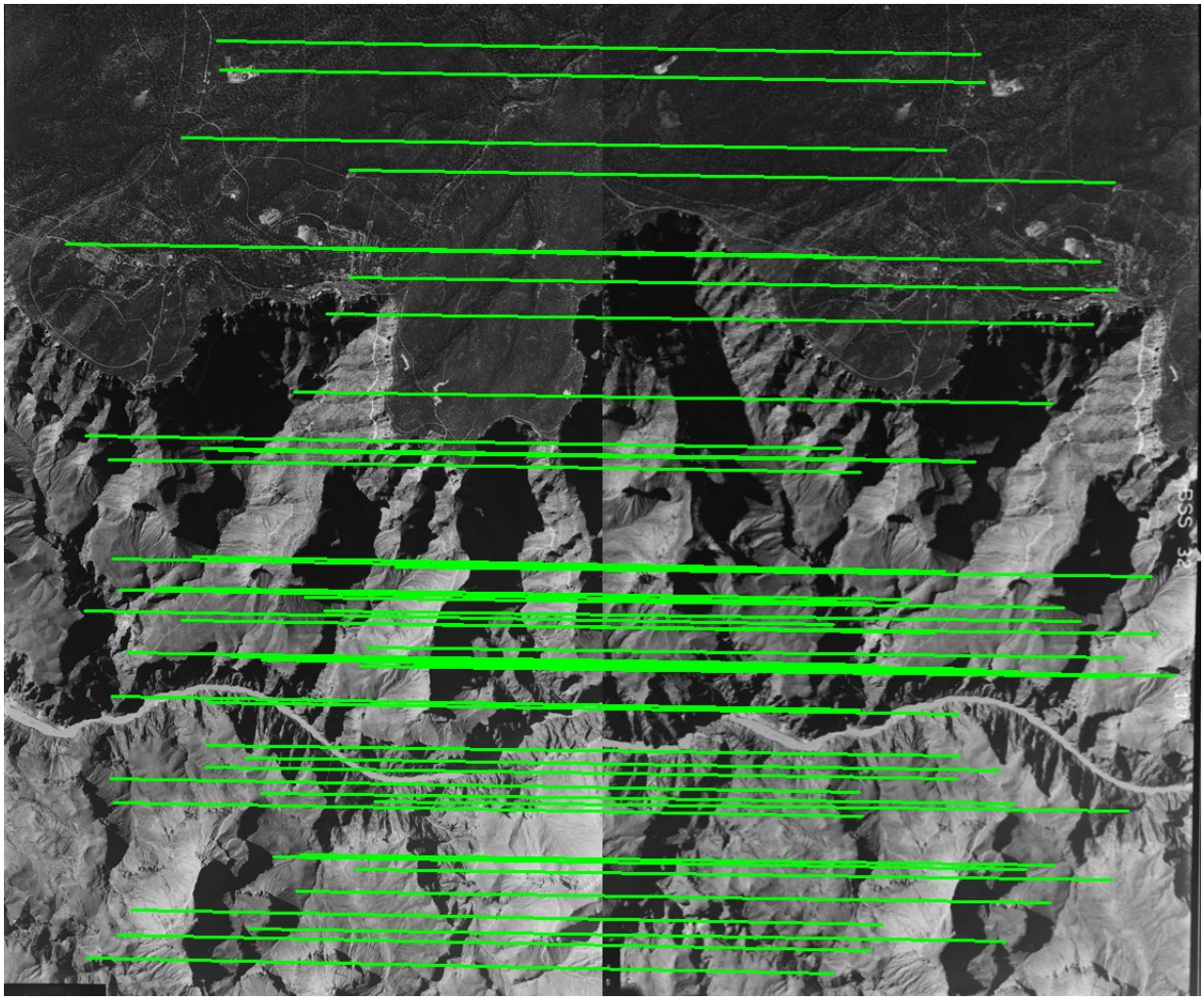

Figure 43: Image pair correspondences using NCC with scale = 5.6

SIFT

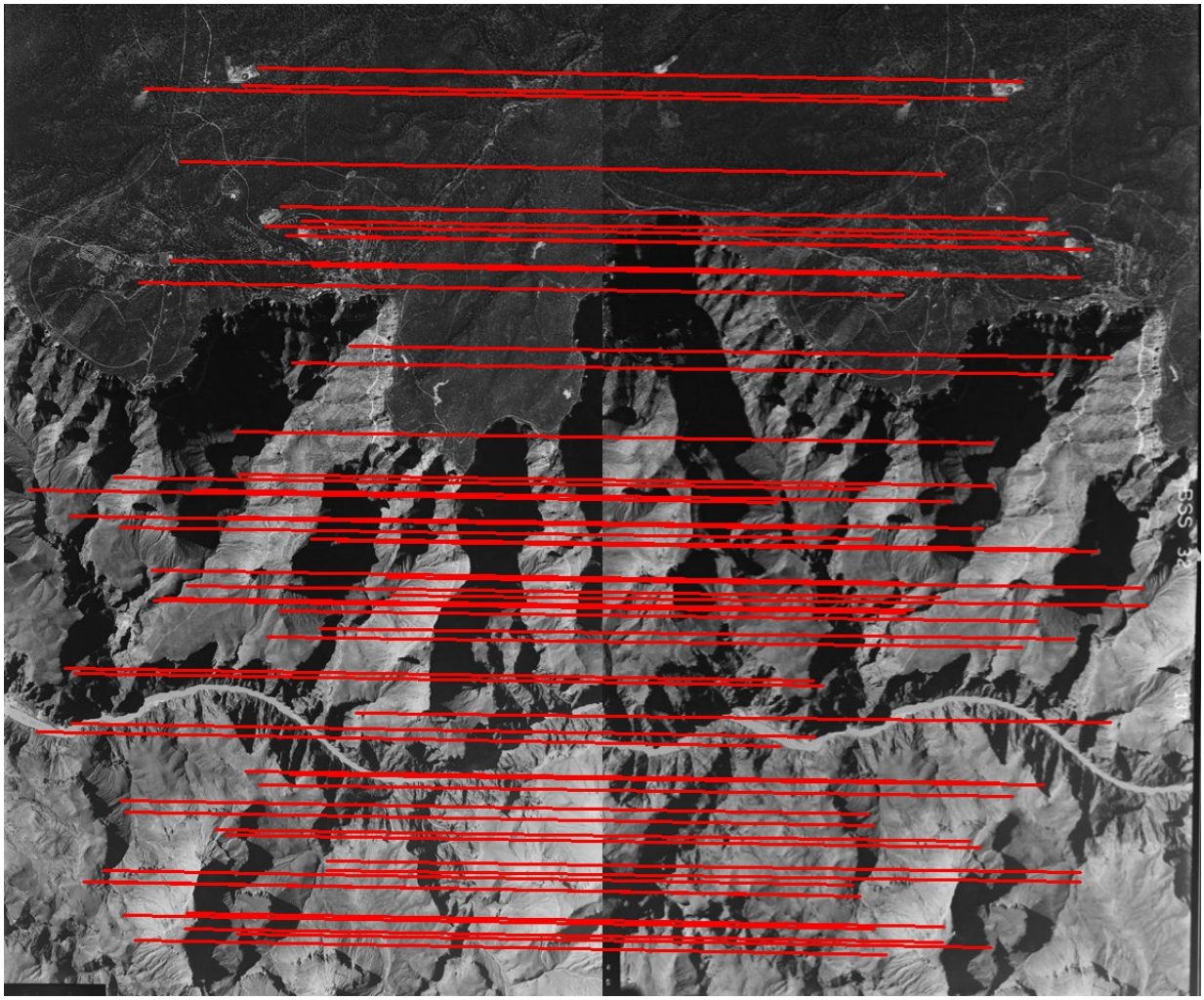

Figure 44: Image pair correspondences using SIFT

As discussed before, the SIFT algorithm performs the best in finding and mapping the corresponding interest points. The NCC metric for computing similarity yields better results in mapping similar corresponding invariant points, when compared to the SSD metric. Also we observe, that as we increase σ the scale space value parameter, the quality increases (i.e. the Mapping lines are more flat), and we also are left with lesser corresponding point pairs, (if we keep the parameters fixed), So, if we increase σ, we might need to change some other parameter like threshold value, to maintain the same number of corresponding point pairs, Else we would have fewer number of point mappings between the two images, though they would be of good quality. This has been done for some of the images in Set 2, as well as some of the other results, I have experimented with using different scale-space values for σ .

6 Python Code

6.1 Harris Code

```
import numpy as np
import cv2 as cv
import math
import matplotlib.pyplot as plt
# Importing neccessary libraries for HW4

# The function below is used for drawing lines, between points.
# The line is drawn between the two images, where corresponding points are matched.

def draw_lines(image1,SSDpts1,NCCpts1,image2,SSDpts2,NCCpts2):

    s1=np.shape(image1)
    s2=np.shape(image2)
    # Combining Both the images, to be placed adjacent to each other
    SSDimage=np.zeros((s1[0],s1[1]+s2[1],3))
```

```
    SSDimage[:,0:s1[1]]=image1
    SSDimage[:,s1[1]:s1[1]+s2[1]]=image2

    NCCimage=np.zeros((s1[0],s1[1]+s2[1],3))
    NCCimage[:,0:s1[1]]=image1
    NCCimage[:,s1[1]:s1[1]+s2[1]]=image2

    for i in range(0,len(SSDpts1)): # Drawing Lines for SSD points matched
        cv.line(SSDimage,(SSDpts1[i][1],SSDpts1[i][0]),
        (s1[1]+SSDpts2[i][1],SSDpts2[i][0]),(0,0,255),thickness=2)

    for i in range(0,len(NCCpts1)): # Drawing Lines for NCC points matched
        cv.line(NCCimage,(NCCpts1[i][1],NCCpts1[i][0]),
        (s1[1]+NCCpts2[i][1],NCCpts2[i][0]),(0,255,0),thickness=2)

    return SSDimage,NCCimage

# Matching the corresponding interest points, between the two images
def match_harris_corners(image1,Harris_Corners1,image2,Harris_Corners2):

    N_Wsize=21  # Window size used for computing the SSD and NCC distance metrics

    l1=len(Harris_Corners1)
    l2=len(Harris_Corners2)

    SSD=np.zeros((l1,l2)) # SSD matrix
    NCC=np.zeros((l1,l2)) # NCC matrix

    SSD_pts1=[]      # Lists to the store the points which have most similarity
    SSD_pts2=[]
    NCC_pts1=[]
    NCC_pts2=[]

    for i in range(0,l1): # Extracting Patches around every interest point
        W1=image1[-(N_Wsize/2)+Harris_Corners1[i][0]:(N_Wsize/2)+
        Harris_Corners1[i][0]+1,-(N_Wsize/2)+
        Harris_Corners1[i][1]:(N_Wsize/2)+Harris_Corners1[i][1]+1]
        m1=np.mean(W1)
        s1=W1-m1
        for j in range(0,l2):
            W2=image2[-(N_Wsize/2)+
            Harris_Corners2[j][0]:(N_Wsize/2)+ Harris_Corners2[j][0]+1,
            -(N_Wsize/2)+Harris_Corners2[j][1]:(N_Wsize/2)+
            Harris_Corners2[j][1]+1]
            m2=np.mean(W2)  # computing mean value
            s2=W2-m2
            SSD[i][j]=np.sum((W1-W2)*(W1-W2)) # Computing SSD metric
            if(np.sum(s1*s2)!=0):
                NCC[i][j]=np.sum(s1*s2)/(np.sqrt(np.sum(s1*s1)*np.sum(s2*s2)))
            # Computing NCC metric

    SSD_threshold=np.min(SSD)*10     # Threshold for SSD
    NCC_threshold=0.95               # Threshold for NCC
    # The thresholds were modified dynamically and statically
    for different image pairs and scales
    # This was done to increase number of interest point correspondences
    list1=[]
    list2=[]

    for i in range(0,len(SSD)):
        val=np.min(SSD[i])
        idx=np.argmin(SSD[i])  # Computing the similarity metric against the threshold
        if((val<=SSD_threshold)and(idx not in list1)):
            list1.append(idx)  # Appending the points with most similarity for SSD
```

```
            SSD_pts1.append([Harris_Corners1[i][0],Harris_Corners1[i][1]])
            SSD_pts2.append([Harris_Corners2[idx][0],Harris_Corners2[idx][1]])

    for i in range(0,len(NCC)):
        val=np.max(NCC[i])
        idx=np.argmax(NCC[i]) # Computing the similarity metric against the threshold
        if((val>=NCC_threshold)and(idx not in list2)):
            list2.append(idx) # Appending the points with most similarity for NCC
            NCC_pts1.append([Harris_Corners1[i][0],Harris_Corners1[i][1]])
            NCC_pts2.append([Harris_Corners2[idx][0],Harris_Corners2[idx][1]])


    return SSD_pts1,SSD_pts2,NCC_pts1,NCC_pts2

# Computing the Harris Corner Points
def harris_corner(image,sigma):

    N_Haar =int(round(np.ceil(4*sigma/2))*2)
    # Haar filter N value to be chosen as even number based on sigma value
    HaarX=np.zeros((N_Haar,N_Haar))
    HaarY=np.zeros((N_Haar,N_Haar))

    HaarX[:,0:N_Haar/2]=-1      # Initializing the HaarX filters
    HaarX[:,N_Haar/2:N_Haar]=1

    HaarY[0:N_Haar/2]=1       # Initializing the HaarY filters
    HaarY[N_Haar/2:N_Haar]=-1
    # Applying the Haar Filters to the image
    Gx=cv.filter2D(image,-1,HaarX).astype('float')
    Gy=cv.filter2D(image,-1,HaarY).astype('float')

    Gx2=Gx*Gx   # Computing the gradients in different directions
    Gy2=Gy*Gy
    Gxy=Gx*Gy

    N_HC=int(round(np.ceil(5*sigma/2))*2)+1
    # Computing the N patch size of computing C matrix to identify corner points
    HC_dr=np.zeros(np.shape(image))
    count=0
    for i in range(N_HC/2,np.shape(image)[0]-N_HC/2):
        for j in range(N_HC/2,np.shape(image)[1]-N_HC/2):
            c11=np.sum(Gx2[-(N_HC/2)+i:(N_HC/2)+i+1,-(N_HC/2)+j:(N_HC/2)+j+1])
            c22=np.sum(Gy2[-(N_HC/2)+i:(N_HC/2)+i+1,-(N_HC/2)+j:(N_HC/2)+j+1])
            c12=np.sum(Gxy[-(N_HC/2)+i:(N_HC/2)+i+1,-(N_HC/2)+j:(N_HC/2)+j+1])
            # Computing the C Matrix, and identifying the threshold for eigen values
            C=np.array([[c11,c12],[c12,c22]])
            if(np.trace(C)!=0):
                HC_dr[i][j]=np.linalg.det(C)/(np.trace(C)*np.trace(C))
            # Computing the Harris corner detector responses
    N_WinMax=29  # Window Size for computing the maxima, and
    finalizing the interest points in an image
    Harris_Corners=[]
    # List for storing the interest points
    for i in range(N_WinMax/2,np.shape(image)[0]-N_WinMax/2):
        for j in range(N_WinMax/2,np.shape(image)[1]-N_WinMax/2):
            R1=HC_dr[-(N_WinMax/2)+i:(N_WinMax/2)+i+1,-(N_WinMax/2)+j:(N_WinMax/2)+j+1]
            H_Max=np.max(R1)
            #  Finding the maxima, based on the Window Patch size used
            if((HC_dr[i][j]==H_Max)and(HC_dr[i][j]>np.mean(HC_dr))):
                Harris_Corners.append([i,j])
            # Computing the corner with the threshold of mean value in the matrix
    return Harris_Corners
```

```
# Reading the two images

image1clr=cv.imread("1.jpg")
image2clr=cv.imread("2.jpg")
# Used for reducing the size of truck images, to improve computation speed.
#image1clr=cv.resize(image1clr, (0,0), fx=0.25, fy=0.25)
#image2clr=cv.resize(image2clr, (0,0), fx=0.25, fy=0.25)
# Converting from color to grayscale
image1=cv.cvtColor(image1clr,cv.COLOR_RGB2GRAY).astype('float')
image2=cv.cvtColor(image2clr,cv.COLOR_RGB2GRAY).astype('float')
# Sigma array of values, we can add or delete any number of values we want in the scale space.
# Feel free to add more values, to compute and map interest points at that scale.
sigma=[1,2,3,4]

for i in range(0,4): # Computing the interest points and mapping for every scale
    Harris_Corners1=harris_corner(image1,sigma[i]) # COMPUTING the Harris
    corner detector responses
    Harris_Corners2=harris_corner(image2,sigma[i])
    # Computing the Distance metrics for SSD and NCC to establish
    similarity among the interest points
    SSD_pts1,SSD_pts2,NCC_pts1,NCC_pts2=match_harris_corners(image1,
    Harris_Corners1,image2,Harris_Corners2)
    # Drawing Lines between the corresponding interest points in the image
    SSDimage,NCCimage=draw_lines(image1clr,SSD_pts1,NCC_pts1,image2clr,SSD_pts2,NCC_pts2)

    cv.imwrite('SSDimage'+str(sigma[i])+'.jpg',SSDimage)
    cv.imwrite('NCCimage'+str(sigma[i])+'.jpg',NCCimage)
    # Writing the Output corresponding interest points in both images, with lines drawn
```

6.2 SIFT Code

```
import numpy as np
import cv2 as cv
import math
import matplotlib.pyplot as plt
# Importing neccessary libraries for HW4

# The function below is used for drawing lines, between points.
# The line is drawn between the two images, where corresponding points are matched.
def draw_lines(image1,sift_pts1,image2,sift_pts2):

    s1=np.shape(image1)
    s2=np.shape(image2)
    # Combining Both the images, to be placed adjacent to each other
    Siftimage=np.zeros((s1[0],s1[1]+s2[1],3))
    Siftimage[:,0:s1[1]]=image1
    Siftimage[:,s1[1]:s1[1]+s2[1]]=image2
    # Drawing Lines for SIFT points matched
    for i in range(0,len(sift_pts1)):
        cv.line(Siftimage,(int(sift_pts1[i][1]),int(sift_pts1[i][0])),
        (s1[1]+int(sift_pts2[i][1]),int(sift_pts2[i][0])),(0,0,255),thickness=2)

    return Siftimage

# Matching the corresponding interest points, between the two images
def match_sift(image1,Sift_Corners1,des1,image2,Sift_Corners2,des2):

    l1=len(Sift_Corners1)
    l2=len(Sift_Corners2)
    euclidian=np.zeros((l1,l2)) # Similarity metric for feature vectors

    for i in range(0,l1):
        for j in range(0,l2): # Normalizing Lengths to make it
        invariant to change in illumniation
```

```
            euclidian[i][j]=np.linalg.norm(des1[i]
            /np.linalg.norm(des1[i])-des2[j]/np.linalg.norm(des2[j]))
    # Computing the Euclidean distance between the 128 bit feature vectors
    sift_threshold=np.min(euclidian)*(2)
    sift_pts1=[]
    sift_pts2=[]
    # Finding similar interest points and mapping them with each other
    for i in range(0,len(euclidian)):
        val=np.min(euclidian[i])    # Similar to the SSD for Harris corner detector
        idx=np.argmin(euclidian[i])
        if(val<=sift_threshold):
            sift_pts1.append([Sift_Corners1[i][0],Sift_Corners1[i][1]])
            sift_pts2.append([Sift_Corners2[idx][0],Sift_Corners2[idx][1]])
    # Adding similar and corresponding interest points to the list

    return sift_pts1,sift_pts2

# Using the built in SIFT function from cv library
sift = cv.xfeatures2d.SIFT_create()

# Reading the two images
image1clr=cv.imread("1.jpg")
image2clr=cv.imread("2.jpg")
# Used for reducing the size of truck images, to improve computation speed.
#image1clr=cv.resize(image1clr, (0,0), fx=0.50, fy=0.50)
#image2clr=cv.resize(image2clr, (0,0), fx=0.50, fy=0.50)
# Converting from color to grayscale
image1=cv.cvtColor(image1clr,cv.COLOR_RGB2GRAY)
image2=cv.cvtColor(image2clr,cv.COLOR_RGB2GRAY)
# Finding the interest points and the descriptor vectors for every
interest point in the image
kp1,des1 = sift.detectAndCompute(image1,None)
kp2,des2 = sift.detectAndCompute(image2,None)

Sift_Corners1=[]
Sift_Corners2=[]
# Extracting the interest points for both images
for i in kp1:
    Sift_Corners1.append([i.pt[1],i.pt[0]])

for i in kp2:
    Sift_Corners2.append([i.pt[1],i.pt[0]])

# Finding the similarity metric and mapping similar interest points
sift_pts1,sift_pts2=match_sift(image1,Sift_Corners1,des1,image2,Sift_Corners2,des2)
# Drawing Lines between the corresponding interest points in the image
Siftimage=draw_lines(image1clr,sift_pts1,image2clr,sift_pts2)

cv.imwrite('SIFTimage.jpg',Siftimage)
# Writing the Output corresponding interest points in both images, with lines drawn
```

THE END