COMPUTER VISION

ECE 661

HOMEWORK 4

Tanzeel U. Rehman

Email: *trehman@purdue.edu*

1. Introduction:

This homework uses code for finding the Harris corners in a given gray image. Then we used sum of square differences (SSD) and normalized cross correlation (NCC) to find the correspondences among the corners of a given image pair. We also compared the performance with the SIFT feature and its descriptors for finding the correspondences in an image pair.

Description of Methods:

A. Harris Corner:

The Harris corner algorithm implemented as a part of this algorithm converted the RGB image to grayscale and then performed the corner detection. The corners were identified by computing the x (dx) and y gradients (dy) using the Haar kernels having a size equal to the smallest even number greater than 4*sigma, where sigma controls the scale. Given a sigma of 1.2, the Ix and Iy are 6 × 6 Haar kernels which can be given as Eq.1. These kernels can be convolved with the grayscale image to obtain the gradients. After obtaining the gradients, a *C* matrix was computed using Eq. 2 for every pixel along with its neighboring pixels contained in the 5*sigma × 5*sigma window.

$$dx(1.2) = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \quad \text{(Eq.1)}$$

$$dy(1.2) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$C = \begin{bmatrix} \sum dx^2 & \sum dxdy \\ \sum dxdy & \sum dy^2 \end{bmatrix} \quad \text{(Eq.2)}$$

Where, summations are performed over 5*sigma × 5*sigma window. The *C* is a full rank matrix for true corner. Now, instead of decomposing this matrix, we can compute the Harris cornerness response using Eq. 3.

$$R = \det(C) - k(trace(C))\text{^}2 \quad \text{(Eq.3)}$$

$$\det(C) = \sum dx^2 \sum dy^2 - (\sum dxdy)^2 \quad \text{(Eq.4)}$$

$$\text{trace}(C) = \sum dx^2 + \sum dy^2 \quad \text{(Eq.5)}$$

$$R >= 0 \quad \text{(Eq.6)}$$

Where the *k* is an empirical constant which was set to 0.04 by default. The Harris corner can be given by Eq.6; however, this can lead to the clusters of corners which are very close to each other. Therefore, noise suppression was performed by defining the neighborhood having a size of 25 x 25 pixels and only maximum corners was picked from this window.

The correspondences between two image pairs was calculated by finding a corner in image 2 which is closest to the corner in image 1. The closeness measure for this homework was SSD and NCC, which can be computed using Eq. 7 and 8.

$$SSD = \sum_x \sum_y (f_1(x,y) - f_2(x,y))^2 \quad \text{(Eq.7)}$$

$$NCC = \frac{\sum_x \sum_y ((f_1(x,y) - m1)(f_2(x,y) - m2)}{\sqrt{\sum_x \sum_y ((f_1(x,y) - m1)^2 (f_2(x,y) - m2)^2}} \quad \text{(Eq.8)}$$

B. SIFT:

For SIFT, follow the steps below:

1) Construct the scale space

2) Perform the difference of Gaussian (DOG) approximation for finding the interest points.

3) Find the maxima and minima in DOG as the keypoints.

4) Remove the edges from keypoints and threshold based on the contrast

5) Assign orientation and generate the sift features.

1.1. Assumptions and Notes:

a) The value of sigma for computing the Harris corner in the code is configurable.

b) The gradients are measures using the Haar kernels at the different scales controlled by the sigma.

c) In this assignment, we computed the sum over 5*sigma for calculating the C matrix.

d) The noise suppression was performed by finding the maxima in the window and rejecting all other possible corners nearby. This helped to find the strongest corner in the neighborhood window. The half window size for every image pair is set to possible value of 15 ×15 otherwise mentioned in the caption of each image. The k-value for the Harris corner is set to 0.04 otherwise mentioned in the caption of each image in this report. The window size and k-value are configurable.

e) The correspondences were computed using SSD and NCC with a window size of 25 ×25, but this can be configured.

f) The distance in case of SSD and NCC was normalized and a rejection threshold was used for identifying the good matches. The distance was measured, and normalized distance was used to find the closest point in image 2 to that of image 1. The rejection threshold was then used to find if the point qualifies to be a good match.

g) An SSD threshold of 0.8 and 0.88 was used for pair 1, 3 and pair 2 images, respectively. For my own 1st and 2nd pair of images, the NCC threshold was 0.85 and 0.9, respectively.

h) An NCC threshold of 0.9 and 0.88 was used for pair 1, 2 and pair 3 images, respectively. For my own 1st and 2nd pair of images, the SSD threshold was 0.8 and 0.65, respectively.

i) My own 1st image pair was relatively bigger in size and therefore took very long computation time as the code is not currently fully vectorized. Therefore, I cropped the 1st image pair and also resized it for finding the Harris corners. For the SIFT, this image pair was only cropped and not resized.

j) In addition to the different sigma values, we also used the different k_value and noise suppression window with my own 1st image pair. The results are added in this report.

k) Since, my own 1st image (captured with cell phone camera) was relatively big and took lot of computation time, therefore, for my 2nd image pair, I used the images from last year which has relatively small size. This helped to reduce the processing time.

2. Task 1:

2.1. Task 1.1 (Harris corners, SSD and NCC):

Fig 1: Harris corners from images in pair 1 at sigma = 0.6; k = 0.04, noise_window = 15.

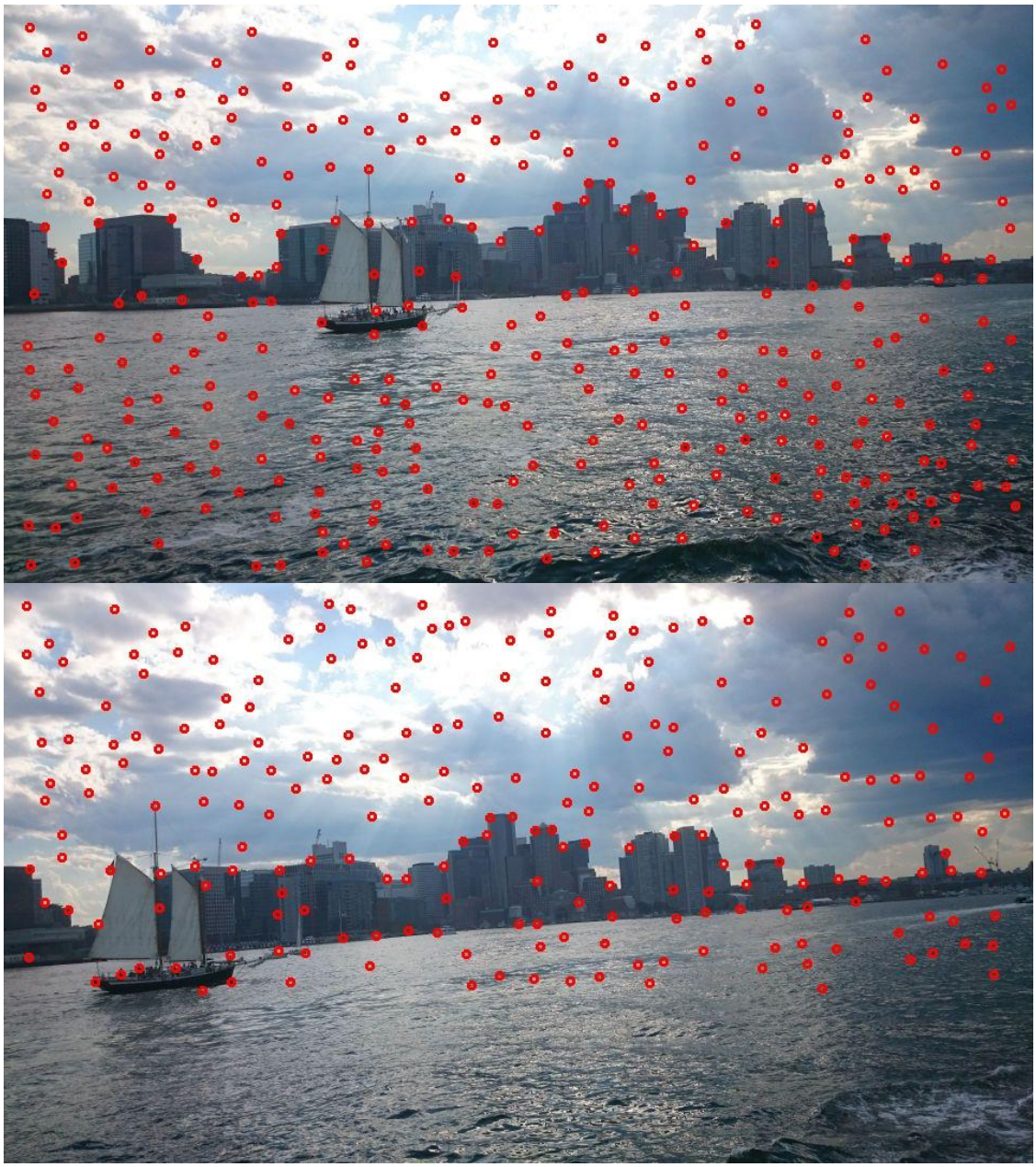

Fig 2: Harris corners from images in pair 1 at sigma = 1.2; k = 0.04, noise_window = 15.

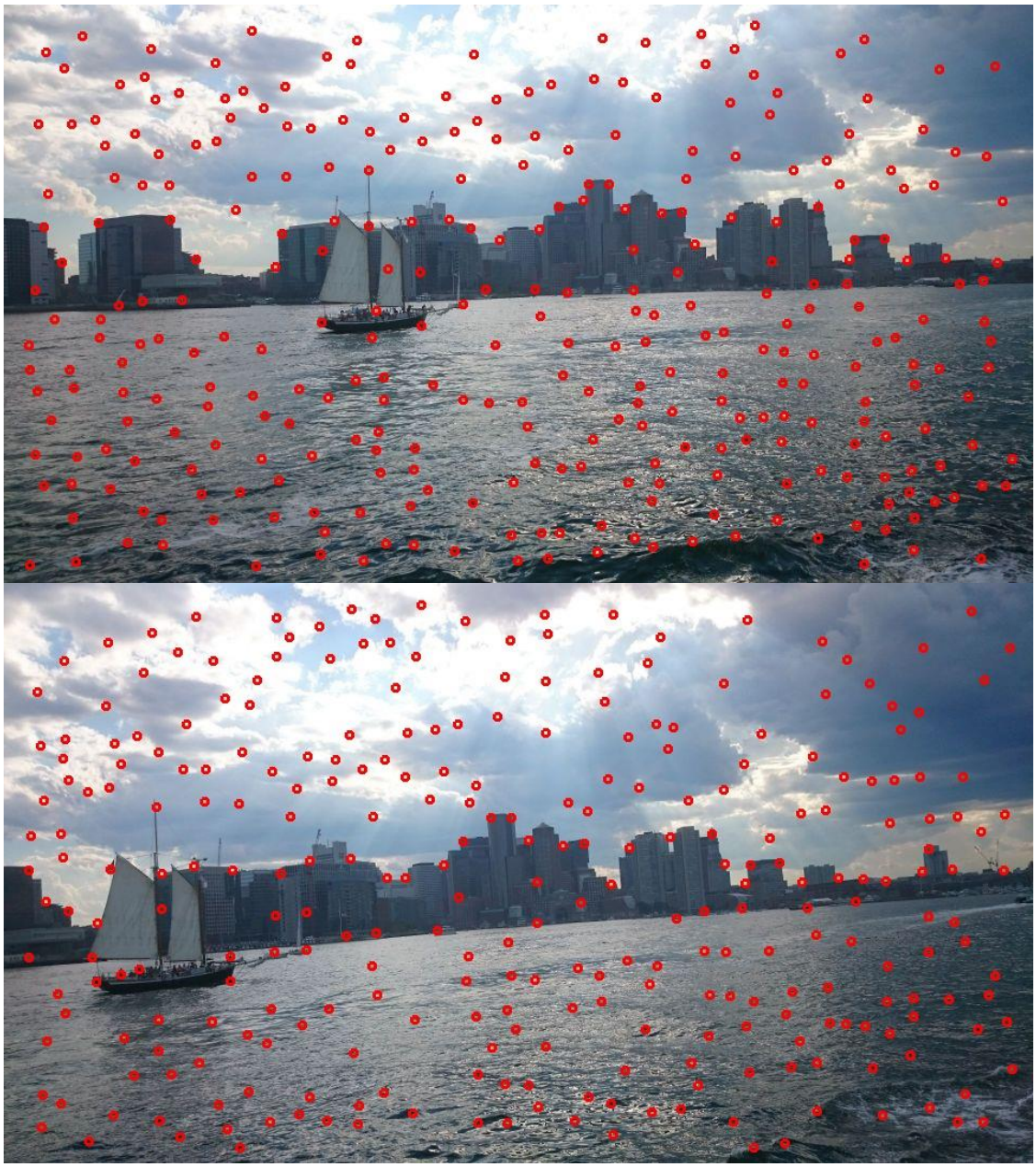

Fig 3: Harris corners from images in pair 1 at sigma = 1.8; k = 0.04, noise_window = 15.

Fig 4: Harris corners from images in pair 1 at sigma = 2.4; k = 0.04, noise_window = 15.

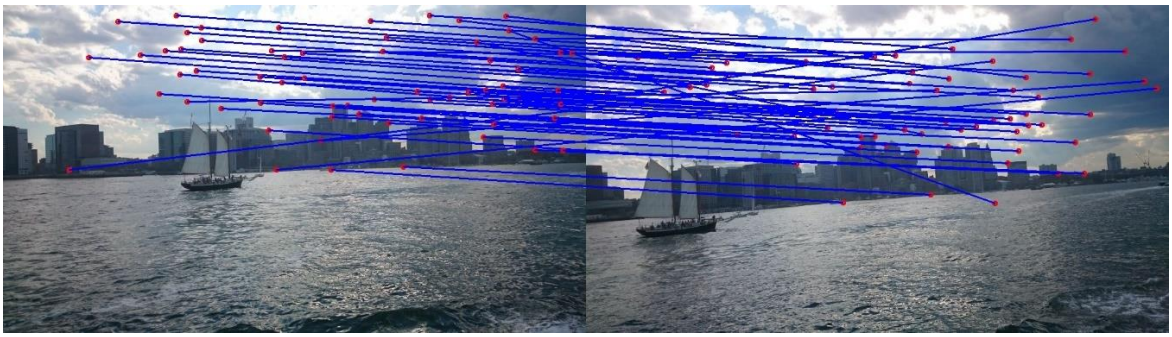

Fig 5: SSD on Harris corners from images in pair 1 at sigma = 0.6; k = 0.04, noise_win= 15.

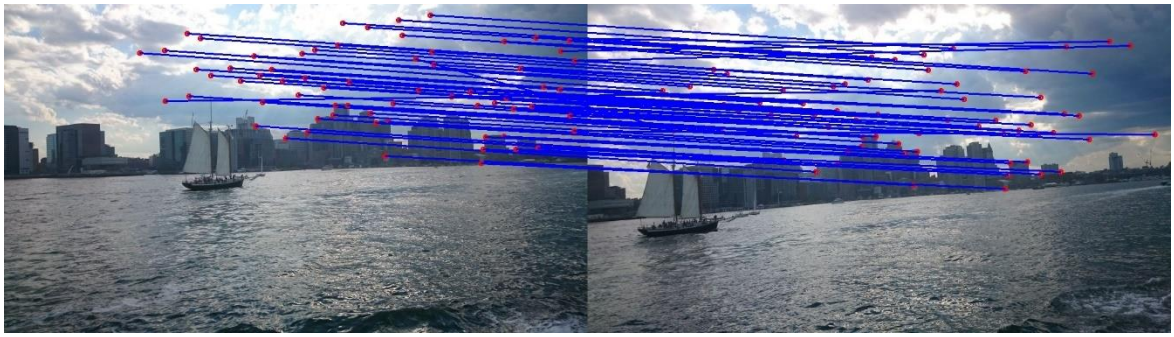

Fig 6: SSD on Harris corners from images in pair 1 at sigma = 1.2; k = 0.04, noise_win= 15.

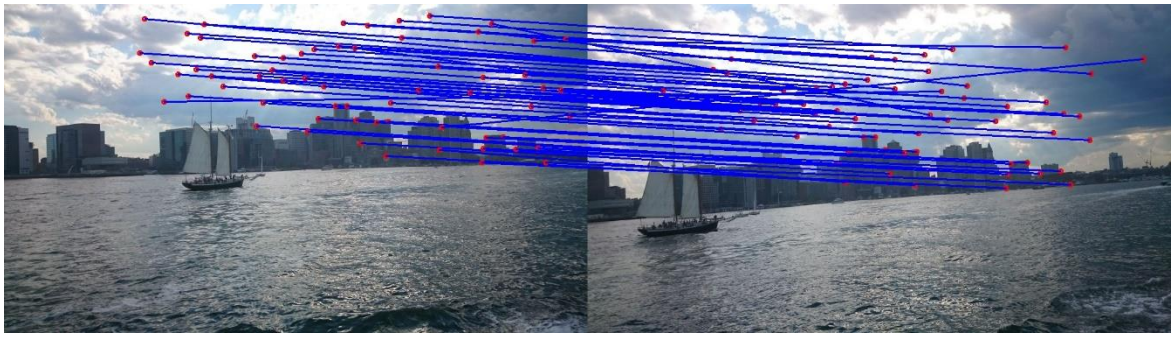

Fig 7: SSD on Harris corners from images in pair 1 at sigma = 1.8; k = 0.04, noise_win= 15.

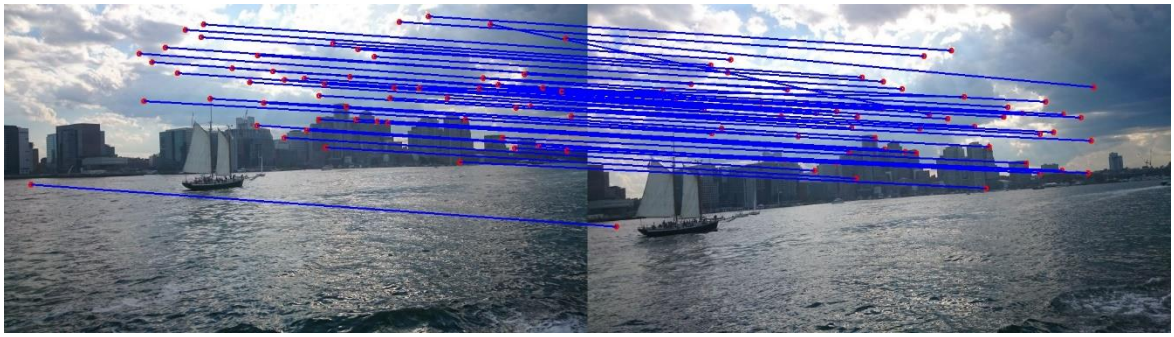

Fig 8: SSD on Harris corners from images in pair 1 at sigma = 2.4; k = 0.04, noise_win= 15.

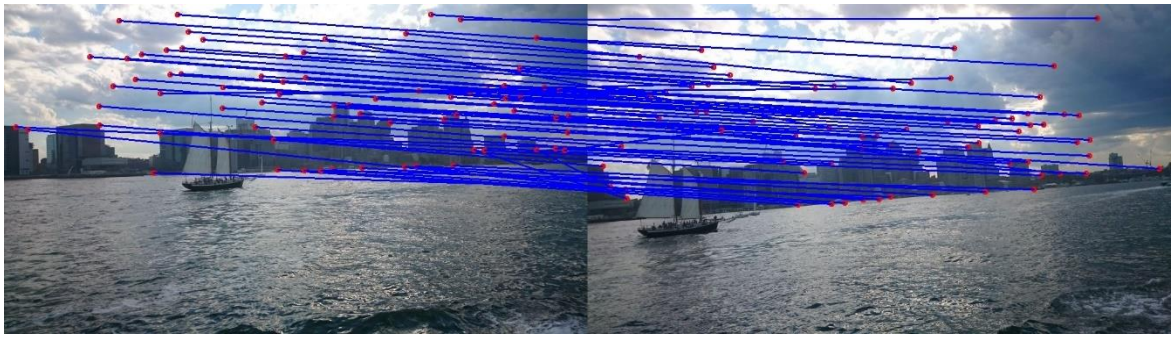

Fig 9: NCC on Harris corners from images in pair 1 at sigma = 0.6;k = 0.04, noise_win= 15.

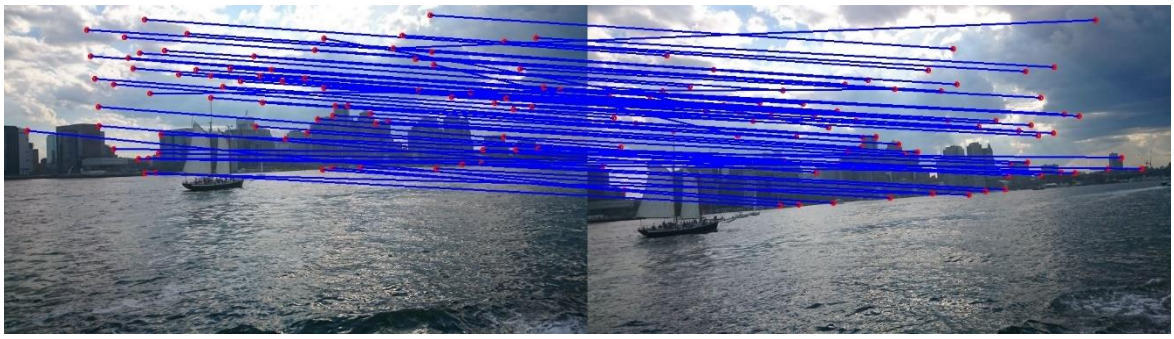

Fig 10: NCC on Harris corners from images in pair 1 at sigma =1.2;k = 0.04,noise_win= 15.

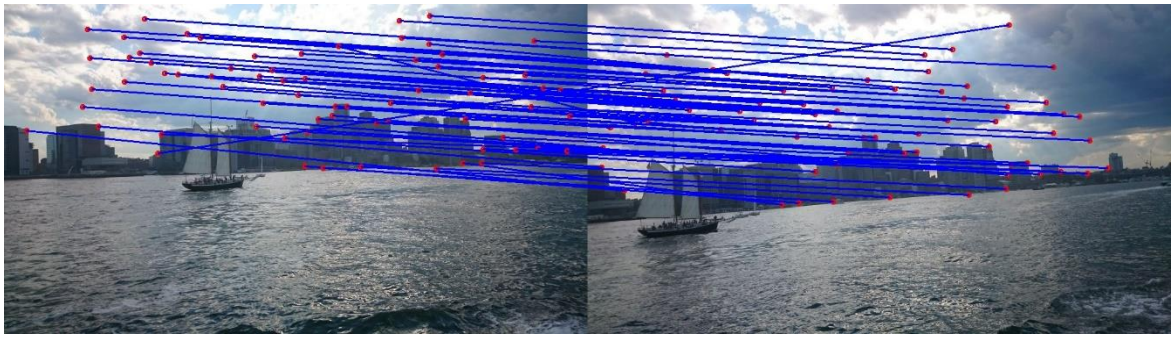

Fig 11: NCC on Harris corners from images in pair 1 at sigma =1.8;k = 0.04,noise_win= 15.

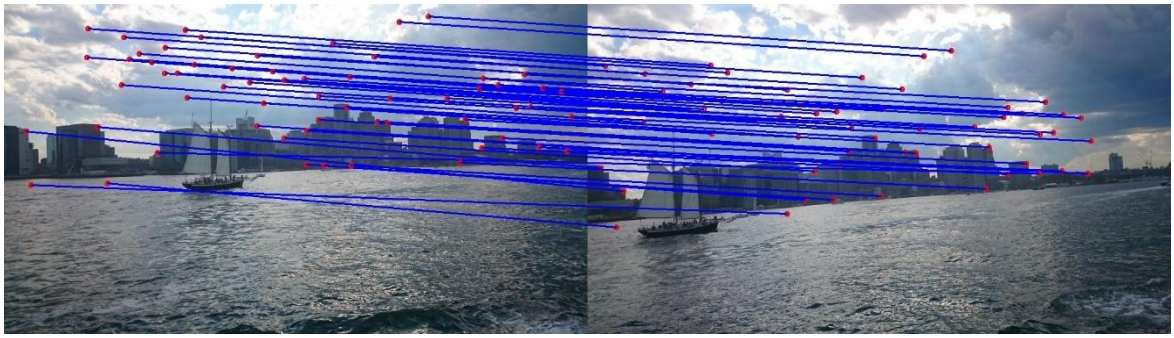

Fig 12: NCC on Harris corners from images in pair 1 at sigma =2.4;k = 0.04,noise_win= 15.

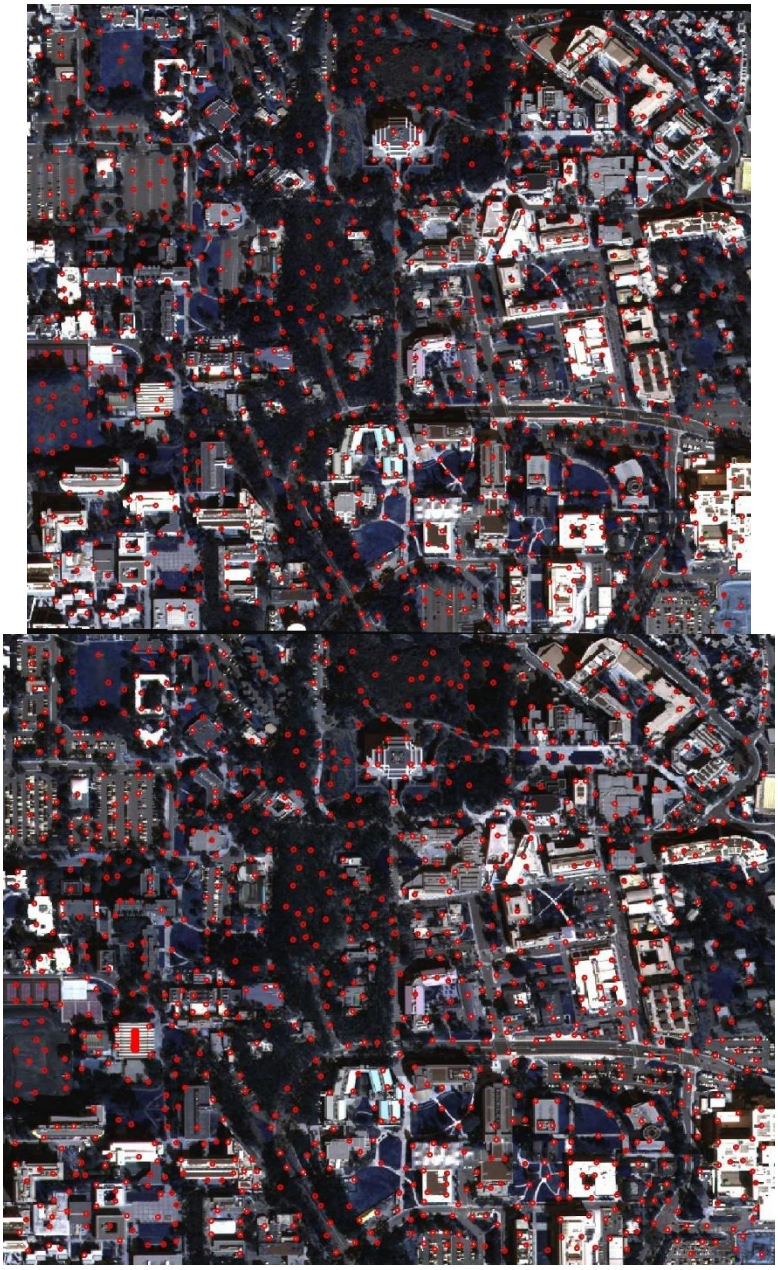

Fig 13: Harris corners from images in pair 2 at sigma = 0.6; k = 0.04, noise_window = 15.

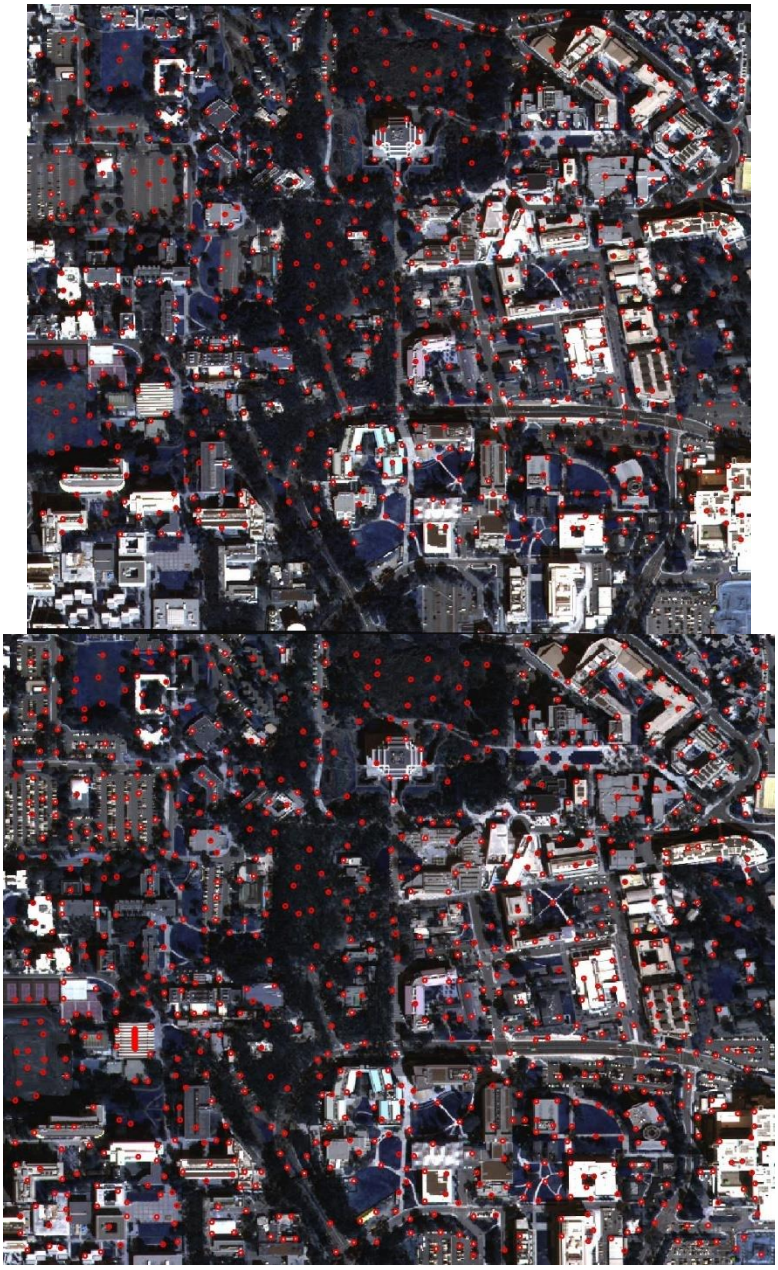

Fig 14: Harris corners from images in pair 2 at sigma = 1.2; k = 0.04, noise_window = 15.

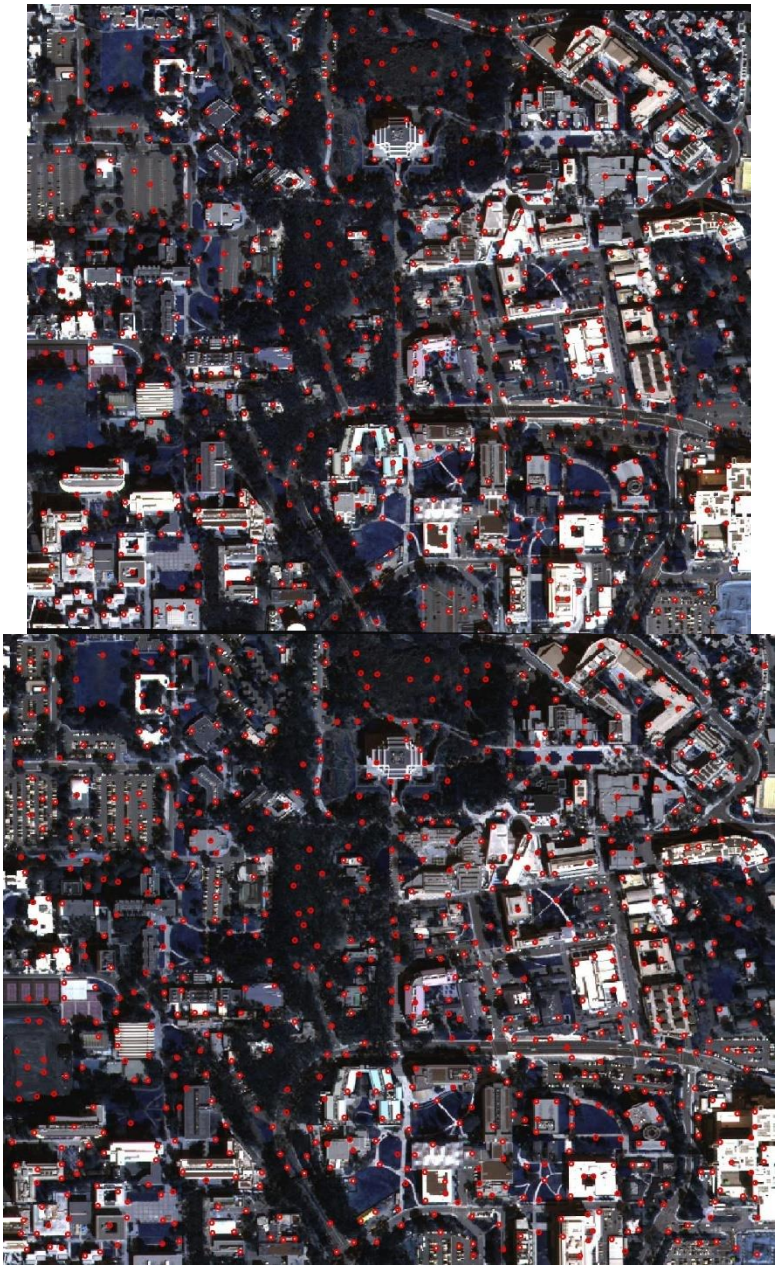

Fig 15: Harris corners from images in pair 2 at sigma = 1.8; k = 0.04, noise_window = 15.

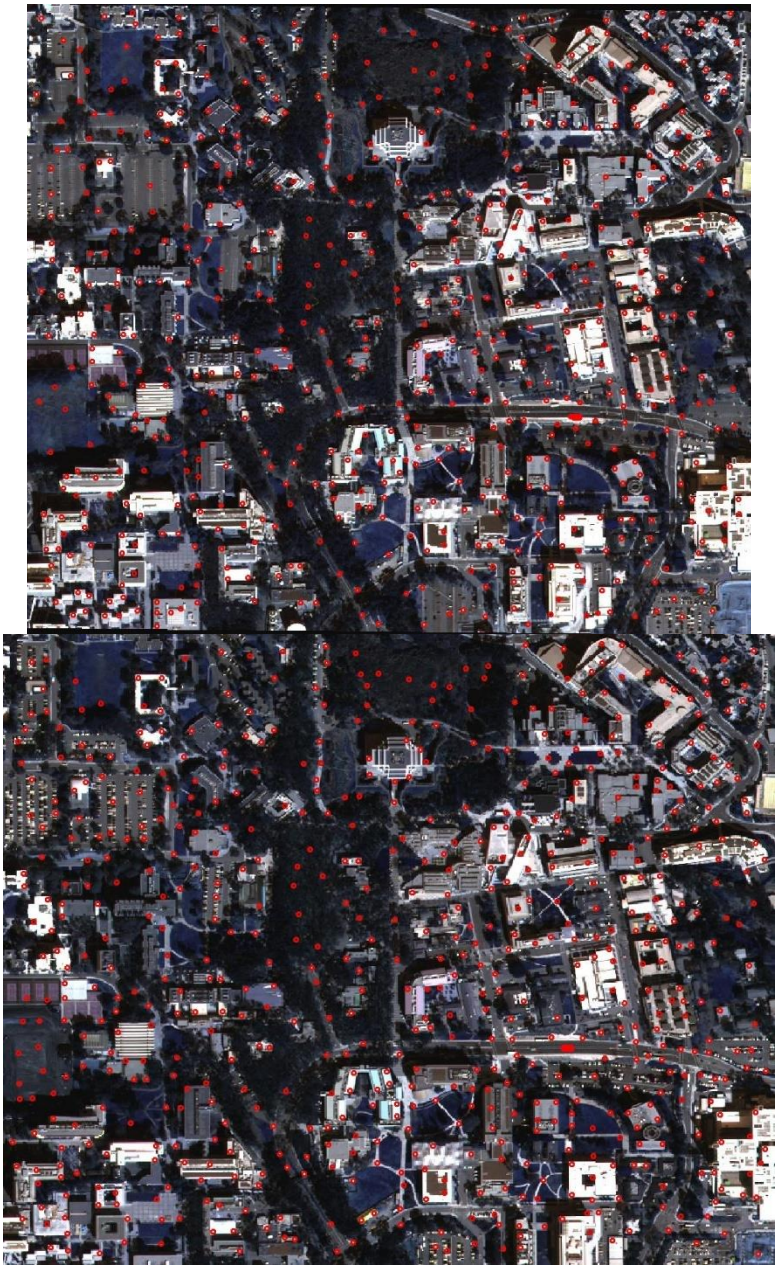

Fig 16: Harris corners from images in pair 2 at sigma = 2.4; k = 0.04, noise_window = 15.

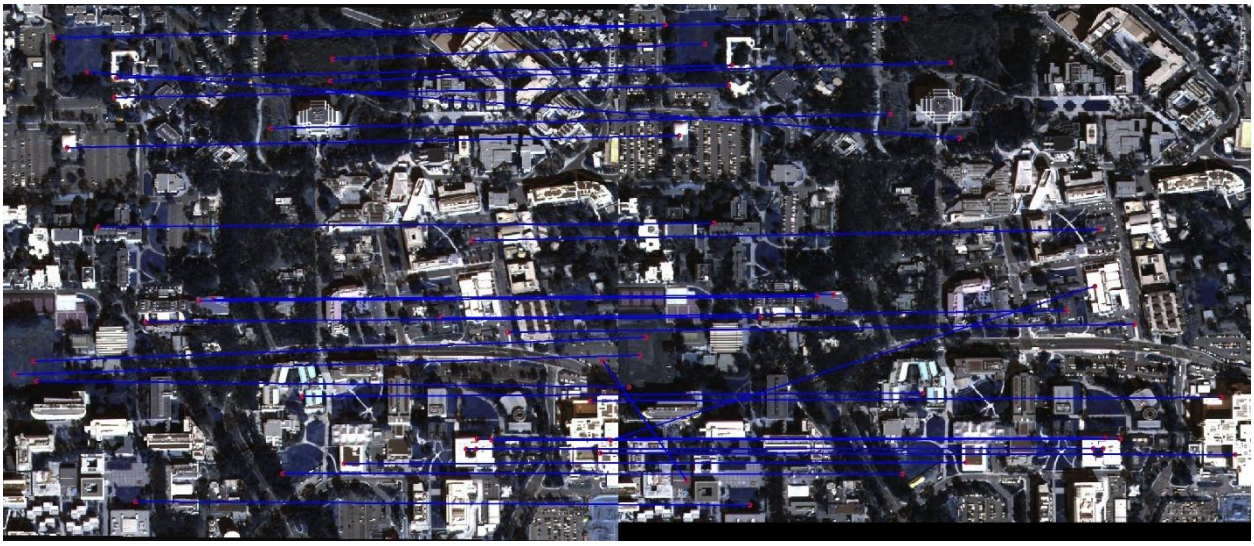

Fig 17: SSD on Harris corners from images in pair 2 at sigma =0.6;k = 0.04,noise_win= 15.

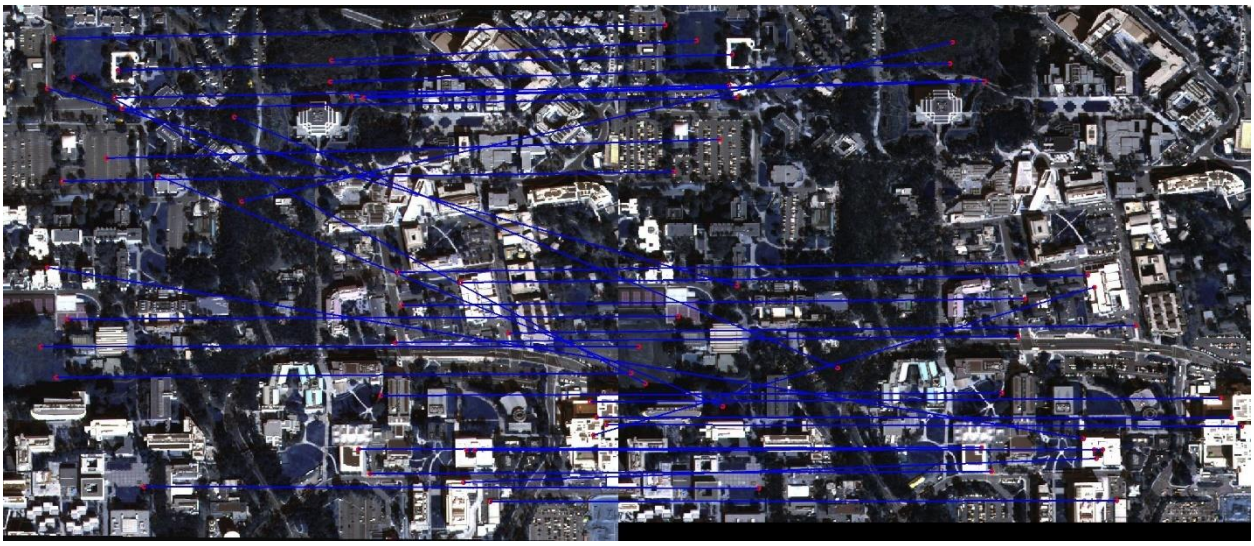

Fig 18: SSD on Harris corners from images in pair 2 at sigma =1.2;k = 0.04,noise_win= 15.

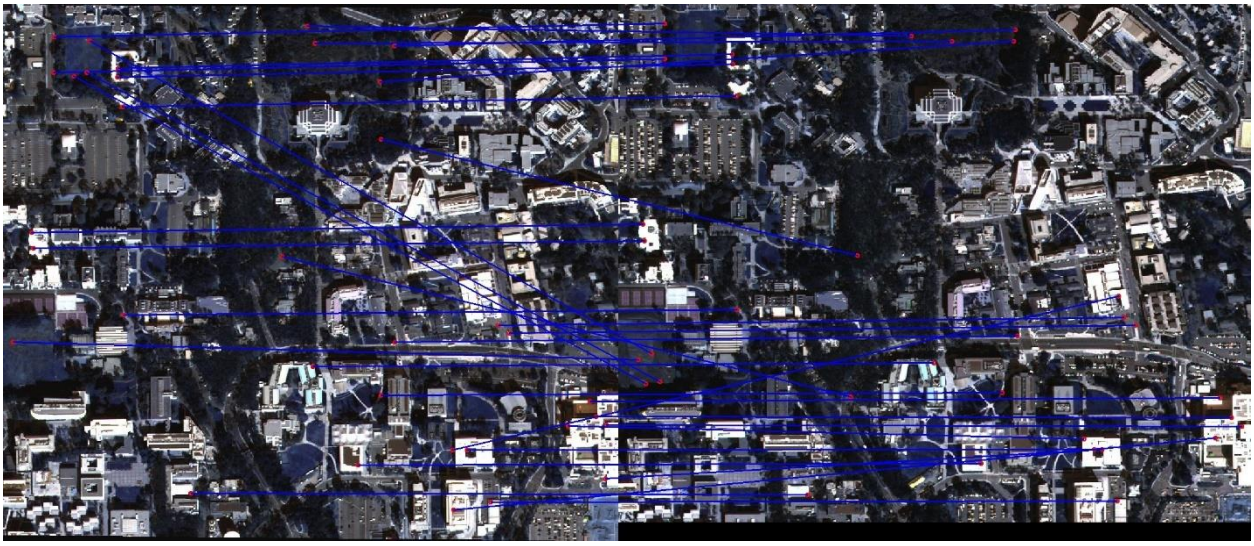

Fig 19: SSD on Harris corners from images in pair 2 at sigma =1.8;k = 0.04,noise_win= 15.

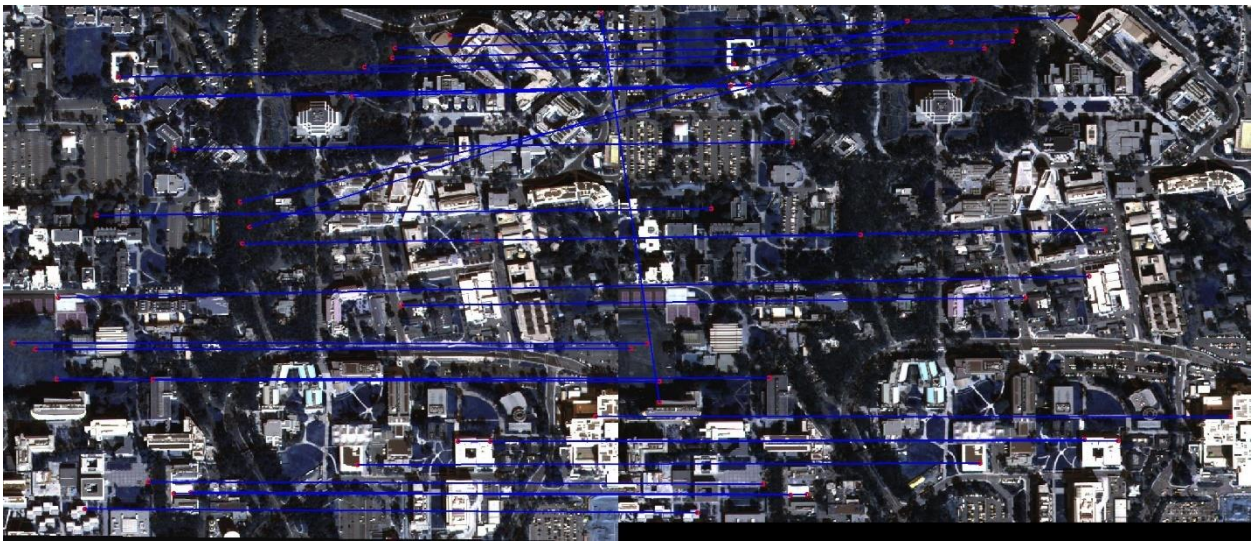

Fig 20: SSD on Harris corners from images in pair 2 at sigma =2.4;k = 0.04,noise_win= 15.

Fig 21: NCC on Harris corners from images in pair 2 at sigma =0.6;k = 0.04,noise_win= 15.

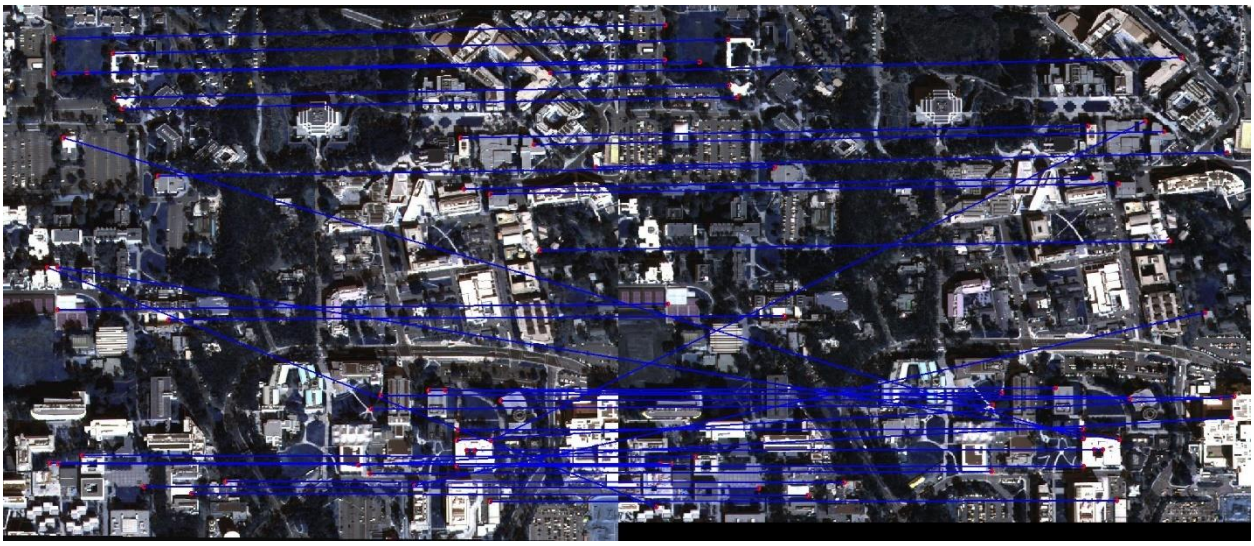

Fig 22: NCC on Harris corners from images in pair 2 at sigma =1.2;k = 0.04,noise_win= 15.

Fig 23: NCC on Harris corners from images in pair 2 at sigma =1.8;k = 0.04,noise_win= 15.

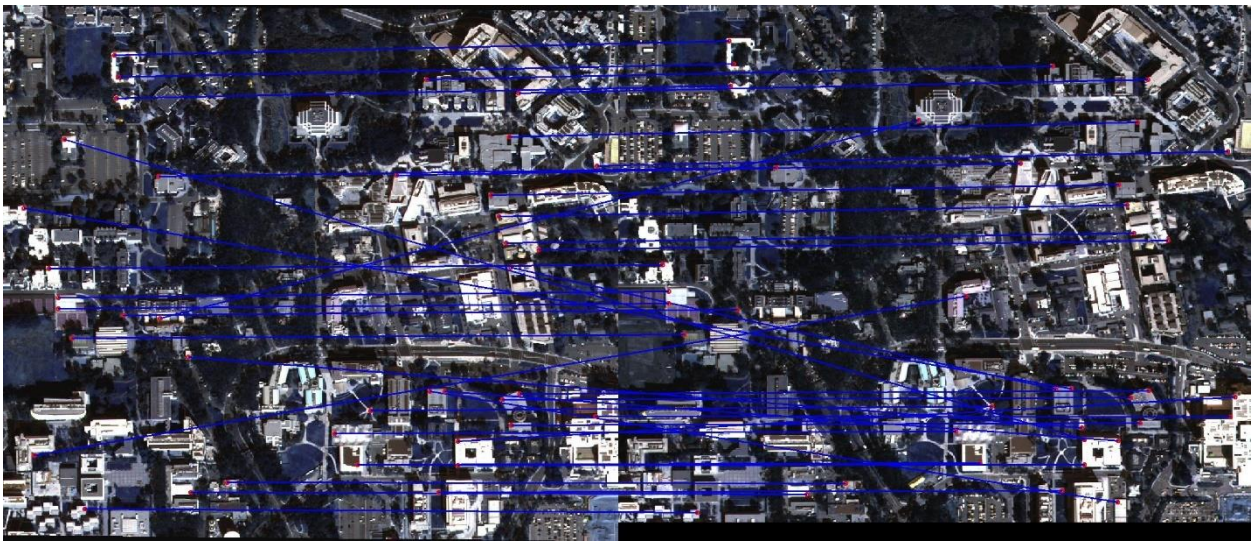

Fig 24: NCC on Harris corners from images in pair 2 at sigma =2.4;k = 0.04,noise_win= 15.

Fig 25: Harris corners from images in pair 3 at sigma = 0.6; k = 0.04, noise_window = 15.

Fig 26: Harris corners from images in pair 3 at sigma = 1.2; k = 0.04, noise_window = 15.

Fig 27: Harris corners from images in pair 3 at sigma = 1.8; k = 0.04, noise_window = 15.

Fig 28: Harris corners from images in pair 3 at sigma = 2.4; k = 0.04, noise_window = 15.

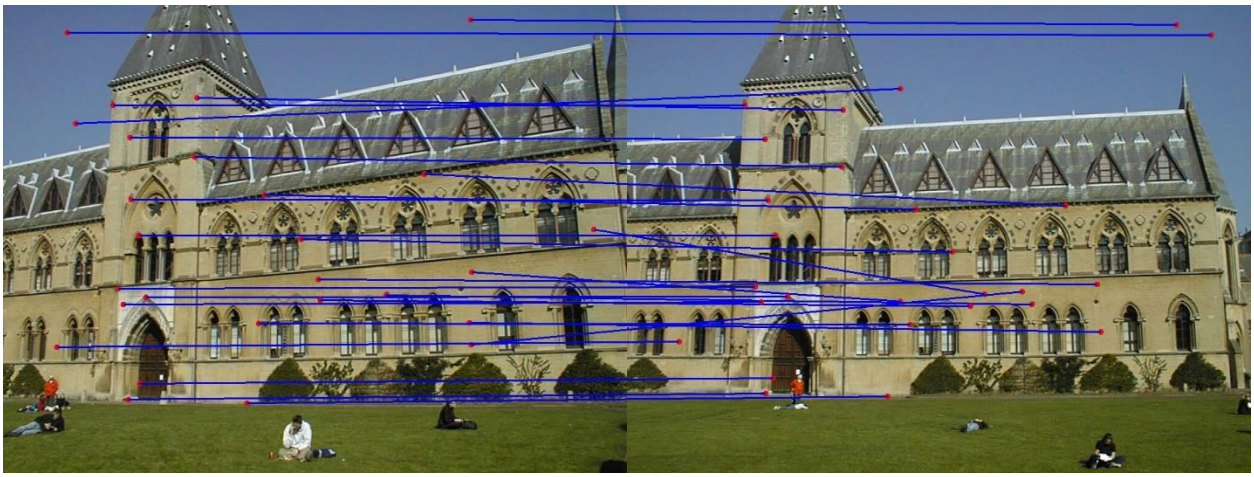

Fig 29: SSD on Harris corners from images in pair 3 at sigma =0.6;k = 0.04,noise_win= 15.

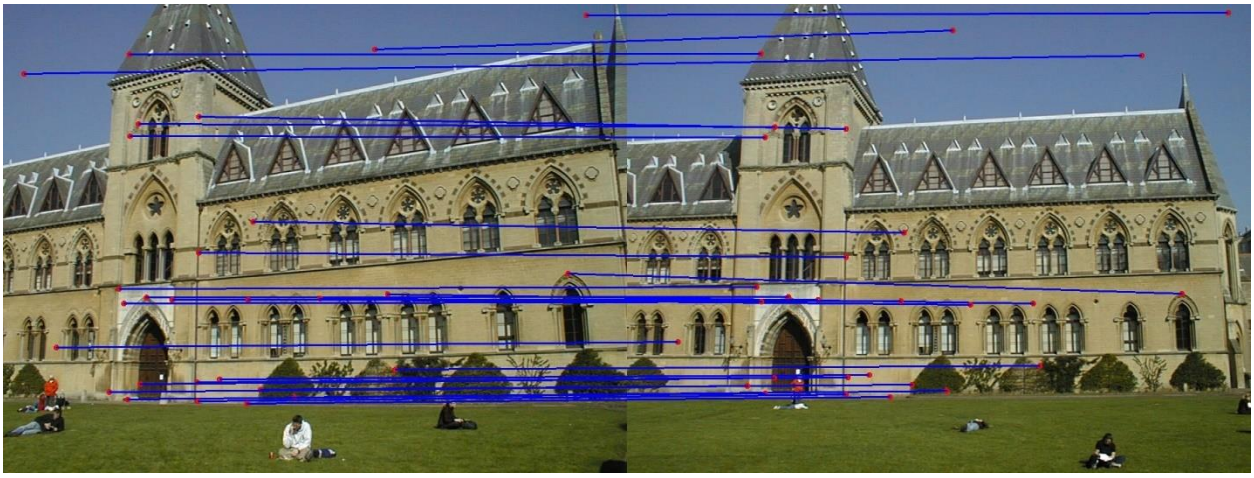

Fig 30: SSD on Harris corners from images in pair 3 at sigma =1.2;k = 0.04,noise_win= 15.

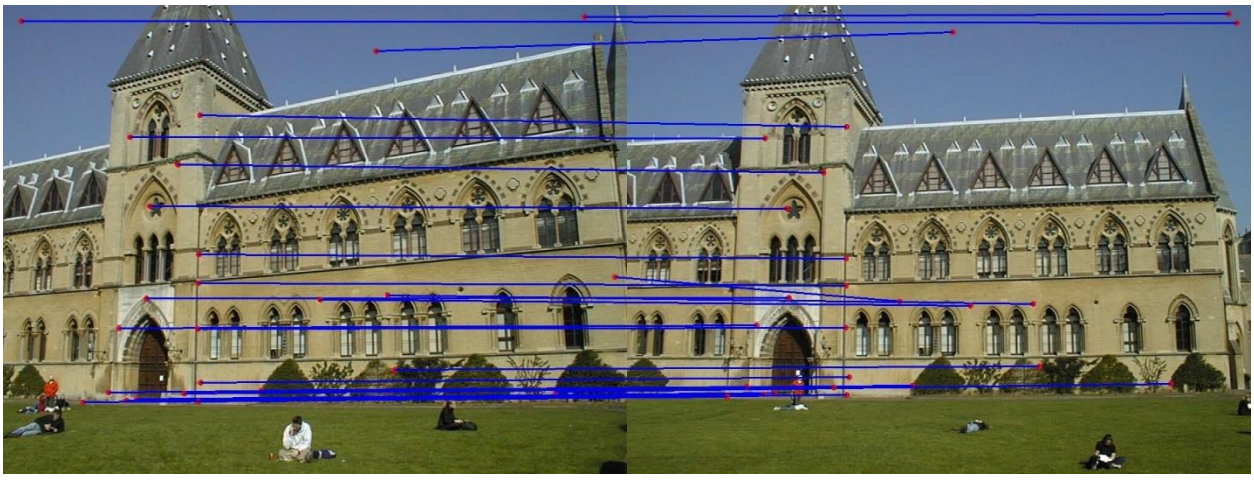

Fig 31: SSD on Harris corners from images in pair 3 at sigma =1.8;k = 0.04,noise_win= 15.

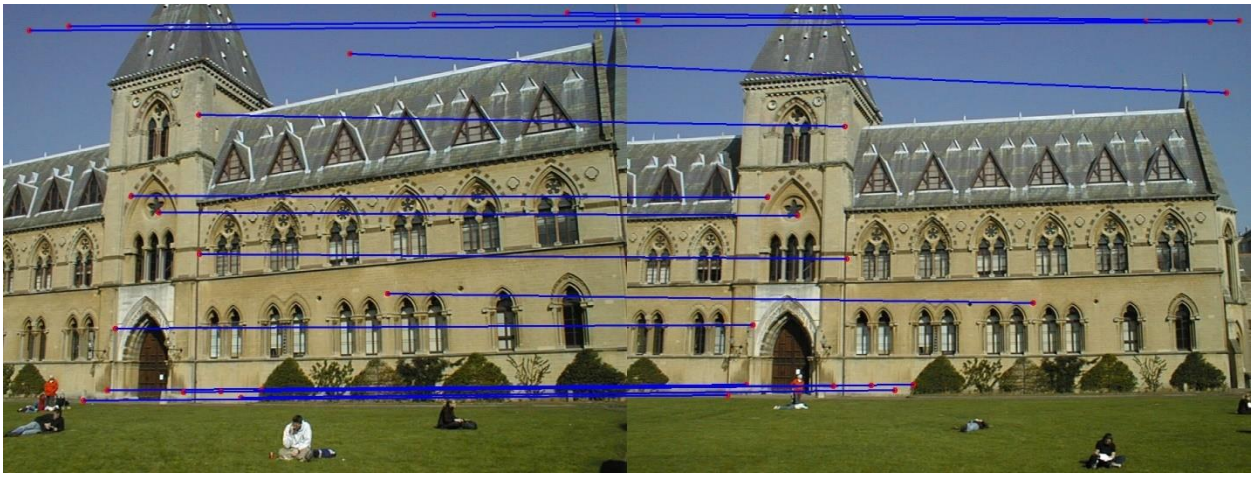

Fig 32: SSD on Harris corners from images in pair 3 at sigma =2.4;k = 0.04,noise_win= 15.

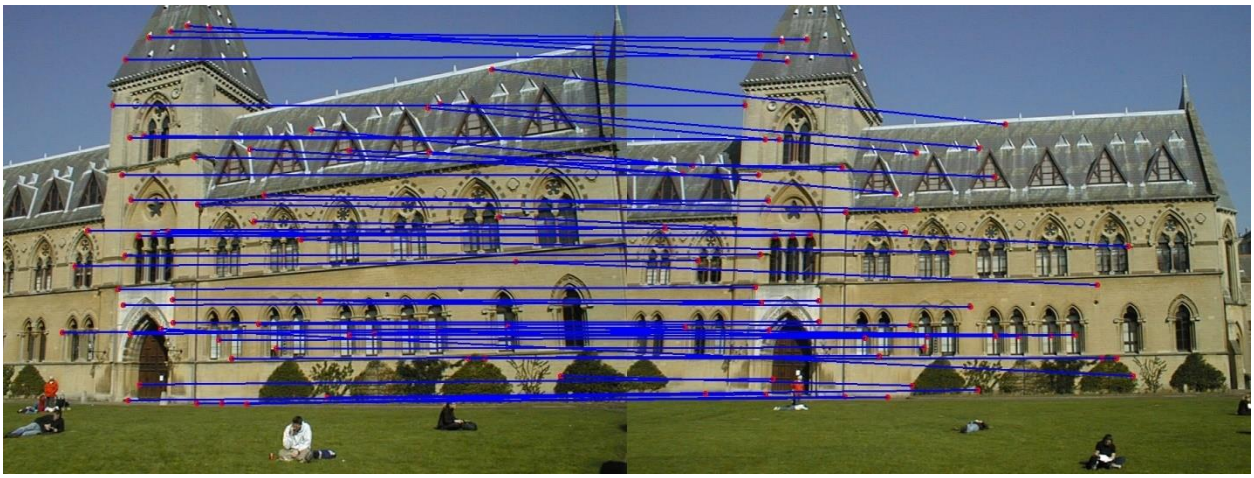

Fig 33: NCC on Harris corners from images in pair 3 at sigma =0.6;k = 0.04,noise_win= 15.

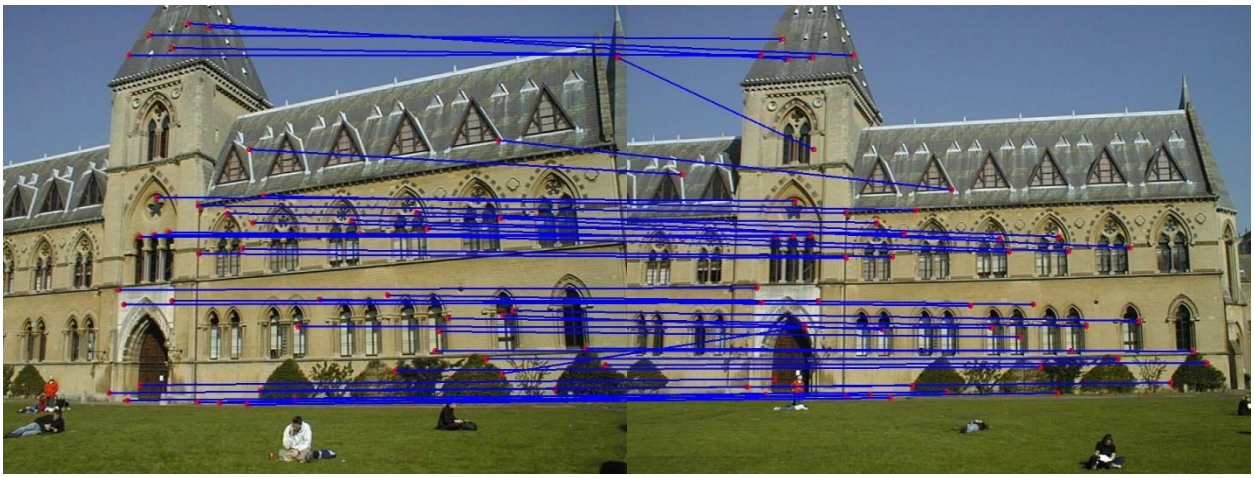

Fig 34: NCC on Harris corners from images in pair 3 at sigma =1.2;k = 0.04,noise_win= 15.

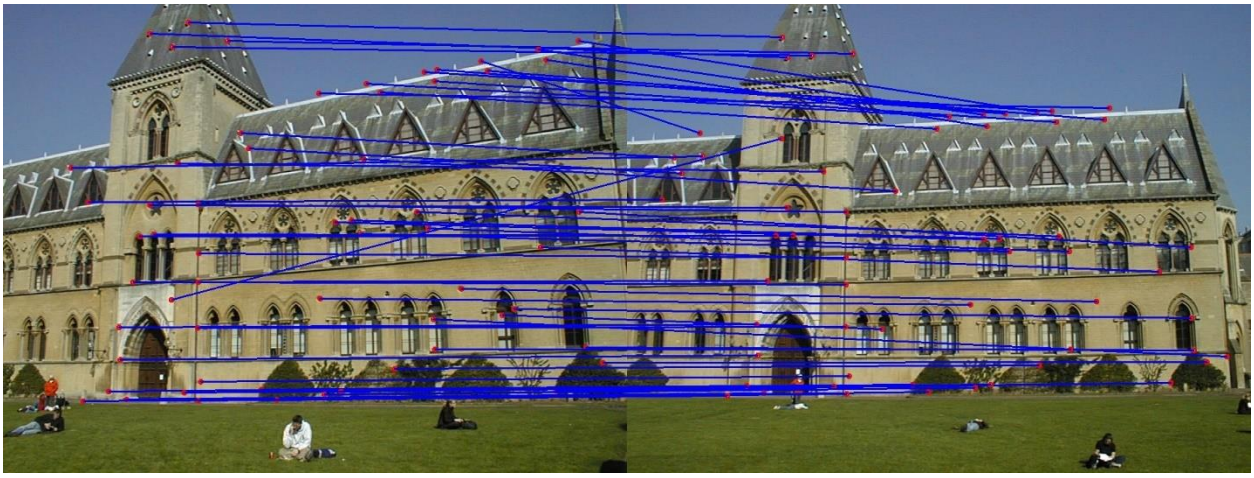

Fig 35: NCC on Harris corners from images in pair 3 at sigma =1.8;k = 0.04,noise_win= 15.

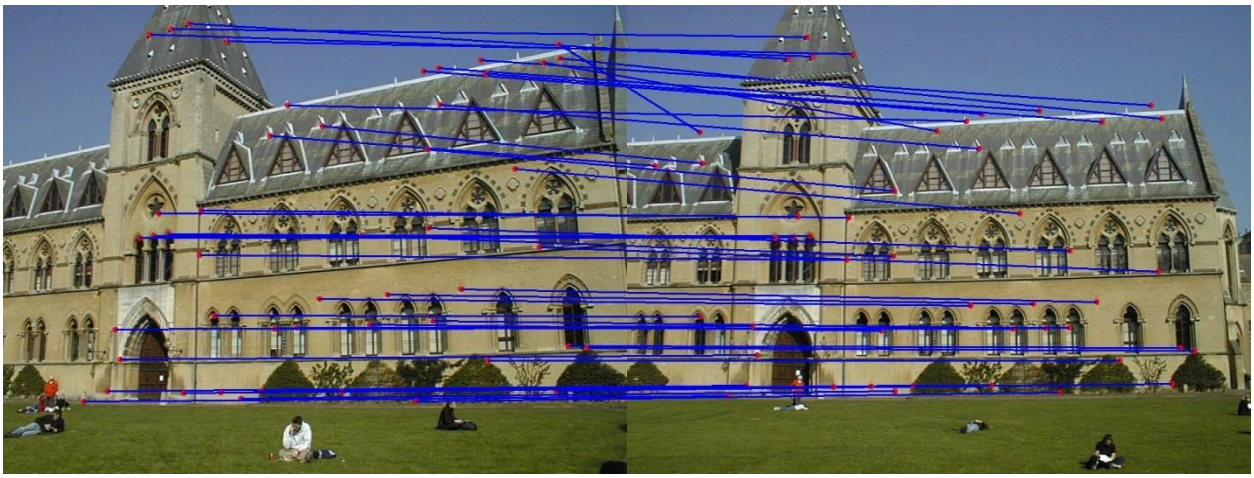

Fig 36: NCC on Harris corners from images in pair 3 at sigma =2.4;k = 0.04,noise_win= 15.

2.2. Task 1.1 (SIFT features and correspondences):

Fig 1: SIFT features from images in pair 1 with contrast thresh =0.1 & max features =5000.

Fig 2: Corresponding SIFT features from images in pair 1 with contrast thresh =0.1 & max features =5000 and brute force matching.

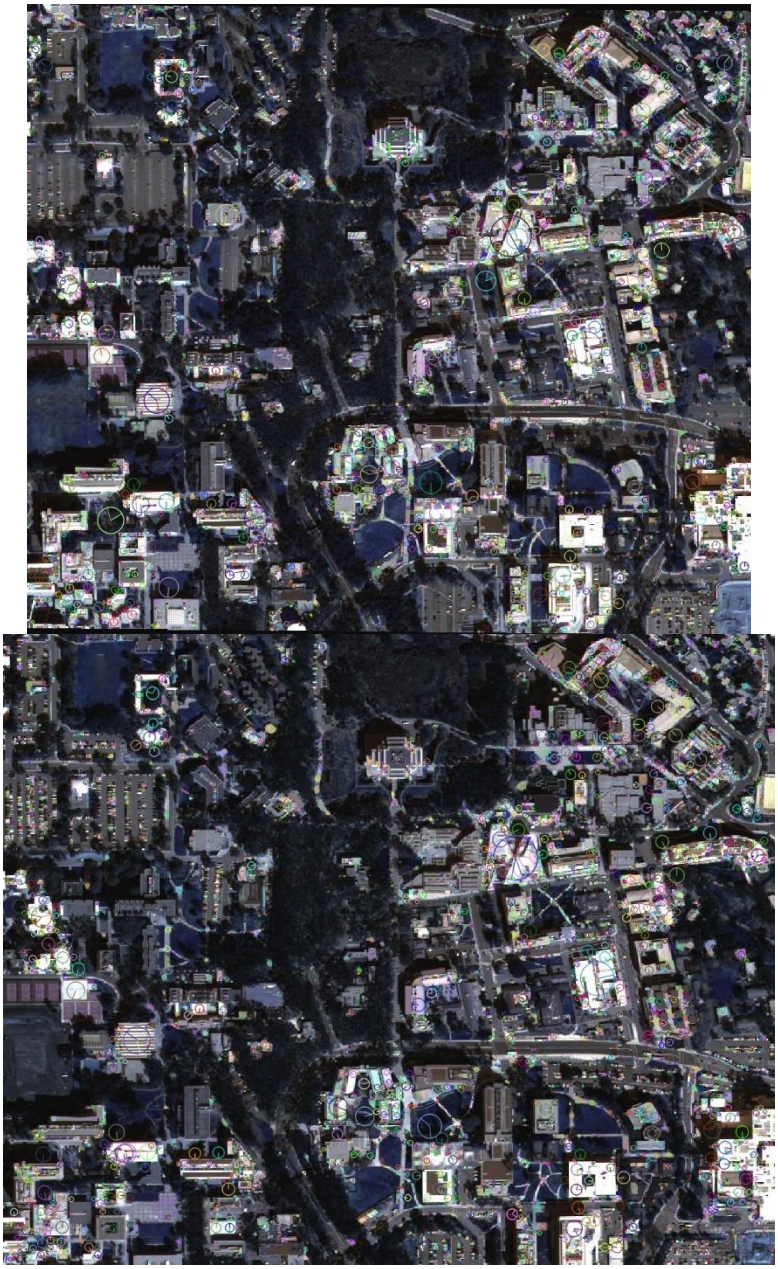

Fig 3: SIFT features from images in pair 2 with contrast thresh =0.1 & max features =5000.

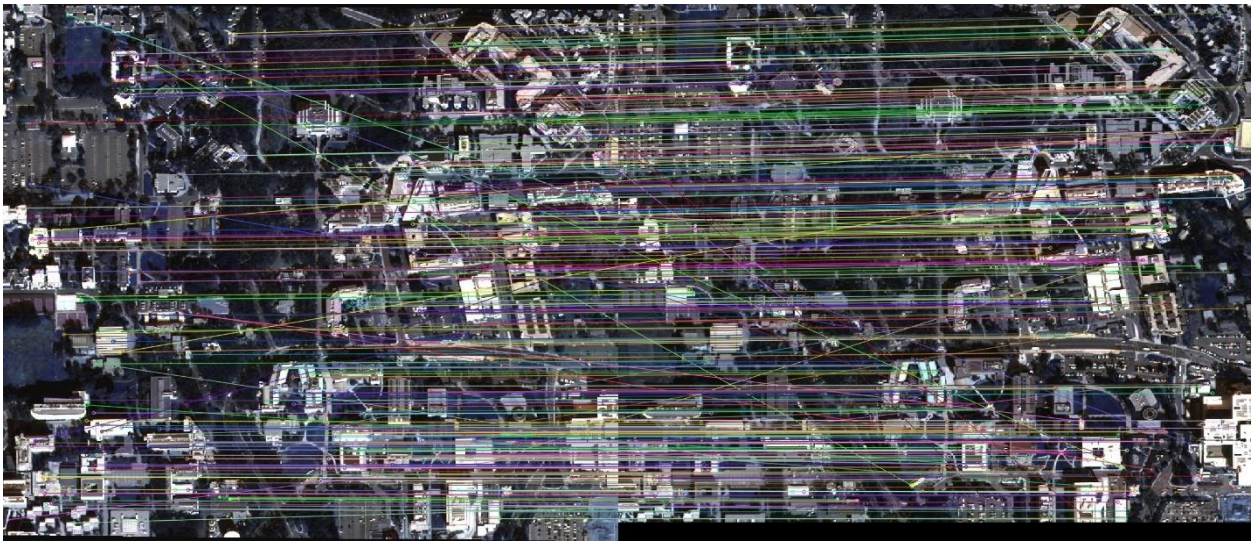

Fig 4: Corresponding SIFT features from images in pair 1 with contrast thresh =0.1 & max features =5000 and brute force matching.

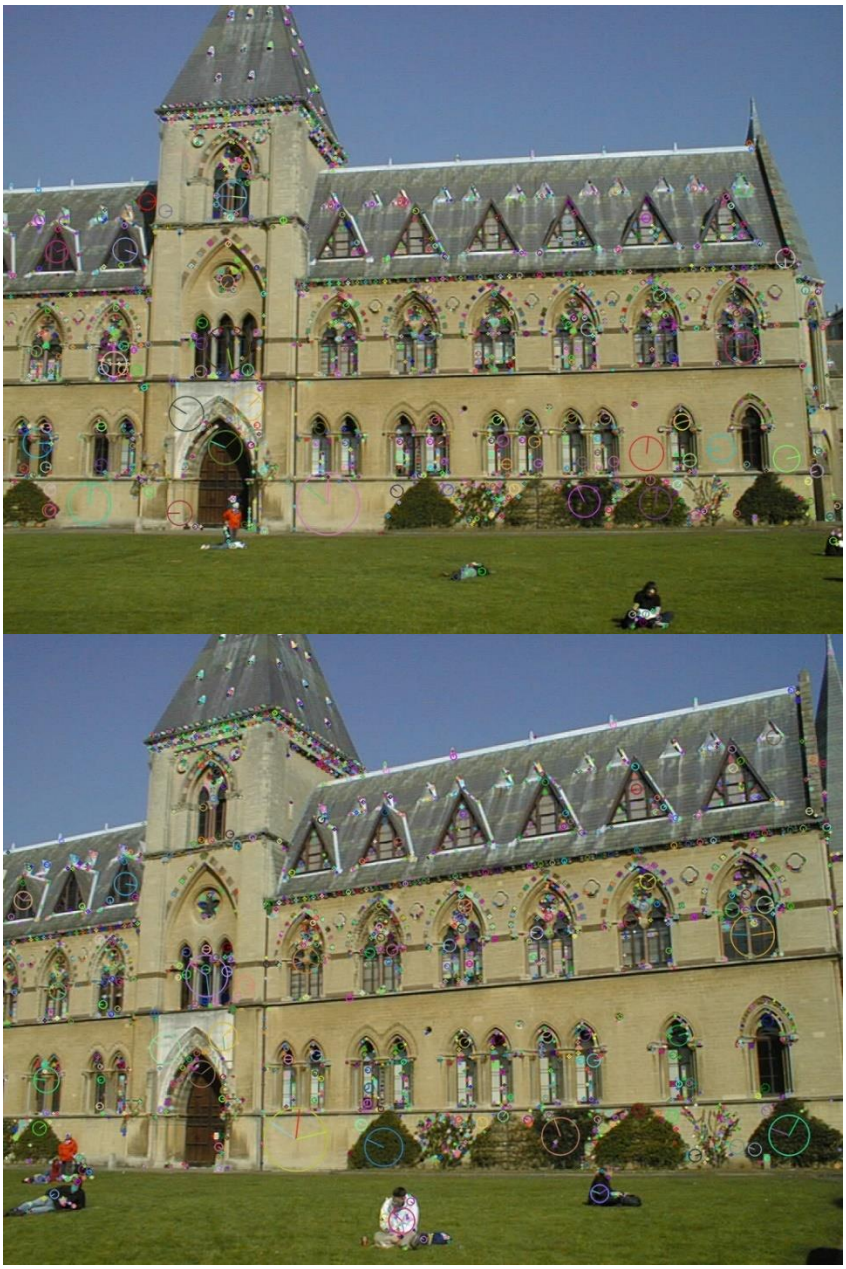

Fig 5: SIFT features from images in pair 3 with contrast thresh =0.1 & max features =5000.

Fig 6: Corresponding SIFT features from images in pair 1 with contrast thresh =0.1 & max features =5000 and brute force matching.

3. Task 2:

3.1. Input Images:

Fig 1: 1st image in my 1st pair

Fig 2: 2nd image in my 1st pair

Fig 2: 1st image in my 2nd pair

Fig 3: 2nd image in my 2nd pair

3.2. Task 2.1 (Harris corners, SSD and NCC):

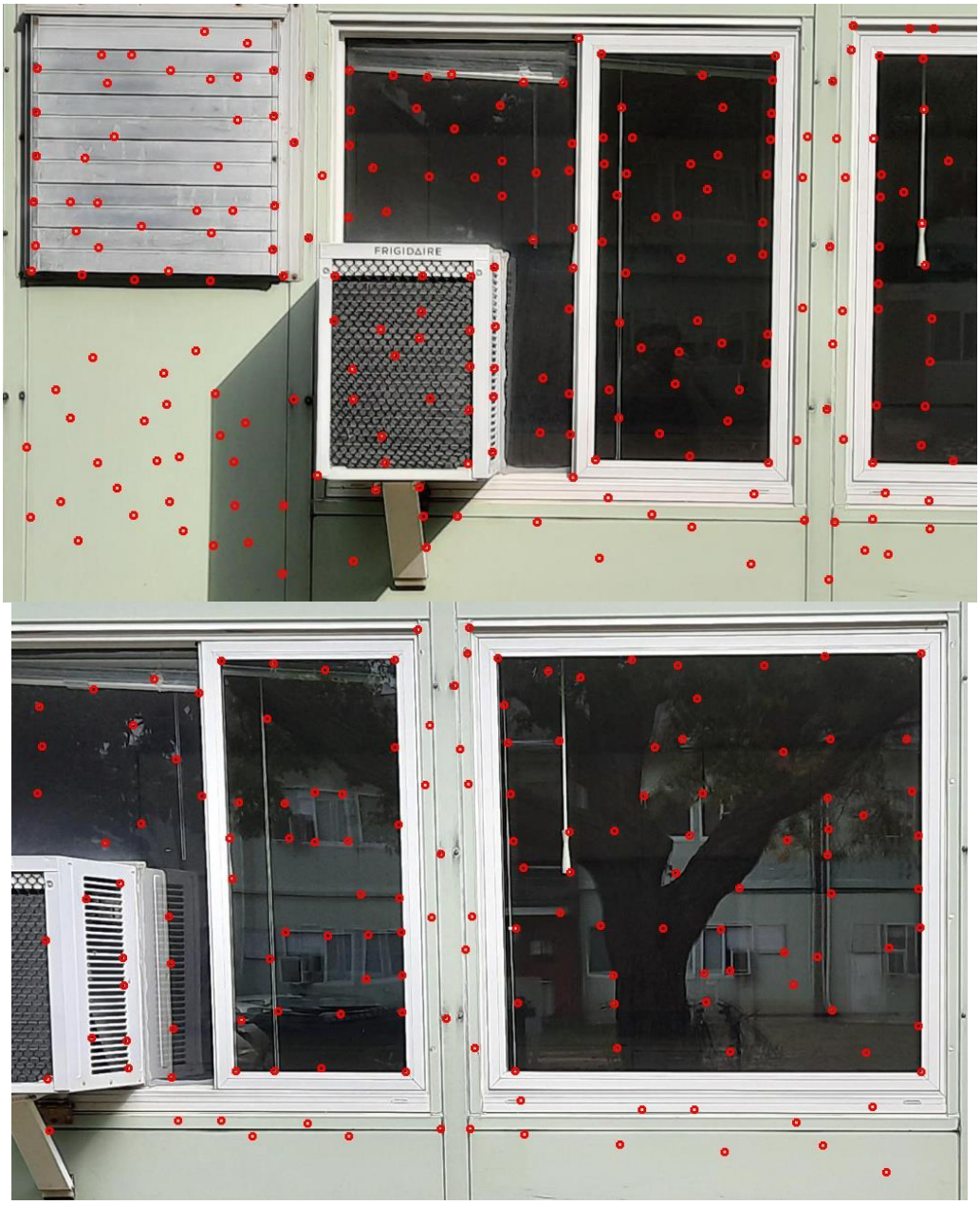

Fig 1: Harris corners from images in my pair 1 at sigma = 0.6; k = 0.06, noise_window = 21.

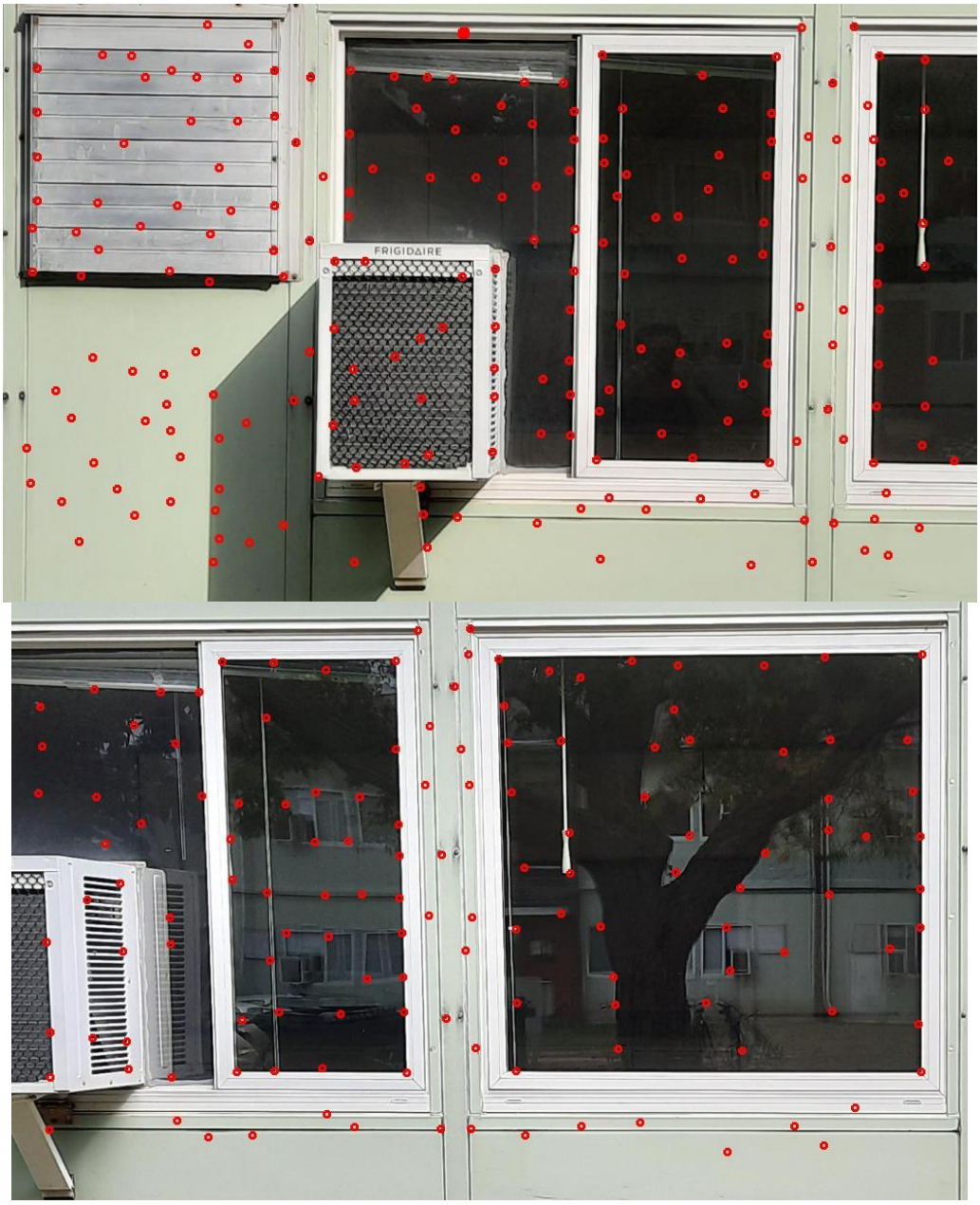

Fig 2: Harris corners from images in my pair 1 at sigma = 0.9; k = 0.03, noise_window = 21.

Fig 3: Harris corners from images in my pair 1 at sigma = 1.2; k = 0.02, noise_window = 27.

Fig 4: Harris corners from images in my pair 1 at sigma = 1.6; k = 0.05, noise_window = 37.

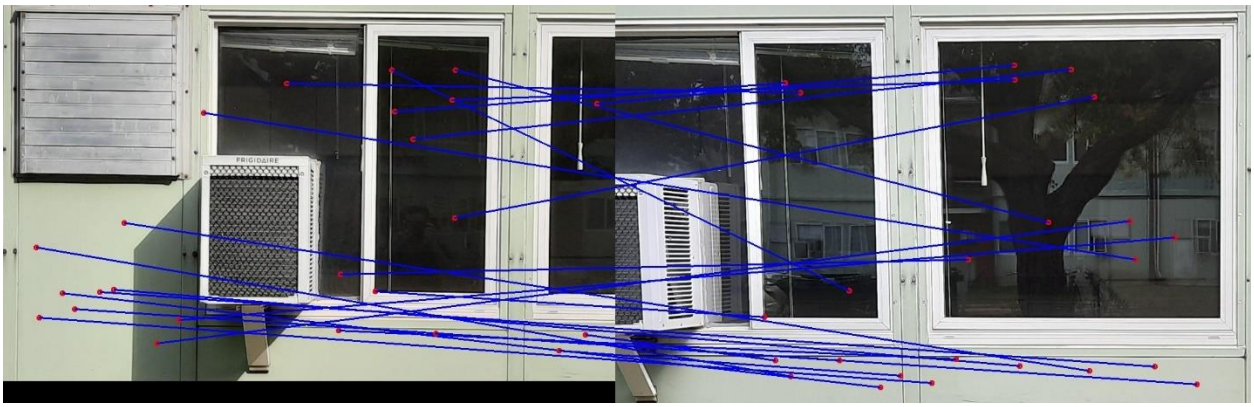

Fig 5: SSD on Harris corners from images in my pair 1 at sigma = 0.6; k = 0.06, noise_window = 21.

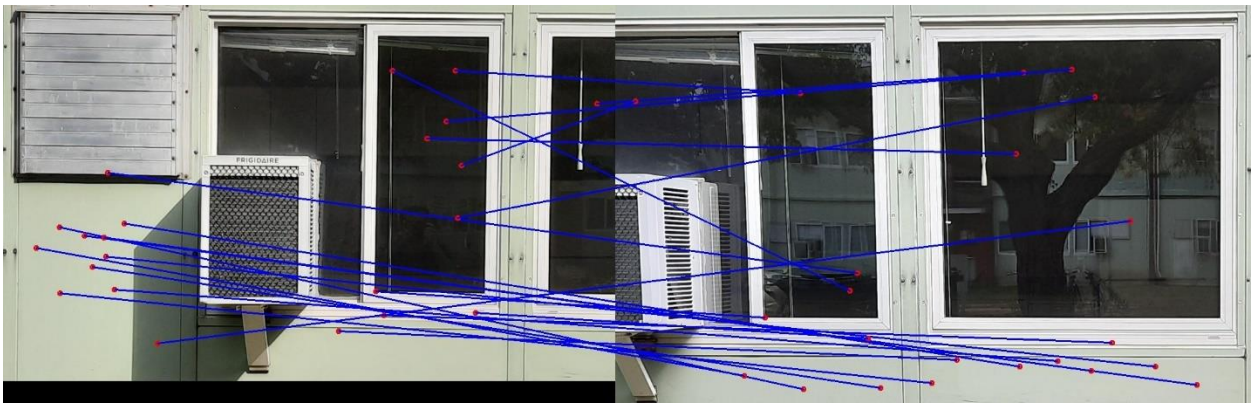

Fig 6: SSD on Harris corners from images in my pair 1 at sigma = 0.9; k = 0.03, noise_window = 21.

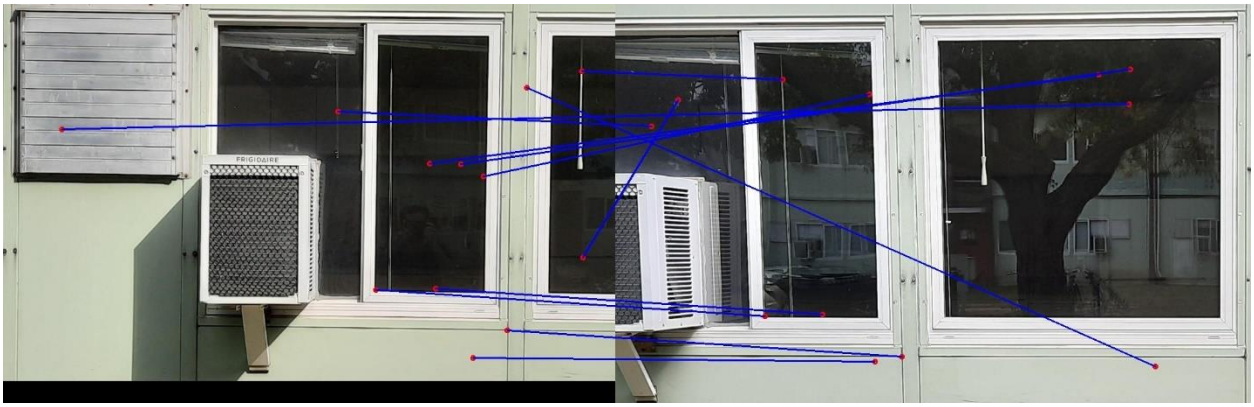

Fig 7: SSD on Harris corners from images in my pair 1 at sigma = 1.2; k = 0.02, noise_window = 27.

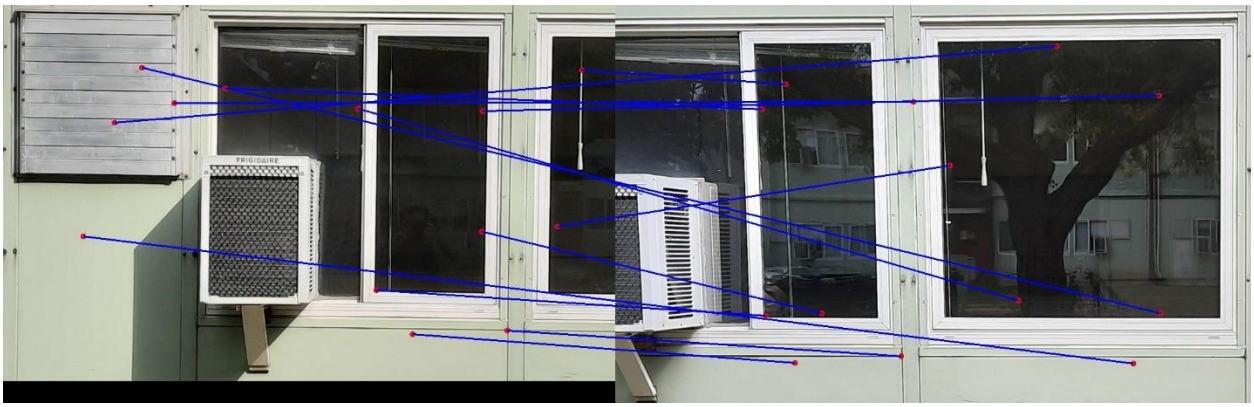

Fig 8: SSD on Harris corners from images in my pair 1 at sigma = 1.6; k = 0.05, noise_window = 37.

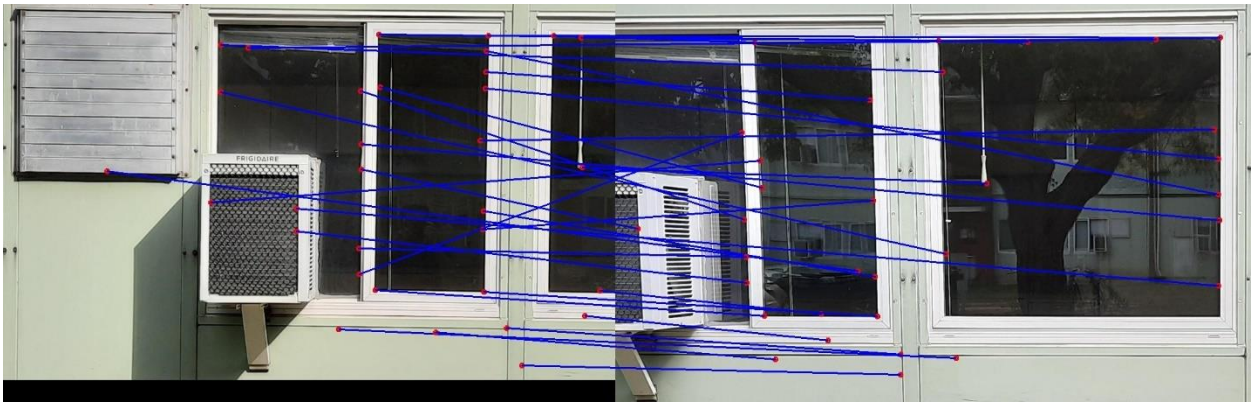

Fig 5: NCC on Harris corners from images in my pair 1 at sigma = 0.6; k = 0.06, noise_window = 21.

Fig 10: NCC on Harris corners from images in my pair 1 at sigma = 0.9; k = 0.03, noise_window = 21.

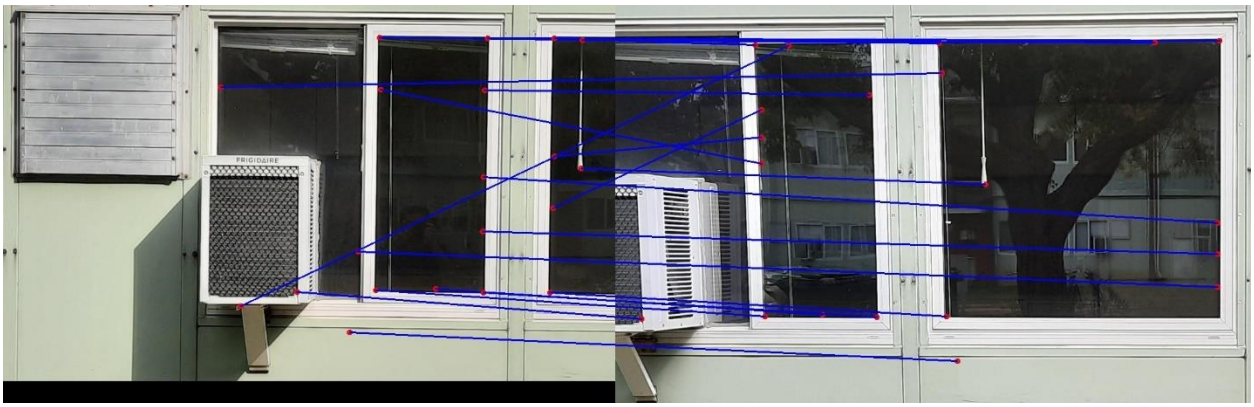

Fig 7: NCC on Harris corners from images in my pair 1 at sigma = 1.2; k = 0.02, noise_window = 27.

Fig 12: NCC on Harris corners from images in my pair 1 at sigma = 1.6; k = 0.05, noise_window = 37.

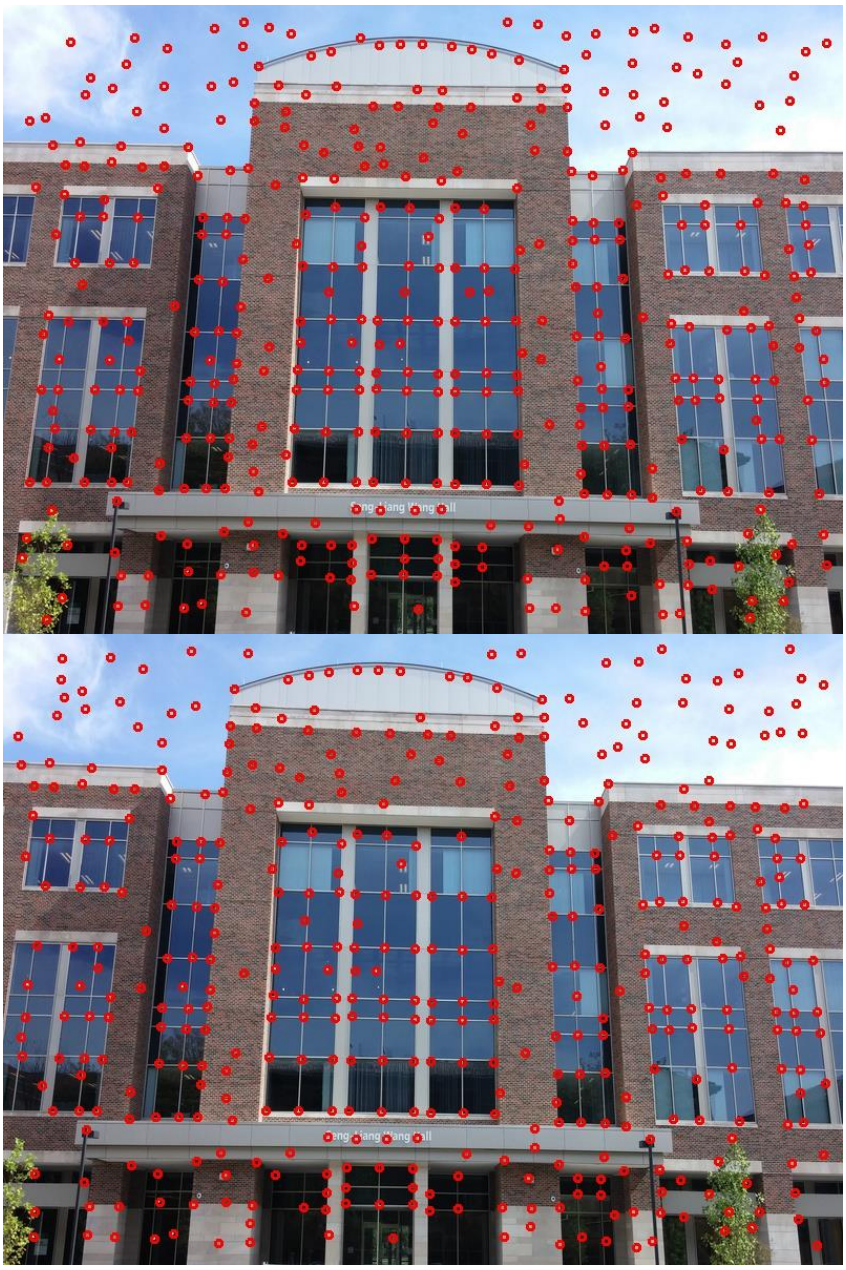

Fig 13: Harris corners from images in my pair 2 at sigma = 0.6;k = 0.04, noise_window= 15.

Fig 14: Harris corners from images in my pair 2 at sigma = 1.2;k = 0.04, noise_window= 15.

Fig 15: Harris corners from images in my pair 2 at sigma = 1.8;k = 0.04, noise_window= 15.

Fig 16: Harris corners from images in my pair 2 at sigma = 2.4;k = 0.04, noise_window= 15.

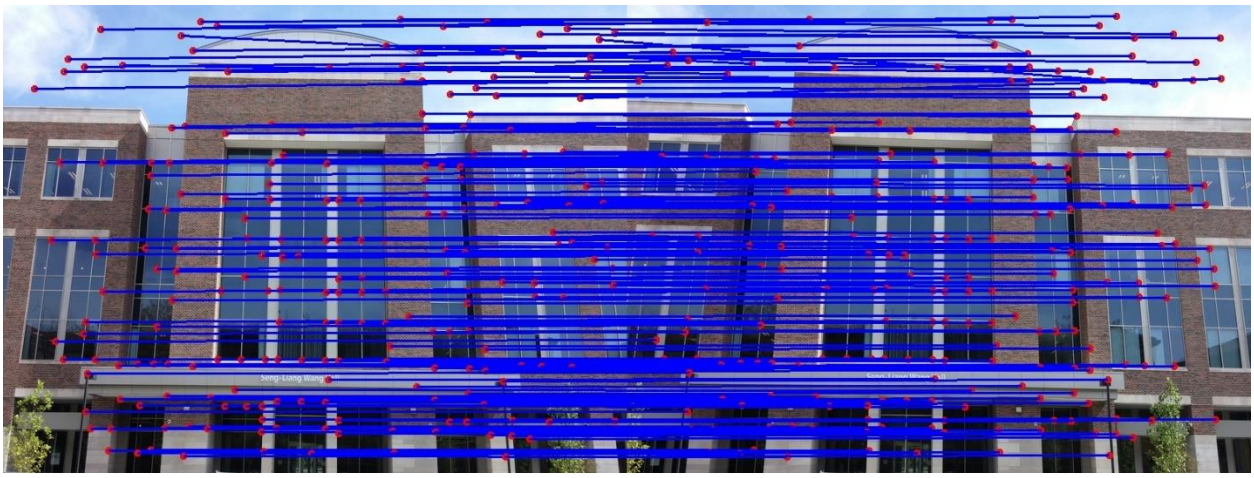

Fig 17: SSD on Harris corners from images in my pair 2 at sigma = 0.6;k = 0.04, noise_window= 15.

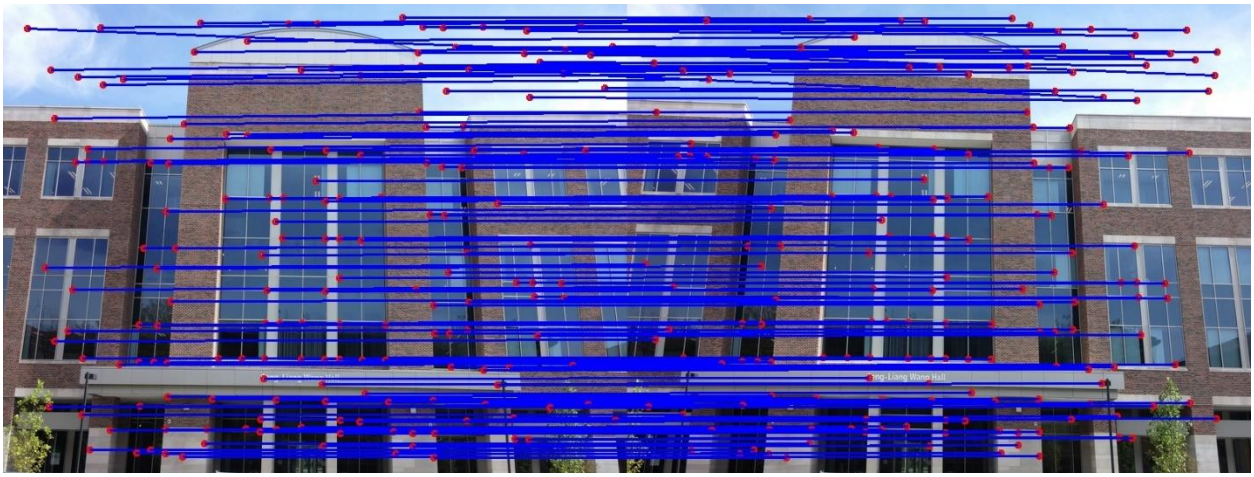

Fig 18: SSD on Harris corners from images in my pair 2 at sigma = 1.2;k = 0.04, noise_window= 15.

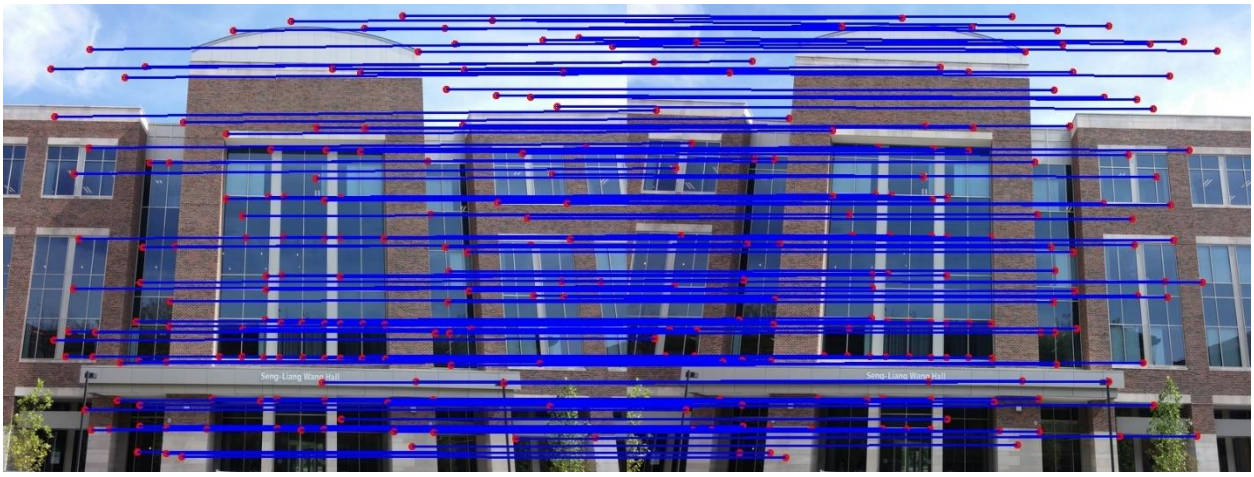

Fig 19: SSD on Harris corners from images in my pair 2 at sigma = 1.8;k = 0.04, noise_window= 15.

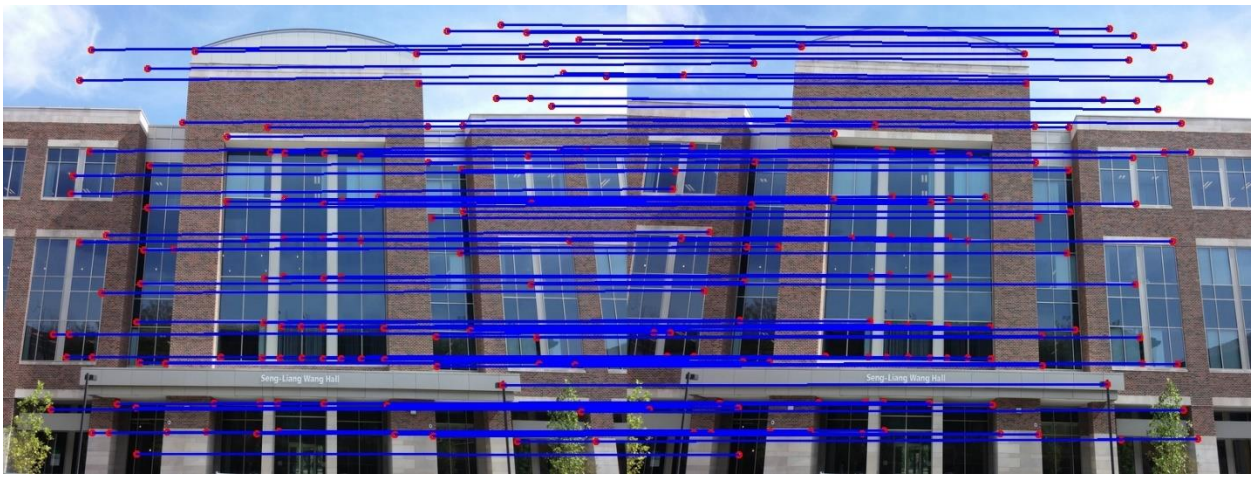

Fig 20: SSD on Harris corners from images in my pair 2 at sigma = 2.4;k = 0.04, noise_window= 15.

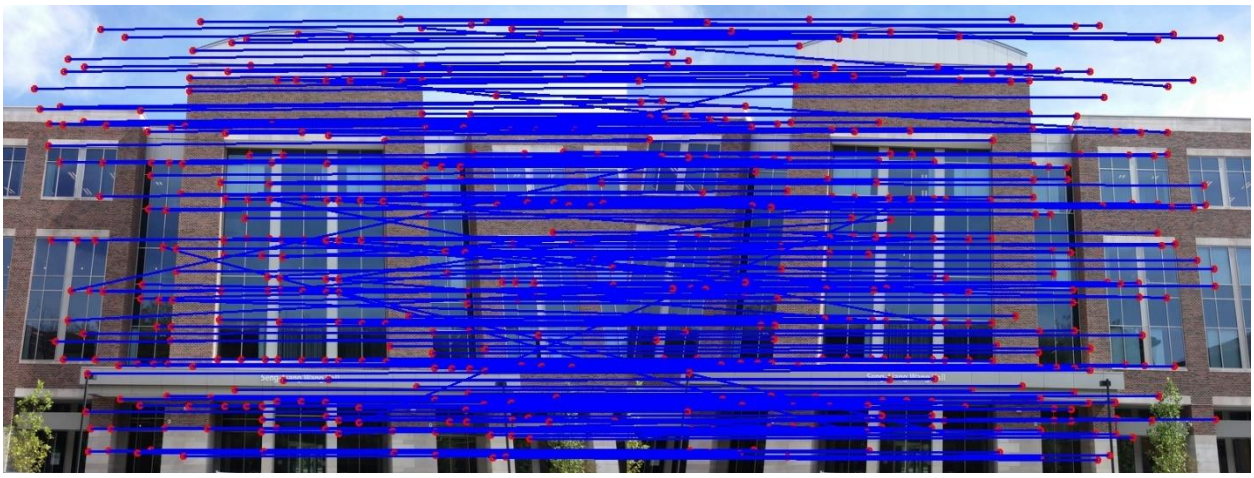

Fig 21: NCC on Harris corners from images in my pair 2 at sigma = 0.6;k = 0.04, noise_window= 15.

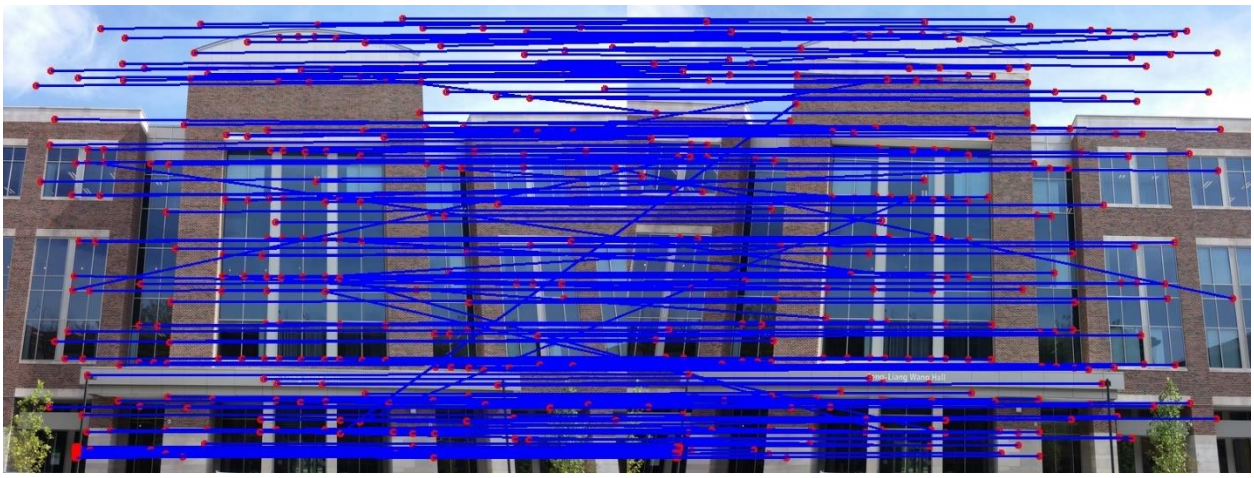

Fig 22: NCC on Harris corners from images in my pair 2 at sigma = 1.2;k = 0.04, noise_window= 15.

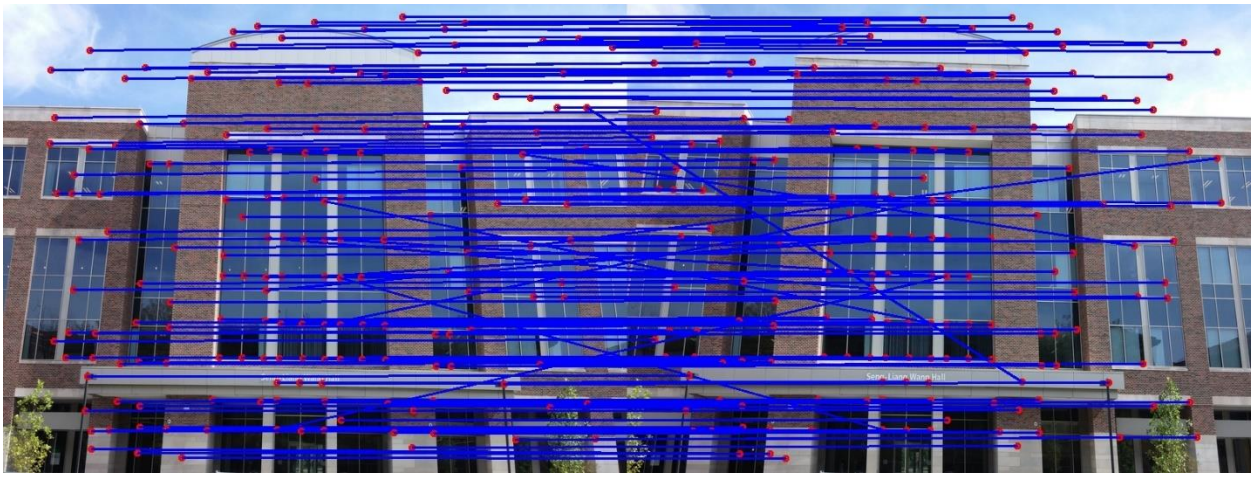

Fig 23: NCC on Harris corners from images in my pair 2 at sigma = 1.8;k = 0.04, noise_window= 15.

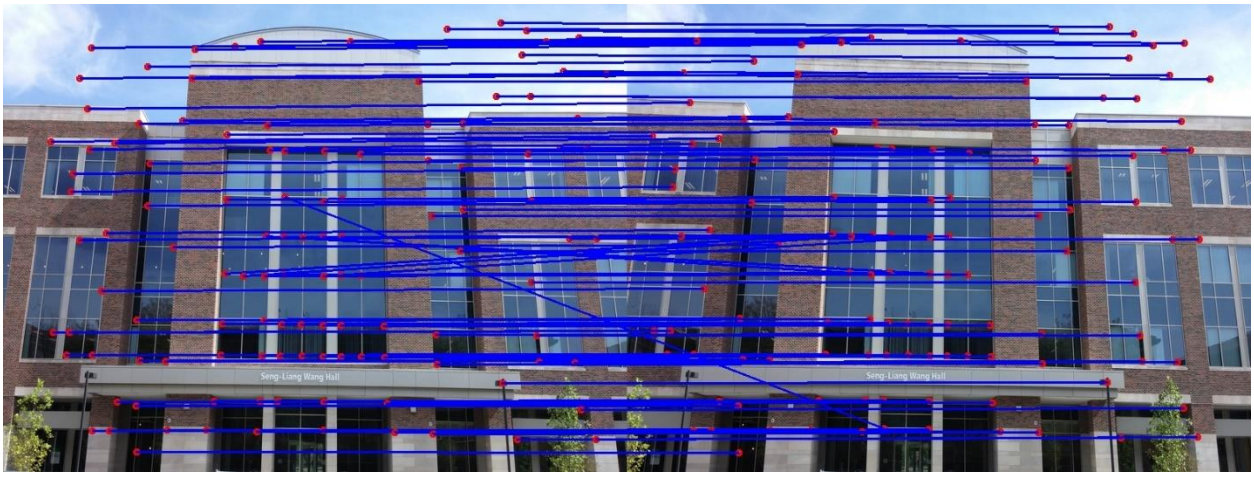

Fig 24: NCC on Harris corners from images in my pair 2 at sigma = 2.4;k = 0.04, noise_window= 15.

3.3. Task 2.2 (SIFT features and correspondences):

Fig 1: SIFT features from images in my pair 1 with contrast thresh =0.1 & max features =5000.

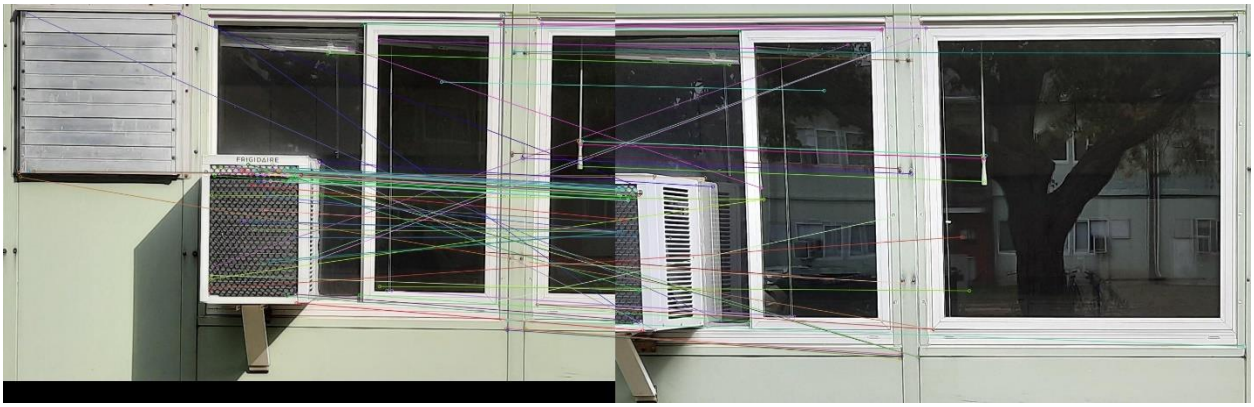

Fig 2: Corresponding SIFT features from images in my pair 1 with contrast thresh =0.1 & max features =5000 and brute force matching.

Fig 3: SIFT features from images in my pair 2 with contrast thresh =0.1 & max features =5000.

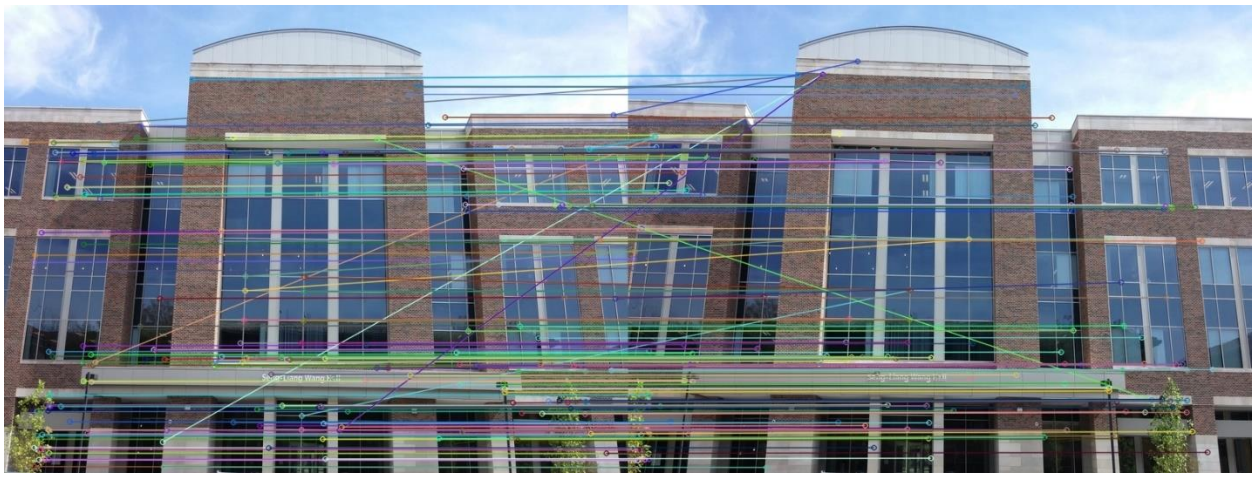

Fig 4: Corresponding SIFT features from images in my pair 2 with contrast thresh =0.1 & max features =5000 and brute force matching.

4. Observation of different methods:

1. Large scale results in reduced num of corners.

Source Code:

4.1. Function calls for Harris corners:

```
'''--------Main Code for Task 1.1--------'''

img_1,img_col_1=grayscale("./mypair2/1.JPG",0.75,False)
#Make a copy of color image before plotting
img_col_1_copy=np.copy(img_col_1)
#Pick the different scales,find the corner, plot and save
draw_and_save(img_col_1,img_1,0.04,0.6,14,'./mypair2_corners/1_Corners_0.6.jpg')
draw_and_save(np.copy(img_col_1_copy),img_1,0.04,1.2,14,'./mypair2_corners/1_Corners_
1.2.jpg')
draw_and_save(np.copy(img_col_1_copy),img_1,0.04,1.8,16,'./mypair2_corners/1_Corners_
1.8.jpg')
draw_and_save(np.copy(img_col_1_copy),img_1,0.04,2.4,18,'./mypair2_corners/1_Corners_
2.4.jpg')

#For image 2
img_2,img_col_2=grayscale("./mypair2/2.JPG",0.75,False)
#Make a copy of color image before plotting
img_col_2_copy=np.copy(img_col_2)
#Pick the different scales,find the corner, plot and save
```

```
draw_and_save(img_col_2,img_2,0.04,0.6,14,'./mypair2_corners/2_Corners_0.6.jpg')
draw_and_save(np.copy(img_col_2_copy),img_2,0.04,1.2,16,'./mypair2_corners/2_Corners_1.2.jpg')
draw_and_save(np.copy(img_col_2_copy),img_2,0.04,1.8,16,'./mypair2_corners/2_Corners_1.8.jpg')
draw_and_save(np.copy(img_col_2_copy),img_2,0.04,2.4,18,'./mypair2_corners/2_Corners_2.4.jpg')
```

4.2. Function calls for SSD and NCC:

```
'''-------Main Code for Task 1.2 (SSD)----------'''

img_1,img_col_1=grayscale("./mypair2/1.JPG",0.75,False)  #Resize my own images as they are big
img_2,img_col_2=grayscale("./mypair2/2.JPG",0.75,False)

Cornerslist_1=Harris_croner(img_1,0.04,2.4,18)
Cornerslist_2=Harris_croner(img_2,0.04,2.4,18)

SSD_Correspondences = Get_SSD_Correspondences(img_1,img_2,Cornerslist_1,Cornerslist_2,25,0.65)
comb_img_ssd = Show_Correspondences(img_col_1,img_col_2,SSD_Correspondences)
cv2.imwrite('./mypair2_corners/SSD_Correspondences_2.4.jpg',comb_img_ssd)

'''-------Main Code for Task 1.3 (NCC)----------'''

img_1,img_col_1=grayscale("./mypair2/1.JPG",0.75,False)  #Resize my own images as they are big
img_2,img_col_2=grayscale("./mypair2/2.JPG",0.75,False)

#Cornerslist_1=Harris_croner(img_1,0.06,0.6,21)
#Cornerslist_2=Harris_croner(img_2,0.06,0.6,21)

NCC_Correspondences = Get_NCC_Correspondences(img_1,img_2,Cornerslist_1,Cornerslist_2,25,0.9)
comb_img_ncc = Show_Correspondences(img_col_1,img_col_2,NCC_Correspondences)
cv2.imwrite('./mypair2_corners/NCC_Correspondences_2.4.jpg',comb_img_ncc)
```

4.3. Function calls for SIFT features:

```
'''-------Main Code for Task 1.4 (SIFT)----------'''
'''
#Read the images
img_1,img_col_1=grayscale("./mypair1/1.JPG",0.5,True)  #Resize my own images as they are big
img_2,img_col_2=grayscale("./mypair1/2.JPG",0.5,True)  #Resize my own images as they are big
#Get the keypoints
kp1,des1=Get_SIFT_Features(img_1)
```

```
kp2,des2=Get_SIFT_Features(img_2)
#Draw and save the output images
draw_and_save_Sift(img_col_1,kp1,'./mypair1_corners/1_SIFT.jpg')
draw_and_save_Sift(img_col_1,kp2,'./mypair1_corners/1_SIFT.jpg')
```

A. Code for reading the image, reducing size and converting to gray scale:

```
def grayscale(imgname,resize_scale=0.5,resize_falg=True):
    '''
    Read the image and convert it into the grayscale if not already.
    '''
    #Read the color image
    img_color = cv2.imread(imgname)
    # Adjust the width and height by a constant factor, this maintains the aspect ratio
    if resize_falg == True:
        w = int(img_color.shape[1]*resize_scale)
        h = int(img_color.shape[0]*resize_scale)
        img_color=cv2.resize(img_color,(w,h),interpolation = cv2.INTER_LINEAR)
    else:
        img_color = img_color

    #Check if the image has been read
    if img_color is not None:
        #Check if the image is color
        if len(img_color.shape)==3:
            #Convert to gray scale
            img_gray= cv2.cvtColor(img_color,cv2.COLOR_BGR2GRAY)
        elif len(img_color.shape)==1:
            img_gray=img_color
        else:
            print("Image shape not identified")
        return  img_gray,img_color
    else:
        print ("Image not found:"+imgname)
    return None
```

B. Function for computing the Haar kernels:

```
def Haar_kernels(sigma):
    '''
    Get the haar kernels for as a function of sigma.
    The sigma controls the size of kernels. The final output
    size is smallest even integer > 4*sigma (as per Lecture 9)
    '''
    kernel_dim=np.ceil(4*sigma)
    if kernel_dim % 2==0:
        kernel_dim=kernel_dim #Scale that rounds up to the even value
```

```
    else:
        kernel_dim=kernel_dim+1 #Scale that rounds upto odd values
    kernel_dim=int(kernel_dim)
    # Get the haar kernel in x direction
    kernel_dx=np.concatenate((-
1*np.ones((kernel_dim,int(kernel_dim/2))),np.ones((kernel_dim,int(kernel_dim/2)))),axis=1)
    # Get the haar kernel in y direction
    kernel_dy=np.concatenate((np.ones((int(kernel_dim/2),kernel_dim)),-
1*np.ones((int(kernel_dim/2),kernel_dim))),axis=0)
    return kernel_dx,kernel_dy
```

C. Function for calculating the Harris corners:

```
def Harris_croner(img_gray,k=0.04,sigma=1.2,suppresion_window=14):
    '''
    Function for finding harris corners
    Inputs:
        img_gray: FThe gray image from which the corners will be detected
        k: Cornerness factor, usually 0.04-0.06
        sigma: scale factor at which the corners will be detected
    Output: A list of found corners

    '''

    #Initialize an empty list for storing corners
    valid_corners = []

    #Calculate the Haar kernel for measuring the gradients in x and y direction
    haar_dx,haar_dy=Haar_kernels(sigma)

    #Compute the x and y derivatives by performing the convolution
    #Keep the dimension same to input image
    Ix = sig.convolve2d(img_gray,haar_dx,mode='same')
    Iy = sig.convolve2d(img_gray,haar_dy,mode='same')

    #Compute the sqaure and cross gradients for the C matrix
    Ixx = Ix**2
    Iyy = Iy**2
    Ixy = Ix*Iy
    '''
    Calculate the sum of gradient at each pixel by 5*sigma window.
    To avoid 2 foor loops this is done by convolving a filter with equal
    weights =1. Alternatively, Guassian kernel can be used with sigma controlling
    width to acheive the spatial integration.
    '''
    kernel_dim = 5*sigma
    if kernel_dim % 2==0:
```

```
        kernel_dim=kernel_dim+1 #If even make it odd
    else:
        kernel_dim=kernel_dim # else odd
    kernel_dim=np.int(kernel_dim)
    Gauss_kernel = np.ones((kernel_dim,kernel_dim))
    #Get the sum for Ixx, Ixy and Ixy
    Ixx_sum = sig.convolve2d(Ixx,Gauss_kernel,mode='same')
    Iyy_sum = sig.convolve2d(Iyy,Gauss_kernel,mode='same')
    Ixy_sum = sig.convolve2d(Ixy,Gauss_kernel,mode='same')

    #Compute the determinant of C matrix, where C = [[Ixx,Ixy],[Ixy,Iyy]]
    detC = (Ixx_sum * Iyy_sum) - (Ixy_sum**2)
    traceC = Ixx_sum + Iyy_sum
    # Compute the cornerness response of Harris detector
    R = detC - k * traceC**2
    #Remove negative entries
    R = R * (R > 0)

    # TODO: Vectorize loops --At present time consuming
    # Do max noise suppression to avoid finding the nearby corners
    #Reference:
(https://www.youtube.com/watch?v=_qgKQGsuKeQ&t=2545s,https://muthu.co/harris-
corner-detector-implementation-in-python/)
    kernel_half=suppresion_window    #This is half kernel upto which suppression will be done
    for y in range( kernel_half, img_gray.shape[0]-kernel_half):
        for x in range( kernel_half, img_gray.shape[1]-kernel_half):
            Win_img = Get_window(R,kernel_half,x,y)
            if R[ y, x ] == np.max(Win_img):
                valid_corners.append( [x,y] )
    return valid_corners

def Get_window(image,kernel_size,x,y):
    '''
    Function for finding the maximum inside a kernel.
    Requires the image, kernelsize and current image coordinates
    to define the current region occupied by kernel
    '''
    # Current kernel centered at x and y
    Window = image[y-kernel_size : y + kernel_size+1, x-kernel_size : x + kernel_size+1]
    #max_val = np.max(Window)
    return Window
```

D. Function for finding the correspondences using SSD:

```
def
Get_SSD_Correspondences(img_1_gray,img_2_gray,Cornerslist_1,Cornerslist_2,window_di
m=21,reject_ratio=0.8):
```

```
    '''
    Function for getting the corner correspondences from a pair of image using SSD
    Inputs:
        img_1_gray,img_2_gray: First and second gray images
        Cornerslist_1,Cornerslist_2: List of corners for 1st and 2nd image
        window_dim: Window size to define neiighboorhood for computing the Sum of Squared differences
        reject_ratio: Ratio for rejecting the corner. large value (0.8,0.9) = less conservative
                      small value (0.4,0.5) = more conservative
    Output: An array having variable length x 4 contining coordinates of corresponding corners

    '''
    # TODO: Vectorize loops --At present time consuming

    #Convert the corner lists to the arrays
    corner_image1 = np.array(Cornerslist_1)
    corner_image2 = np.array(Cornerslist_2)

    win_half = int(window_dim/2)
    #Initialize an empty list for storing corners
    valid_correspondences = []

    #Initialize 2D matrix to store the distances of a specific point in image 1 with everyother point in image 2
    #Size will be num_corners_1 x num_corners_2
    F = np.zeros((len(corner_image1),len(corner_image2)))

    #Fill the 2D matrix with SSD distance
    for y in range(len(corner_image1)):
        for x in range(len(corner_image2)):
            f1 =  Get_window(img_1_gray,win_half,corner_image1[y,0],corner_image1[y,1])
            f2 = Get_window(img_2_gray,win_half,corner_image2[x,0],corner_image2[x,1])
            F[y,x] = np.sum((f1-f2)**2)

    #Find the valid correspondences via normalized distance and reject if doesn't pass threshold
    for y in range(len(corner_image1)):
        #Find the two minimums for each row of a SSD matrix and normalize the distance
        dual_minima = np.partition(F[y,:],2)[:2]
        #Find the index of 1st minima in every row of SSD_matrix
        x=np.argmin(F[y,:])
        #Check if the normalize distance is less than the predefined threshold, then keep this correspondence otherwise reject it
        if dual_minima[0] / dual_minima[1] < reject_ratio:
            F[:,x] = np.inf   #Mark that this column has been taken to avoid double correspondence (hard learn't lesson)
```

```
valid_correspondences.append([corner_image1[y,0],corner_image1[y,1],corner_image2[x,0],
corner_image2[x,1]])

        #Find the min and max for every row of the SSD matrix and normalize the distance
        #reject_ratio=1-reject_ratio
        #x=np.argmin(F[y,:])
        #if np.min(F[y,:]) / np.max(F[y,:]) < reject_ratio:
        #       F[:,x] = np.max(F)   #Mark that this column has been taken to avoid double
correspondence (hard learn't lesson)
        #
valid_correspondences.append([corner_image1[y,0],corner_image1[y,1],corner_image2[x,0],
corner_image2[x,1]])

    return np.array(valid_correspondences)
```

E. Function for computing the correspondences using NCC:

```
def
Get_NCC_Correspondences(image1,image2,Cornerslist_1,Cornerslist_2,window_dim=21,re
ject_ratio=0.8):

    #Convert the corner lists to the arrays
    corner_image1 = np.array(Cornerslist_1)
    corner_image2 = np.array(Cornerslist_2)

    win_half = int(window_dim/2)
    #Initialize an empty list for storing corners
    valid_correspondences = []

    #Initialize 2D matrix to store the distances of a specific point in image 1 with everyother
point in image 2
    #Size will be num_corners_1 x num_corners_2
    F = np.zeros((len(corner_image1),len(corner_image2)))

    for y in range(len(corner_image1)):
        for x in range(len(corner_image2)):
            f1 =  Get_window(image1,win_half,corner_image1[y,0],corner_image1[y,1])
            f2 = Get_window(image2,win_half,corner_image2[x,0],corner_image2[x,1])
            mean1 = np.mean(f1)
            mean2 = np.mean(f2)
            numemnator = np.sum((f1- mean1)*(f2- mean2))
            denomenator = np.sqrt((np.sum((f1- mean1)**2))*(np.sum((f2- mean2)**2)))
            F[y,x] = numemnator/denomenator

    #Identify the corresponding corner points in the two images by thresholding
    for y in range(len(corner_image1)):
        x=np.argmax(F[y,:])
```

```
        if F[y,x] > reject_ratio:
            F[:,x] = np.NINF    #Mark that this column has been taken to avoid double correspondence (hard learn't lesson)

    valid_correspondences.append([corner_image1[y,0],corner_image1[y,1],corner_image2[x,0], corner_image2[x,1]])

    return np.array(valid_correspondences)
```

F. Function for showing the Harris corners and correspondences:

```
def draw_and_save(img_color,img_gray,k_ratio,sigma,suppresion_window,savename):
    Cornerslist=Harris_croner(img_gray,k_ratio,sigma,suppresion_window)

    for i in range (len(Cornerslist)):
        #Plot a circle of red color with a radius of 3 to mark the corner points.
        cv2.circle(img_color, tuple(Cornerslist[i]), 3, (0,0,255), 2)
        #cv2.imshow(savename,img_color)
        cv2.imwrite(savename,img_color)
    print(len(Cornerslist))
    return Cornerslist

def Show_Correspondences(img_1_color,img_2_color,corresponding_corners):
    '''
    Function for plotting the corresponding points on the image
    '''
    #Shape of input images
    h1,w1=img_1_color.shape[0:2]
    h2,w2=img_2_color.shape[0:2]
    #Find the maximum height from 2 images. This will be the height of output image
    max_height = max(h1,h2)
    #create an empty image having size of max_height x w1+w2
    plot_img = np.zeros((max_height,(w1+w2),3))
    #Fill empty image with image1 and 2, this will leave empty border on the
    #image of least height
    plot_img [0:h1,0:w1,:] = img_1_color
    plot_img [0:h2,w1:,:] = img_2_color

    for point in corresponding_corners:
        #Plot a circle of red color with a radius of 3 to mark the corner points on img1.
        cv2.circle(plot_img,tuple(point[0:2]),3,(0,0,255),2)
        #Plot a circle of red color with a radius of 3 to mark the corner points on img2.
        cv2.circle(plot_img,tuple([point[2]+w1,point[3]]),3,(0,0,255),2)  #shift the pointer by width of img1
        #Plot the blue line joining corresponding points on images
        cv2.line(plot_img,tuple(point[0:2]),tuple([point[2]+w1,point[3]]),(255,0,0),2)
```

```
    return plot_img
```

G. Function for computing the SIFT features and correspondences:

```
def Get_SIFT_Features(img_gray,best_features=5000,thresh=0.1):
    #Define the SIFT parameters
    #best_features=5000
    #thresh=0.1 #This will control the contrast impacts
    #Create a SIFT object
    sift = cv2.xfeatures2d.SIFT_create(best_features,contrastThreshold=thresh,sigma=2.6)
    #Compute the sift features and descriptors
    kp, des = sift.detectAndCompute(img_gray,None)
    return kp, des
def Get_BruteForce_Correspondences(sift_des1, sift_des2):
    #Initialize the Brute force matching
    BF_matcher = cv2.BFMatcher()

    #Find the 2 valid matches for each point, Use Lowe threshold
    two_matches = BF_matcher.knnMatch(sift_des1,sift_des2,k=2)

    #Store all valid matches
    Valid_matches = []

    # Pick the best match based on Lowe's paper
    for m1, m2 in two_matches:
        if m1.distance < 0.75 * m2.distance:
            Valid_matches.append([m1])


    return Valid_matches
```

H. Function for showing the SIFT features and correspondences:

```
def Show_Correspondences_Sift(valid_matches,img_1_color,img_2_color,kp1,kp2):
    #def Show_Correspondences(,img_2_color,corresponding_corners)
    plot_img                                                                    =
cv2.drawMatchesKnn(img_1_color,kp1,img_2_color,kp2,valid_matches,None,flags=2)
    return plot_img

def draw_and_save_Sift(img_color,keypoints,savename):

output_img=cv2.drawKeypoints(img_color,keypoints,np.array([]),flags=cv2.DRAW_MATCHE
S_FLAGS_DRAW_RICH_KEYPOINTS)
    cv2.imwrite(savename,output_img)
```