ECE 661 - HW4

Bhavya Patel - pate1539@purdue.edu

9/23/24

Theory Question

Theory Question: What is the theoretical reason for why the LoG of an image can be computed as a DoG?

For a given image f(x,y), let ff(x,y,σ) represent its σ-smoothed version.

$$\mathbf{ff}(\mathbf{x},\mathbf{y},\sigma) = \int\int_{-\infty}^{\infty} f(x',y') \cdot g(x-x',y-y')\,dx'\,dy'$$

where the gaussian, g(x,y), is

$$\mathbf{g}(\mathbf{x},\mathbf{y}) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

The LoG also has the laplacian applied to it after the gaussian smoothing, so we can write the laplacian as

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Now that we have all the equations necessary we can rewrite the LoG as

$$\nabla^2 ff(x,y,\sigma) = f(x,y) \cdot h(x,y,\sigma)$$

where h(x,y,σ) can be written as

$$\mathbf{h}(\mathbf{x},\mathbf{y},\sigma) = \frac{1}{2\pi\sigma^4}(2 - \frac{x^2+y^2}{\sigma^2})\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

After some simplification, you will find that

$$\frac{\partial}{\partial\sigma} ff(x,y,\sigma) = \sigma \cdot f(x,y) \cdot h(x,y,\sigma)$$

which turns in

$$\frac{\partial}{\partial\sigma} ff(x,y,\sigma) = \sigma \cdot \nabla^2 f(x,y,\sigma)$$

The last equation shows that the LoG of an image can be found by taking the difference between the $(\sigma + \delta\sigma)$ smoothed version of the f(x,y) from the σ smoothed version. This difference is known as the DoG.

Theory Question

Theory Question: Explain in your own words why computing the LoG of an image as a DoG is computationally much more efficient for the same value of σ.

To compute the LoG with σ, you must calculate a 2nd-order derivative. However, when calculating DoG you can use the fact that in the gaussian, g(x,y) defined in the section above, is separable in x and y. Then you can do the 1-D smoothing along the x and y, which is much more computationally efficient then doing just one 2-D smoothing. This does not work for the LoG operator of h(x,y) because x and y are not separable.

Logic

Logic 4.1.1 - Harris Corner Detector

Some background on the Harris Corner Detection is it was the world's most popular intersect point detector, before SIFT and SURF. Its main use in today's world is if your application does not require a scale-space analysis or if the value of the scale is fixed and known in advance. The steps to implement the Harris Corner Detection are discussed below.

First, you turn the image into gray-scale because Harris Detector identifies if a pixel is a corner by changes in pixel intensity. This means you do not need to have color in the image, so we changed the image to gray-scale. Then you will use the Haar Wavelet Filter to calculate the first derivative based on x and y. These derivatives will be denoted as d_x and d_y respectively and will be used later.

I will expand on the Haar Wavelet Filters. The most basic form of the Haar Wavlets is (-1, 1) for x and $(1, -1)^T$ for y. These are scaled up to an MxM operator where M is the smallest even integer greater than 4 times σ. The gray-scale image will be convolved with both x and y Haar Wavelet to get d_x and d_y. Then using those calculated values you will create matrix C.

$$\mathbf{C} = \begin{pmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{pmatrix}$$

Matrix C is in a 5σx5σ neighborhood of the pixel where we wish to determine the presence or absence of a corner. All the summations are over all the pixels in the 5σx5σ neighborhood. Then λ_1 and λ_2 are the eigenvalues of C, where we assume $\lambda_1 \leq \lambda_2$. To be called a corner we need to find the ratio between the eigenvalues $\frac{\lambda_2}{\lambda_1}$. Below I will do some calculations to find the ratio, R, of the eigenvalues.

We know,

$$\mathbf{Tr}(\mathbf{C}) = \sum d_x^2 + \sum d_y^2 = \lambda_1 + \lambda_2$$

and,

$$\mathbf{det}(\mathbf{C}) = \sum d_x^2 \sum d_y^2 - (\sum d_x d_y)^2 = \lambda_1 \lambda_2$$

so,

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

We can rewrite the ratio, R, using the Harris Response Calculation. which is

$$\mathbf{R} = det(C) - k(Tr(C))^2$$

where k is a constant that is always between the values of 0.04 and 0.06. With a larger k, you get fewer false corners but miss more real corners. Then with a smaller k, you get more corners but will get lots of false corners. In my implementation, I found that the optimal k value was very dependent on the image and sigma value, so I keep it as a constant k = 0.5. The larger the R value the more likely the point is going to be a corner point.

With all the chosen corner points, you will apply non-max suppression. This is done because multiple pixels could be detected for some corners, so we will only pick the max R-value within a specified window. I decided that the window could be 15*σ and I automatically removed all R values that were less than the mean of the ratio values.

Logic 4.1.1 - Establishing Correspondences Between Image Pairs

You can establish correspondences between image pairs by directly comparing the gray level in a window, (m+1)x(m+1), around the corner pixel in one image with the gray levels in a similar window around the corresponding pixel in the other image. In my implementation, I used m = 29. We will achieve this by two methods. Normalized Cross-Correlation (NSS) and Sum of Squared Differences (SDD).

Logic 4.1.1 - NCC

NCC can be written as

Logic

$$\textbf{NCC} = \frac{\sum\sum(f_1(i,j) - m_1)(f_2(i,j) - m_2)}{\sqrt{[\sum\sum(f_1(i,j) - m_1)^2][\sum\sum(f_2(i,j) - m_2)^2]}}$$

where f_1 = image 1, f_2 = image 2, and m_i = mean of the f_i in the window. The value of NCC is always between 1 and 0. The closer the value is to 1 the better the match. So in my implementation, I just take the value closest to one as the final value.

Logic 4.1.1 - SSD

SSD can be written as

$$\textbf{SSD} = \sum_i \sum_j |f_1(i,j) - f_2(i,j)|^2$$

where f_1 is image 1 and f_2 is image 2. The SSD in other words just finds the corner points which have the smallest distance between them. So in my implementation, I chose the corner points by choosing the min of all the SSD values.

Logic 4.1.2 - SIFT

The first step when using SIFT is to create a DoG pyramid. A DoG pyramid is represented with D(x,y,σ) where σ is the scale value. The structure is there is σ_0, $2\sigma_0$, $4\sigma_0$, etc. where each sigma value is called an octave. Then for each octave you need to find an σ = xδ+σ_0 where x = 1,2,3. So there will be 4 σ-smoothed images in each octave. This results in 3 DoG images for each octave when you take the difference between the adjacent 4 σ-smoothed images.

Then you can find the local extrema in the DoG pyramid by comparing the middle DoG image pixels with the 26 neighboring pixel values. Where eight of them will be the surround pixels in the same DoG image, 9 will be from the above DoG image, and 9 will be from the below DoG image.

At the higher octaves, like $4\sigma_0$, the extrema are calculated based on a downsampled image,so the corresponding pixel is not exact. You can improve the location of the extremum in the DoG pyramid by taking the 2nd-order derivative of D at the sampling points in the DoG pyramid. The true location of the extremum is given by

$$\vec{x} = -H^{-1}(\vec{x}_0) \cdot J(x_0)$$

where

$$\vec{x} = (x_0, y_0, \sigma_0)^T$$

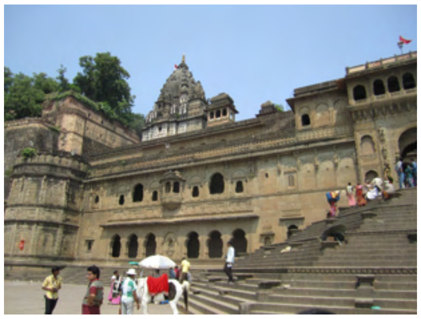
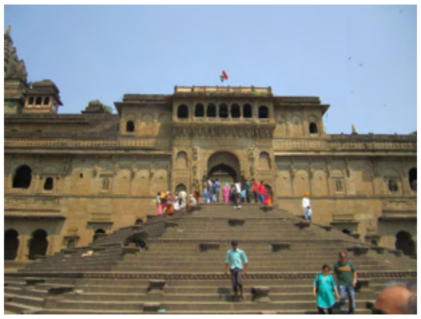
and

$$J(x_0) = \left(\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial \sigma} \right)$$

Then you can weed out extrema by thresholding $|D(\vec{x})|$ where an extremum is rejected if $|D(\vec{x})| \leq 0.03$. You now can find the dominant local orientation by calculating the gradient vector of the Gaussian-Smoothed Image ff(x,y,σ) at the scale value, σ, of the extremum. You compute the gradient magnitude, m(x,y), and gradient orientation, θ(x,y), at each point in a KxK neighborhood around the extremum. Then after weighting θ(x,y) with m(x,y), we construct a histogram of θ(x,y) values using 36 separate bins that span 360 degrees. The bin that is the largest gives the dominant local orientation.

Finally SIFT creates a 128-dimensional vector at each extremum in the DoG pyramid called the feature vector. This vector allows you to know the image's dominant orientation, so you can match point pairs through in-planar rotation. So SIFT is invariant to in-planar rotation.

Task 1 Results

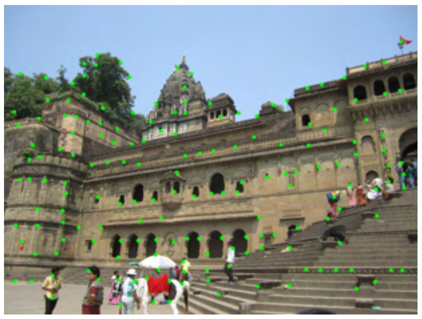
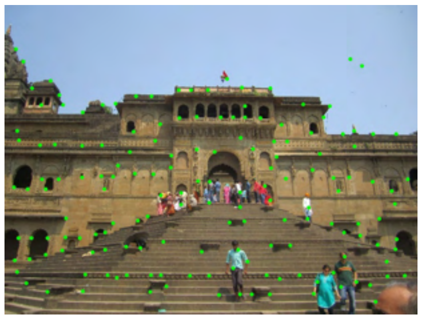



Original Images for Temple Image

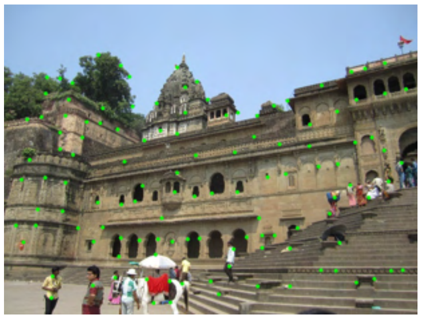
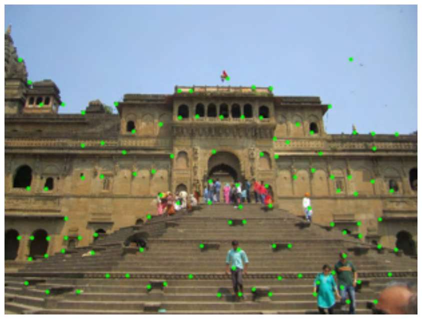
Harris Corner Points Results - Task 1 Temple Image

In the images below I will display what my Harris Corner Point Detector found for the sigma values of 1, 1.2, 1.6, 2 for the Temple Image.

Task 1 Results

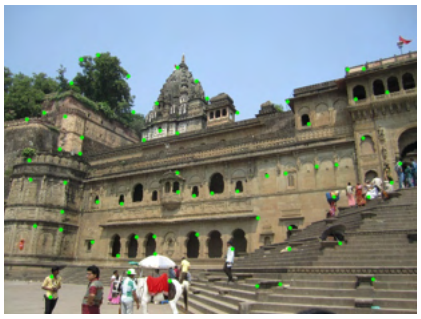
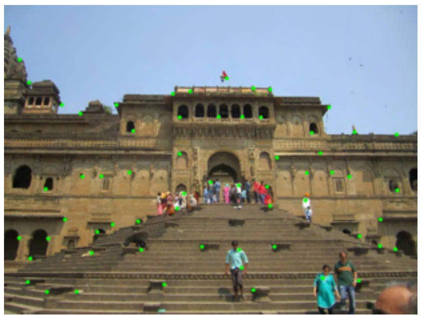



Harris Corner Points for Sigma = 1

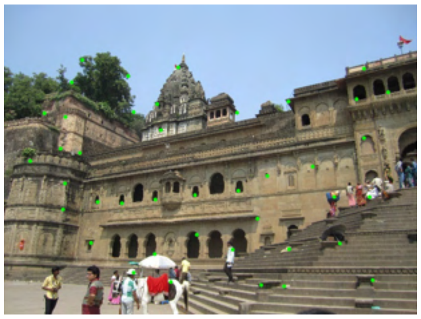
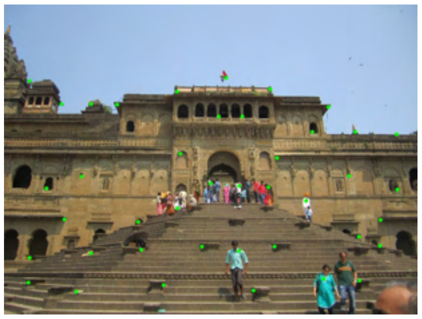



Harris Corner Points for Sigma = 1.2

Task 1 Results

Harris Corner Points for Sigma = 1.6

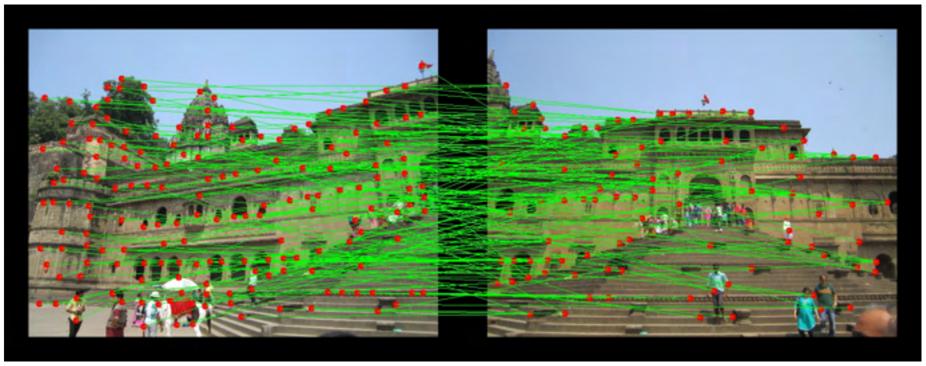



Harris Corner Points for Sigma = 2

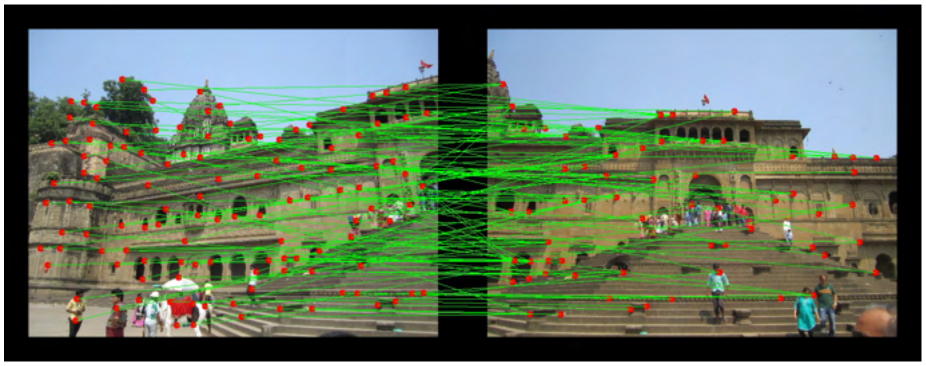
Task 1 Results

NCC Results - Task 1 Temple Image

In the images below I will display the NCC Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for the Temple Image.

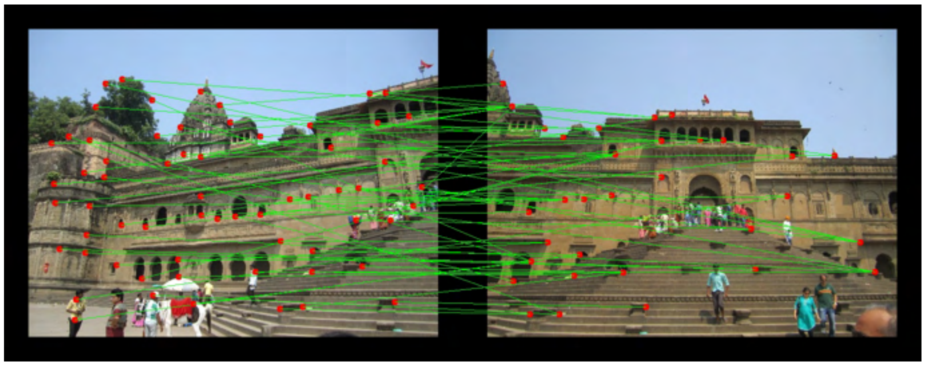


NCC Point Correspondences for Sigma = 1

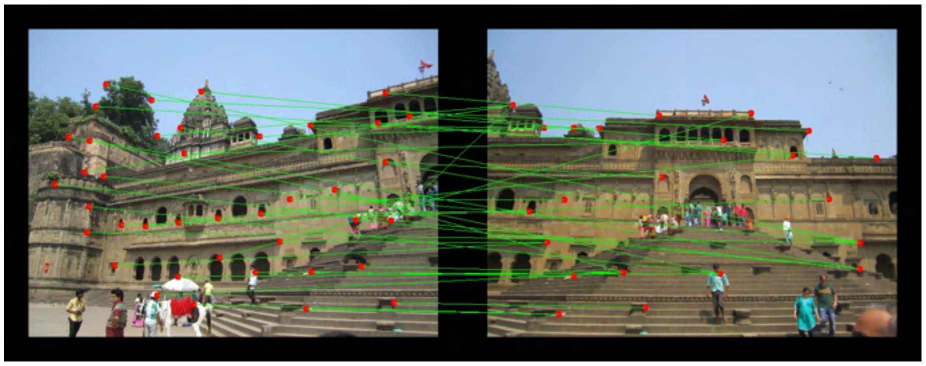


NCC Point Correspondences for Sigma = 1.2

Task 1 Results



NCC Point Correspondences for Sigma = 1.6

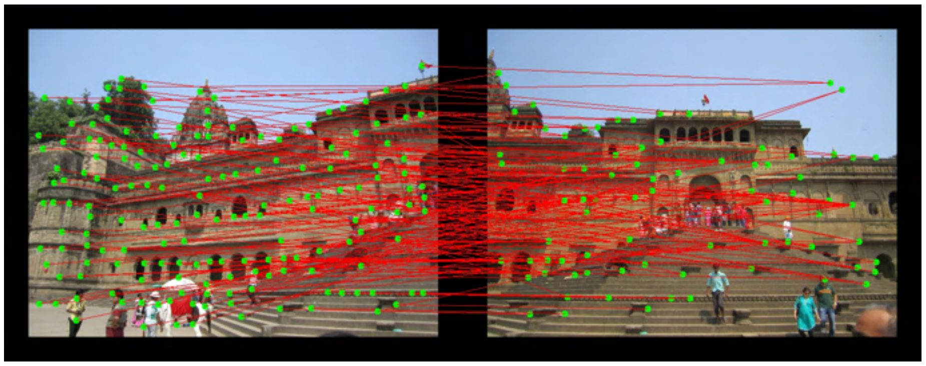


NCC Point Correspondences for Sigma = 2

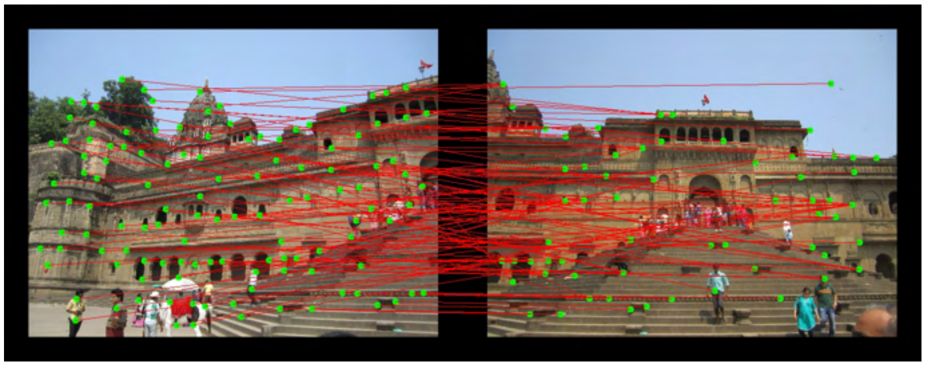
Task 1 Results

SSD Results - Task 1 Temple Image

In the images below I will display the SSD Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for the Temple Image.

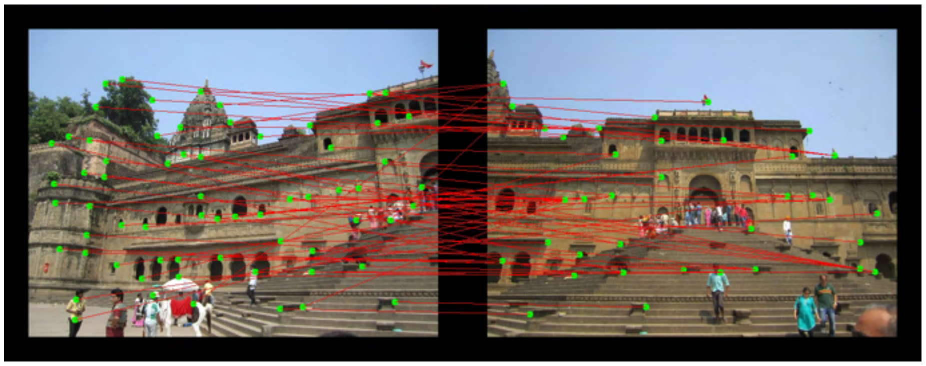


SSD Point Correspondences for Sigma = 1

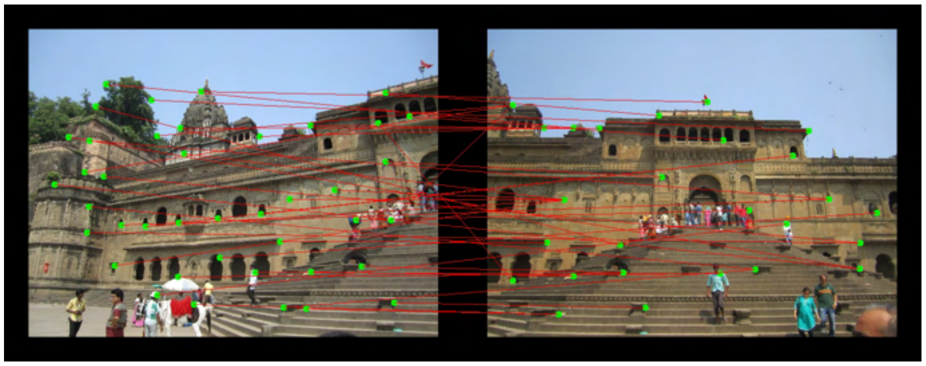


SSD Point Correspondences for Sigma = 1.2

Task 1 Results

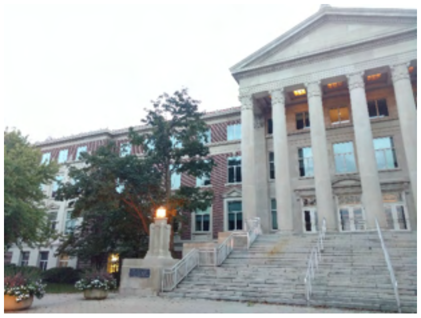
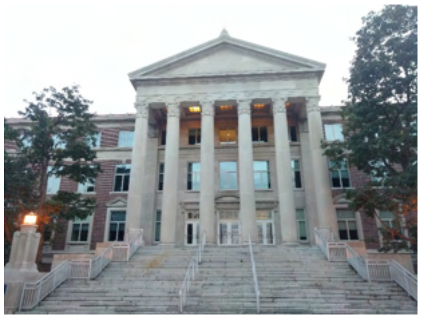


SSD Point Correspondences for Sigma = 1.6



SSD Point Correspondences for Sigma = 2

Task 1 Results

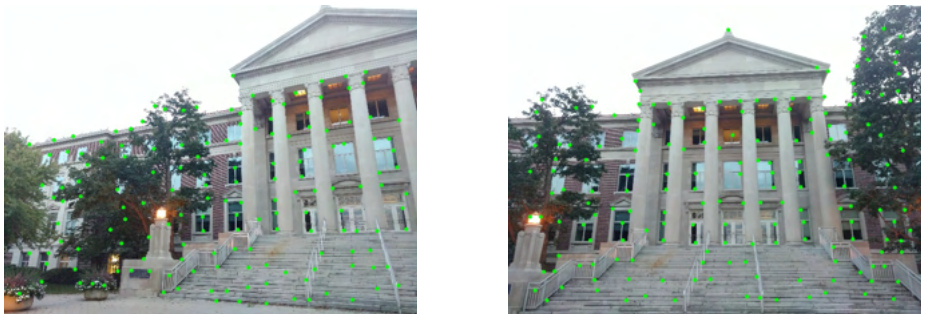



Original Images for Hovde Image

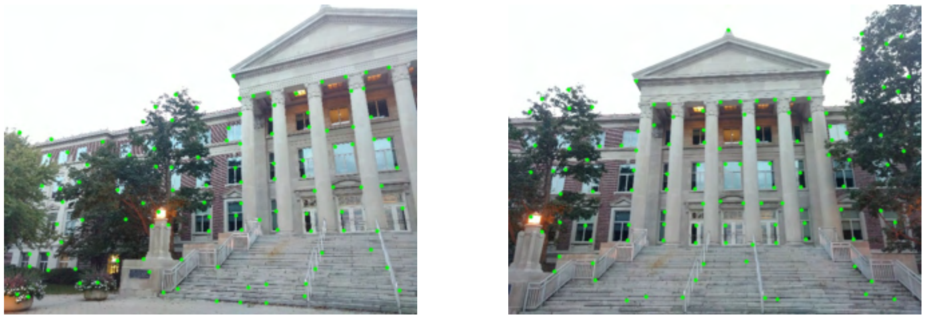
Harris Corner Points Results - Task 1 Hovde Image

In the images below I will display what my Harris Corner Point Detector found for the sigma values of 1, 1.2, 1.6, 2 for the Hovde Image.

Task 1 Results

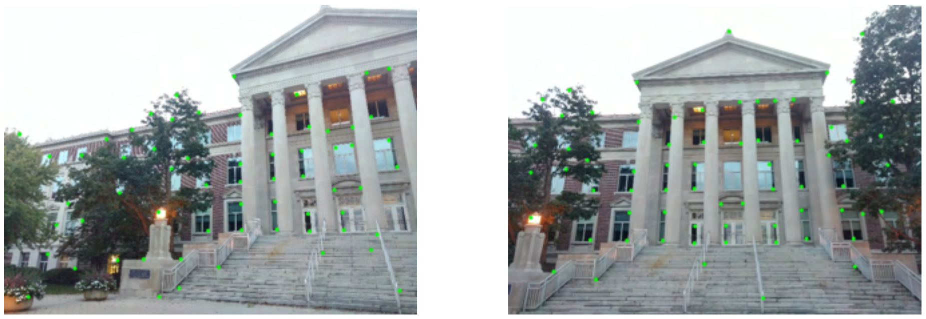


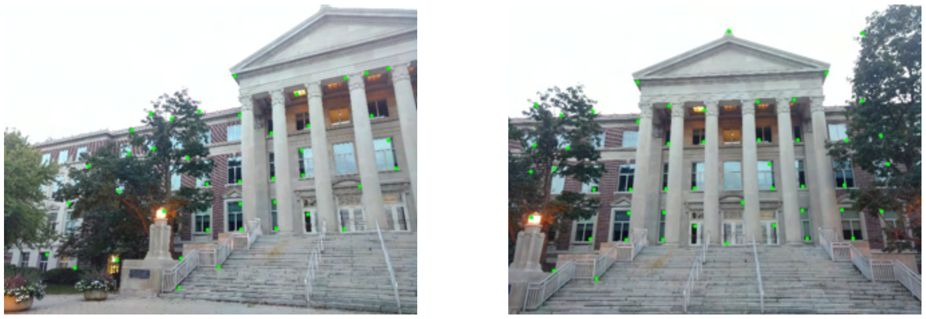
Harris Corner Points for Sigma = 1



Harris Corner Points for Sigma = 1.2

Task 1 Results

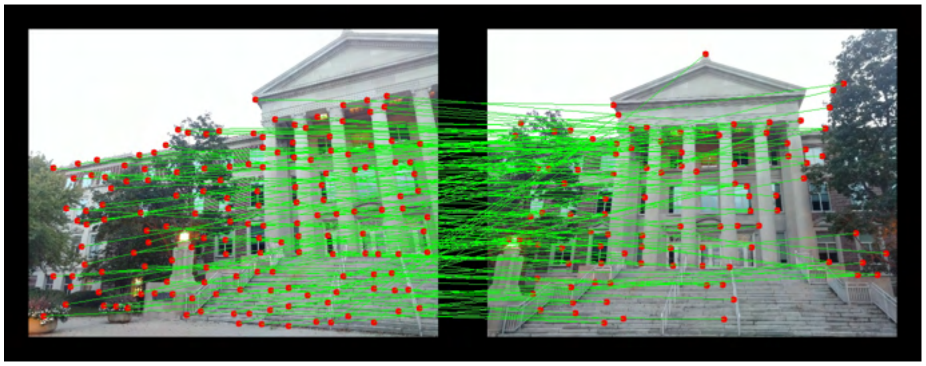

Harris Corner Points for Sigma = 1.6


Harris Corner Points for Sigma = 2

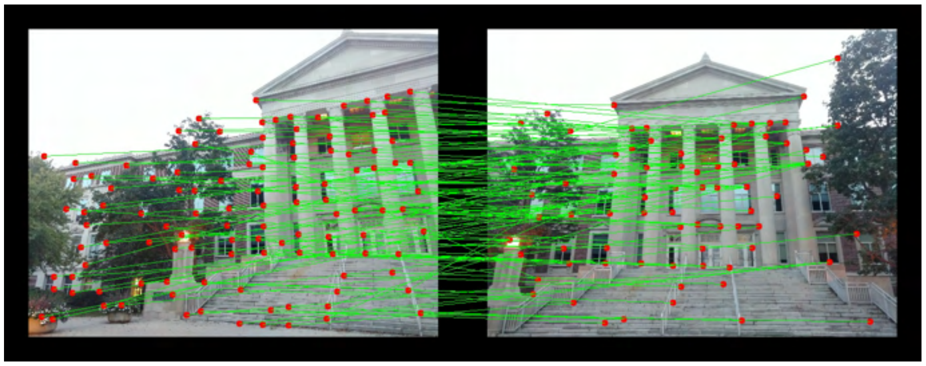
Task 1 Results

NCC Results - Task 1 Hovde Image

In the images below I will display the NCC Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for the Hovde Image.

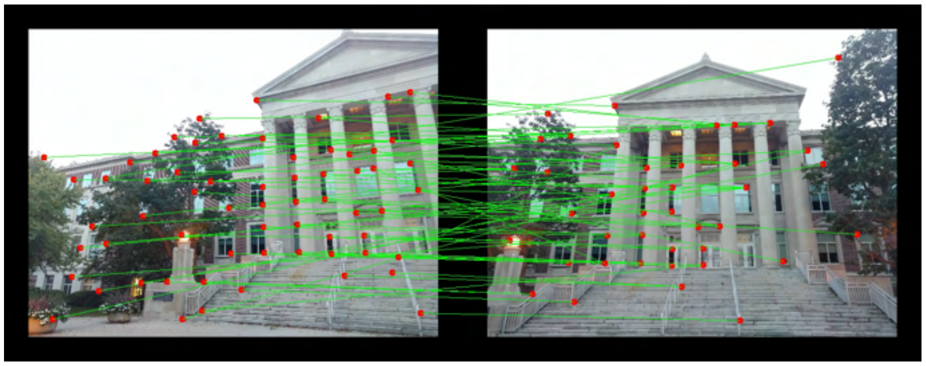


NCC Point Correspondences for Sigma = 1

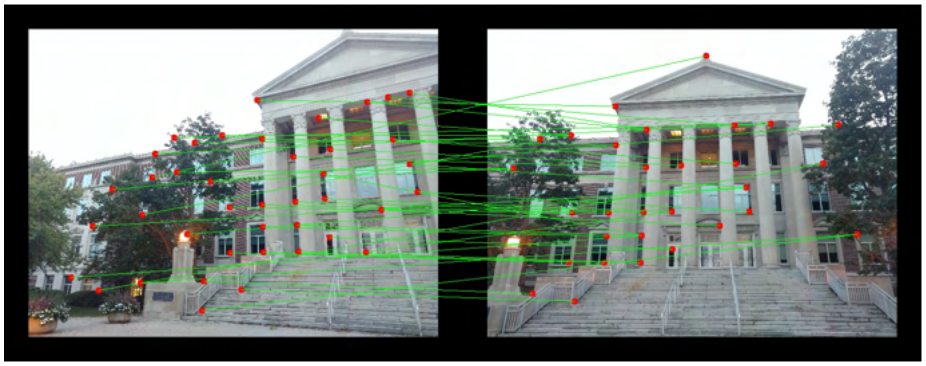


NCC Point Correspondences for Sigma = 1.2

Task 1 Results



NCC Point Correspondences for Sigma = 1.6

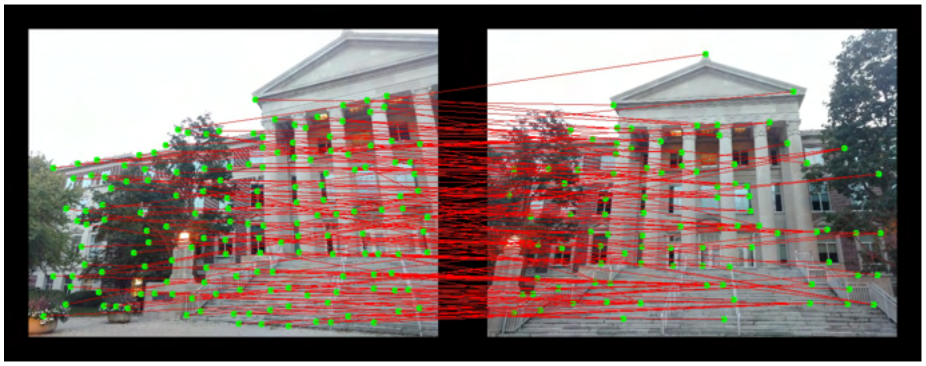


NCC Point Correspondences for Sigma = 2

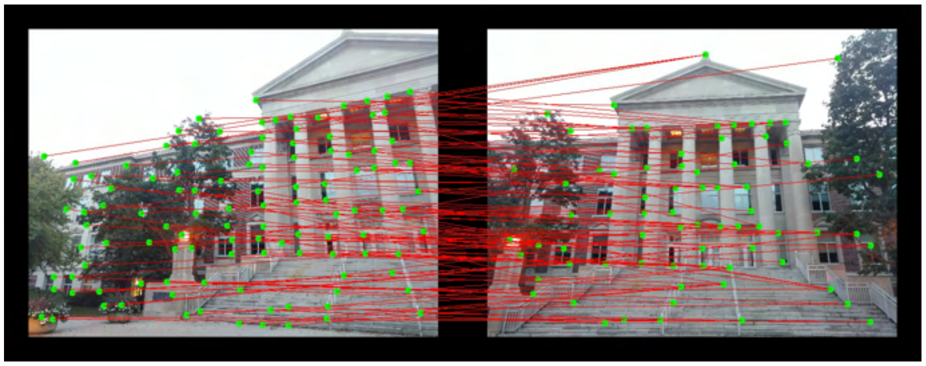
Task 1 Results

SSD Results - Task 1 Hovde Image

In the images below I will display the SSD Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for the Hovde Image.

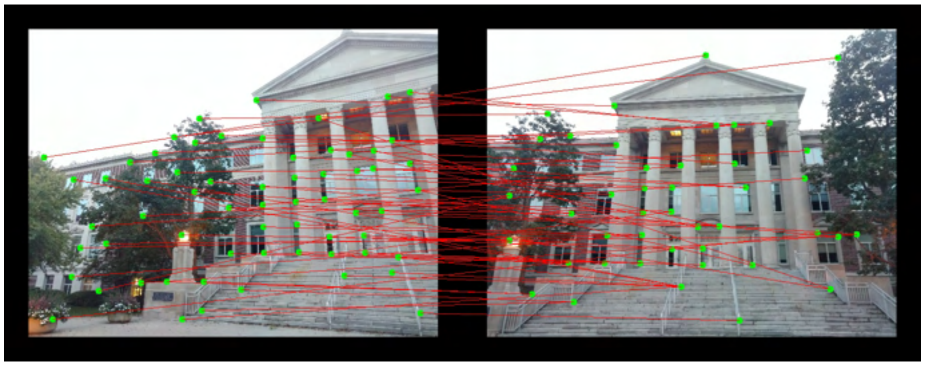


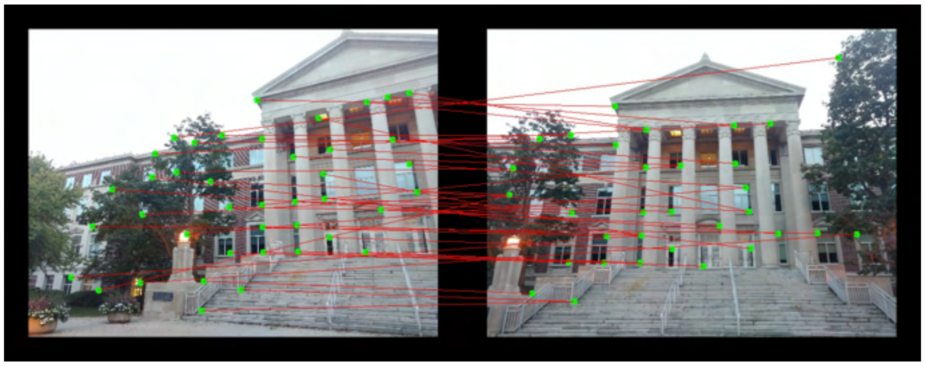
SSD Point Correspondences for Sigma = 1



SSD Point Correspondences for Sigma = 1.2

Task 1 Results

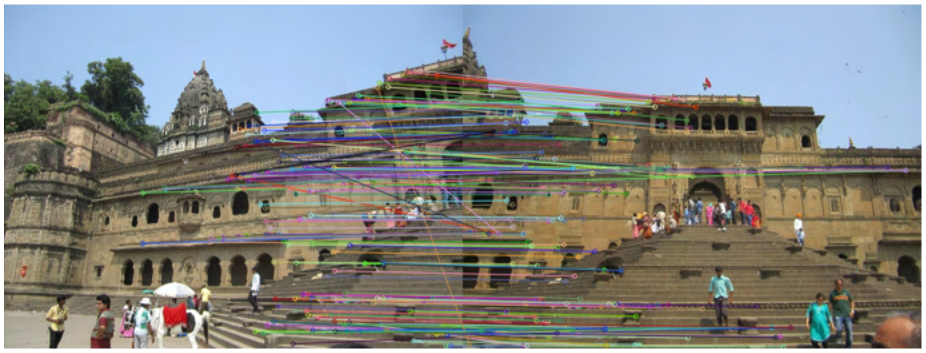

SSD Point Correspondences for Sigma = 1.6


SSD Point Correspondences for Sigma = 2

Task 1 Results

SIFT Results - Task 1 Temple Image

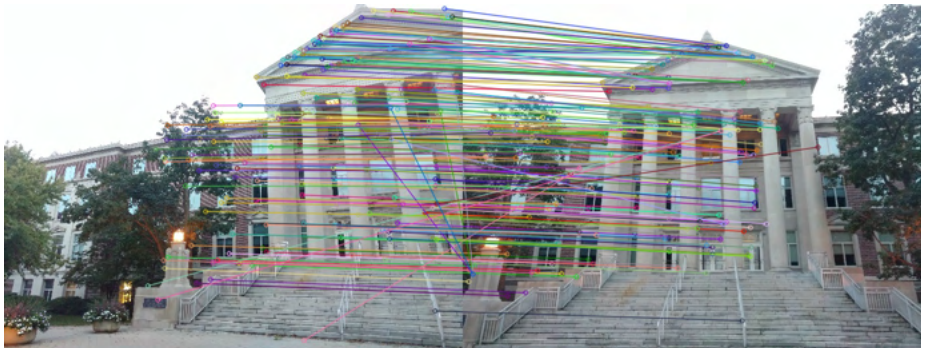
In the image below, I will display the SIFT algorithm result for the Temple Image.



SIFT Result for Temple Image

SIFT Results - Task 1 Hovde Image

In the image below, I will display the SIFT algorithm result for the Hovde Image.

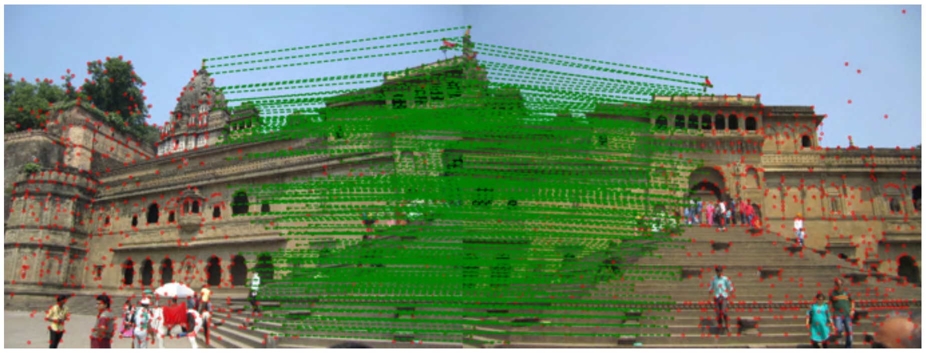


SIFT Result for Hovde Image

Task 1 Results

SuperPoint and SuperGlue Results - Task 1 Temple Image

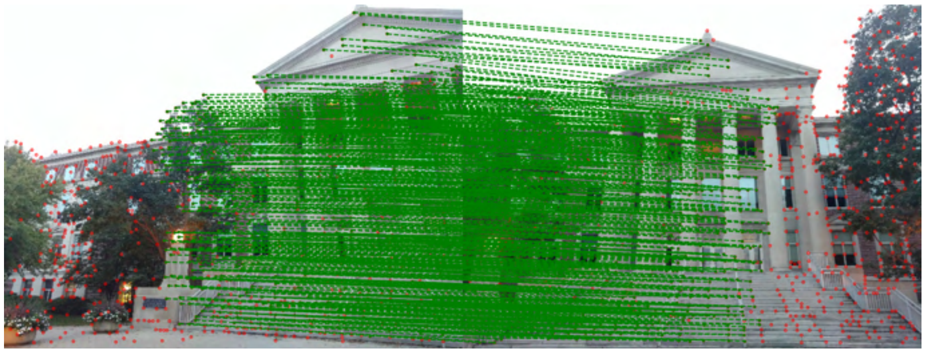
In the image below, I will display the Superpoint and Superglue result for the Temple Image.



Superpoint and Superglue Result for Temple Image

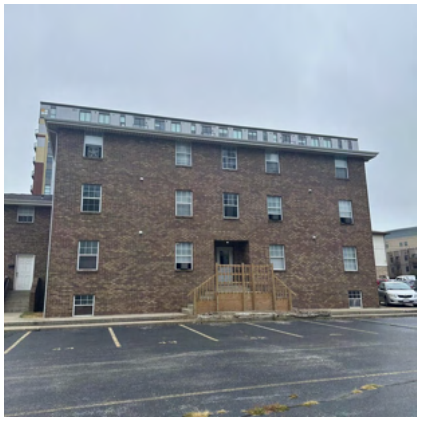
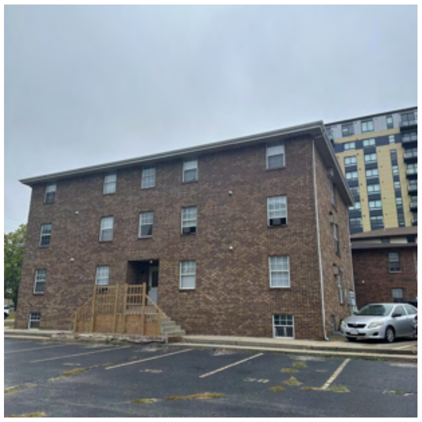
SuperPoint and SuperGlue Results - Task 1 Hovde Image

In the image below, I will display the Superpoint and Superglue result for the Hovde Image.



Superpoint and Superglue Result for Hovde Image

Task 2 Results

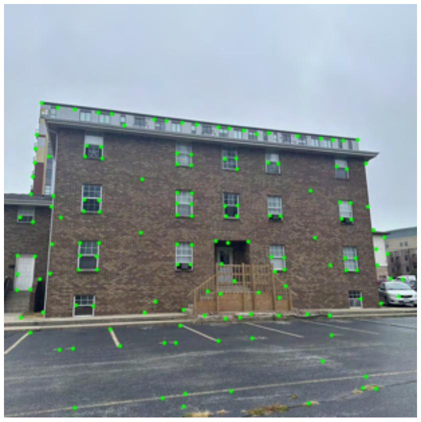
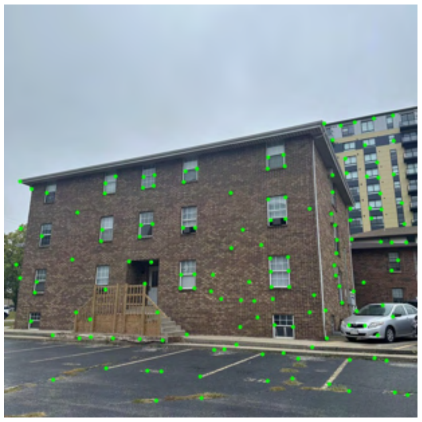



Original Images for My Image 1

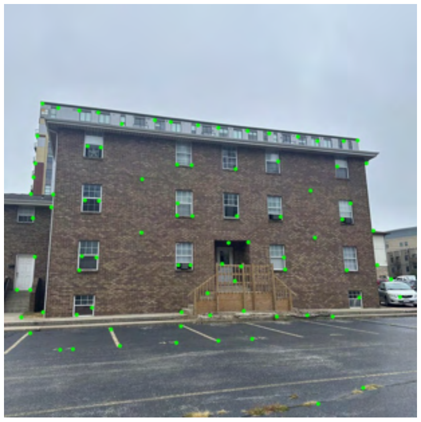
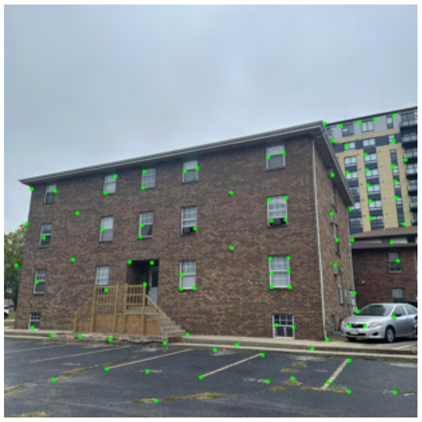
Harris Corner Points Results - Task 2 My Image 1

In the images below I will display what my Harris Corner Point Detector found for the sigma values of 1, 1.2, 1.6, 2 for My Image 1.

Task 2 Results

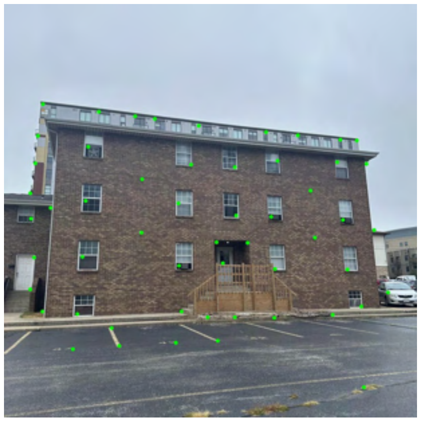
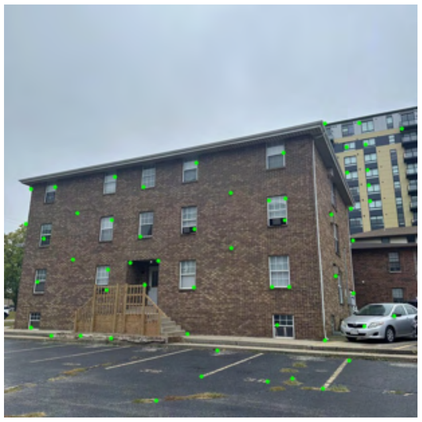



Harris Corner Points for Sigma = 1

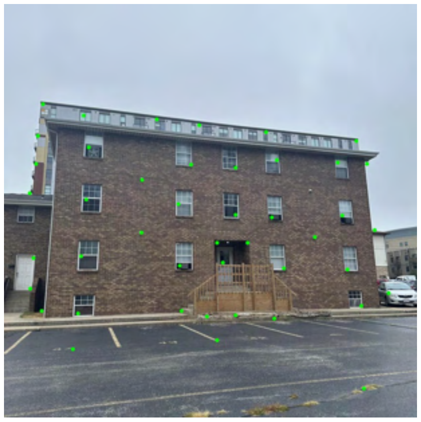
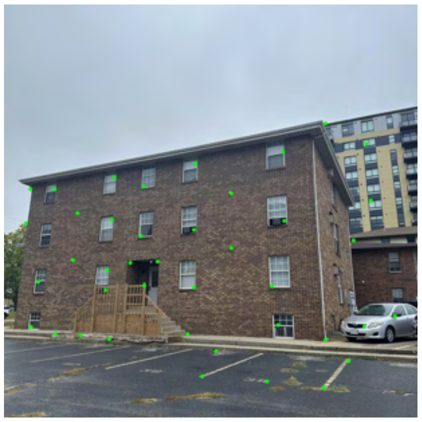



Harris Corner Points for Sigma = 1.2

Task 2 Results

Harris Corner Points for Sigma = 1.6

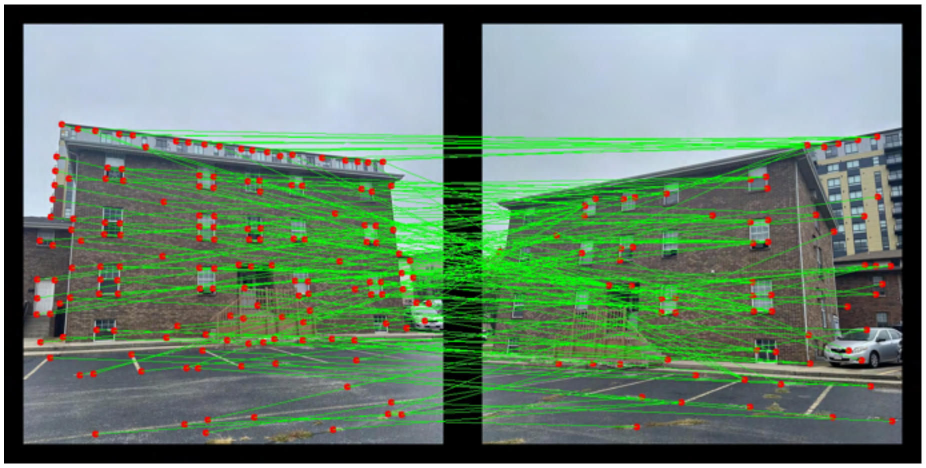



Harris Corner Points for Sigma = 2

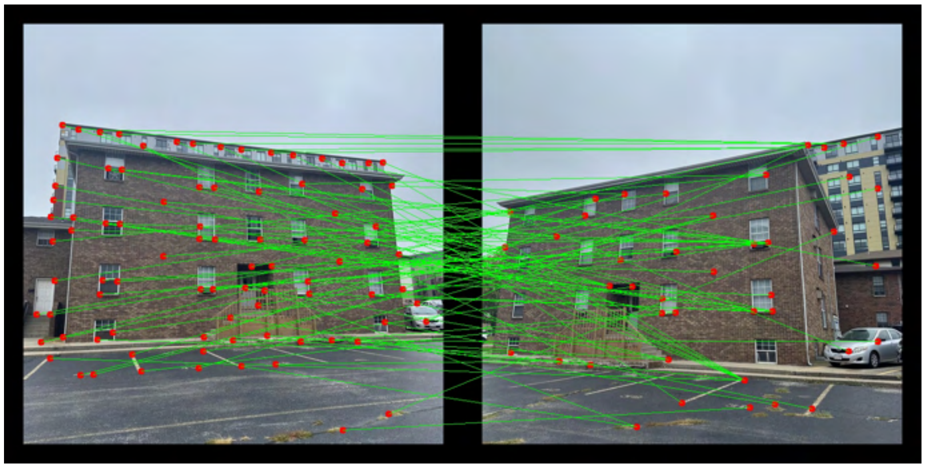
Task 2 Results

NCC Results - Task 2 My Image 1

In the images below I will display the NCC Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for My Image 1.

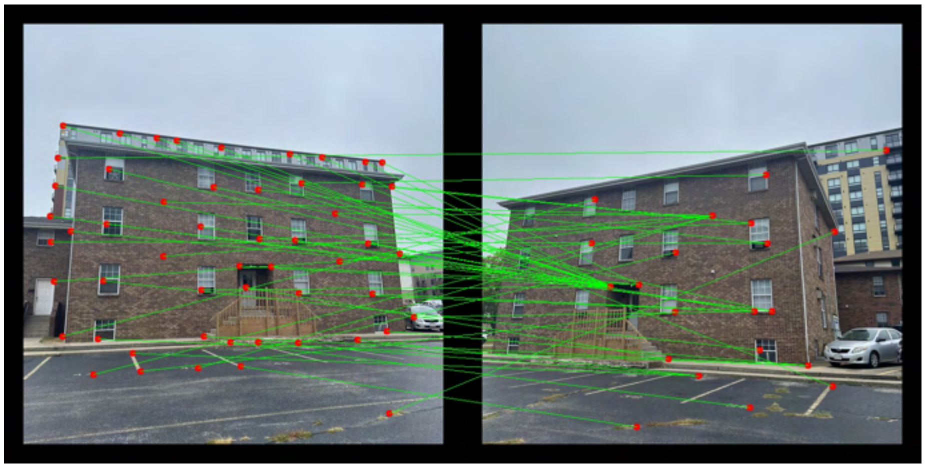


NCC Point Correspondences for Sigma = 1

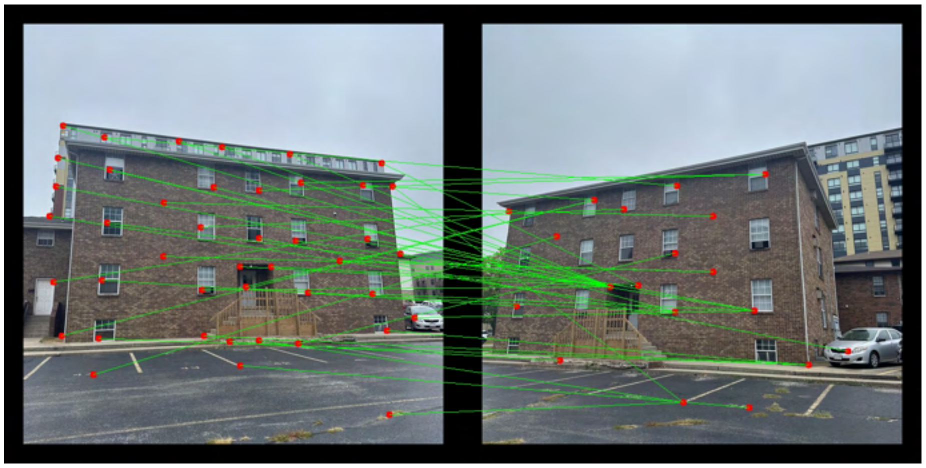


NCC Point Correspondences for Sigma = 1.2

Task 2 Results



NCC Point Correspondences for Sigma = 1.6

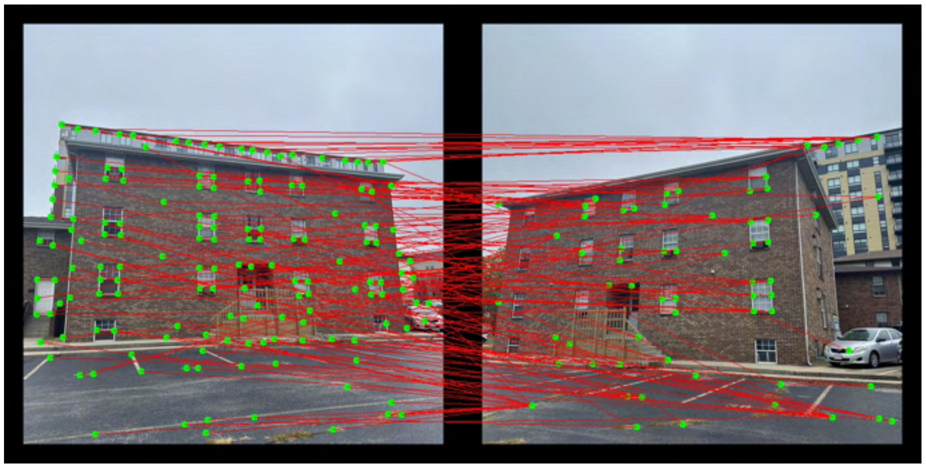


NCC Point Correspondences for Sigma = 2

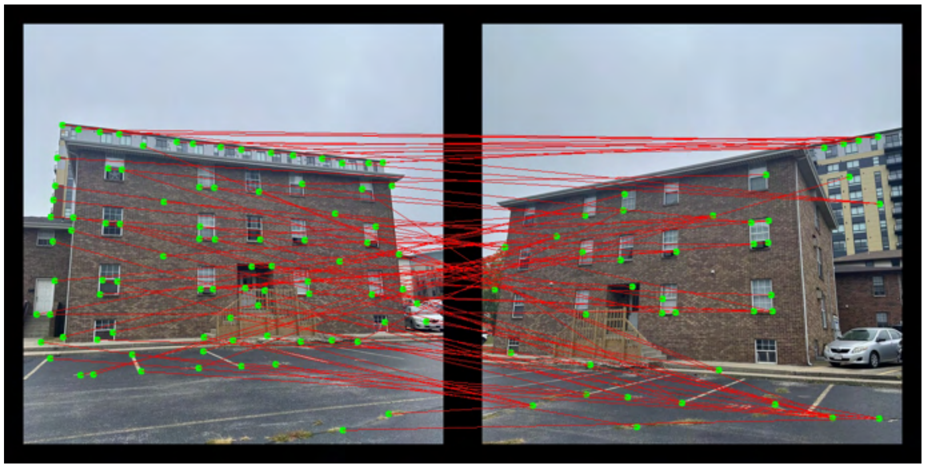
Task 2 Results

SSD Results - Task 2 My Image 1

In the images below I will display the SSD Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for My Image 1.

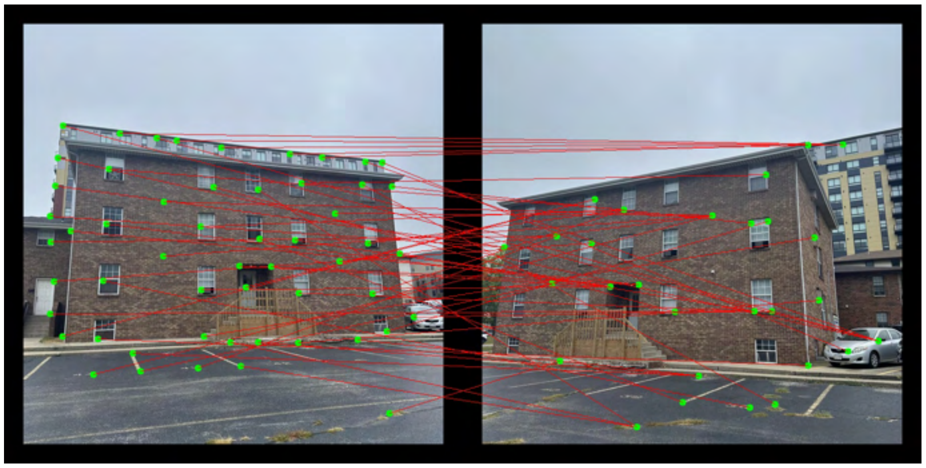


SSD Point Correspondences for Sigma = 1

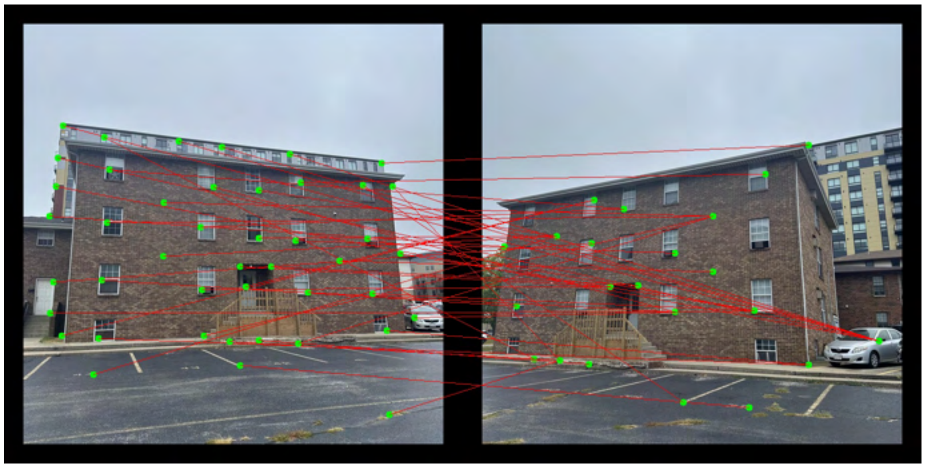


SSD Point Correspondences for Sigma = 1.2

Task 2 Results

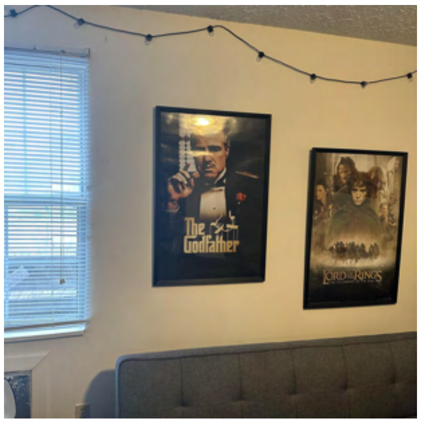


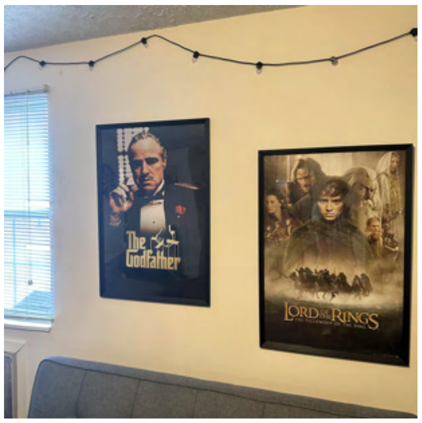
SSD Point Correspondences for Sigma = 1.6



SSD Point Correspondences for Sigma = 2

Task 2 Results



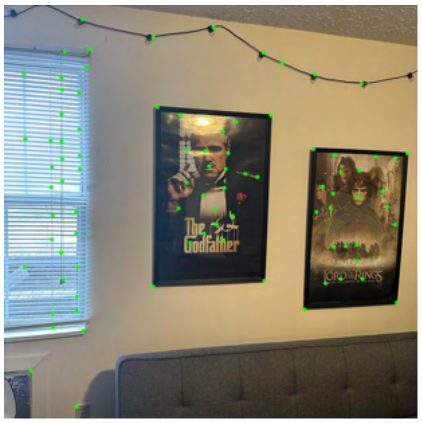
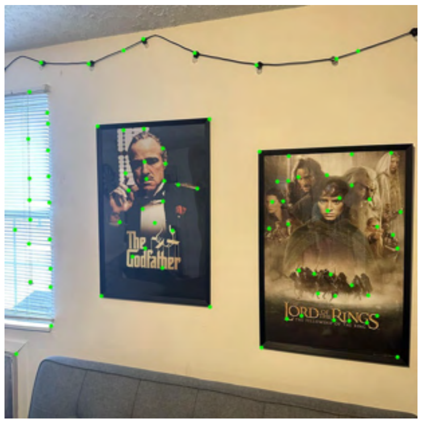


Original Images for My Image 2

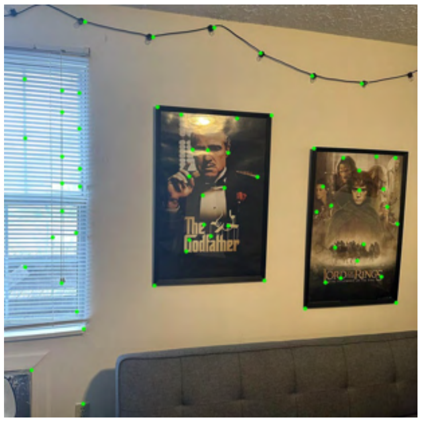
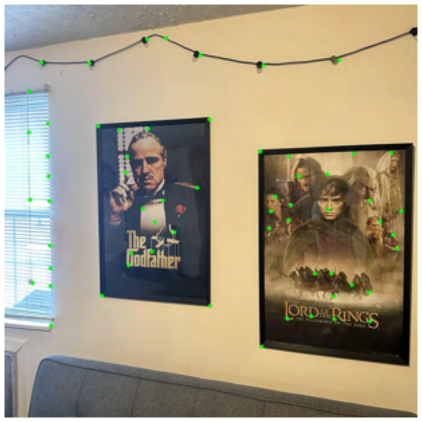
Harris Corner Points Results - Task 2 My Image 2

In the images below I will display what my Harris Corner Point Detector found for the sigma values of 1, 1.2, 1.6, 2 for My Image 2.

Task 2 Results

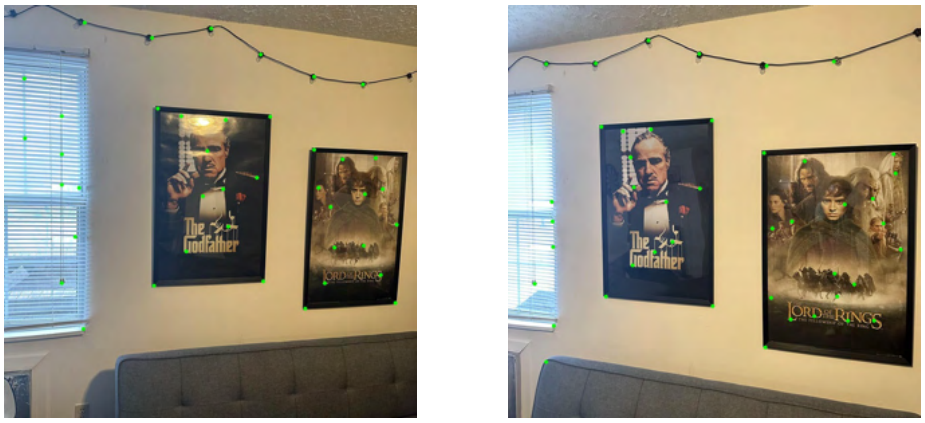



Harris Corner Points for Sigma = 1

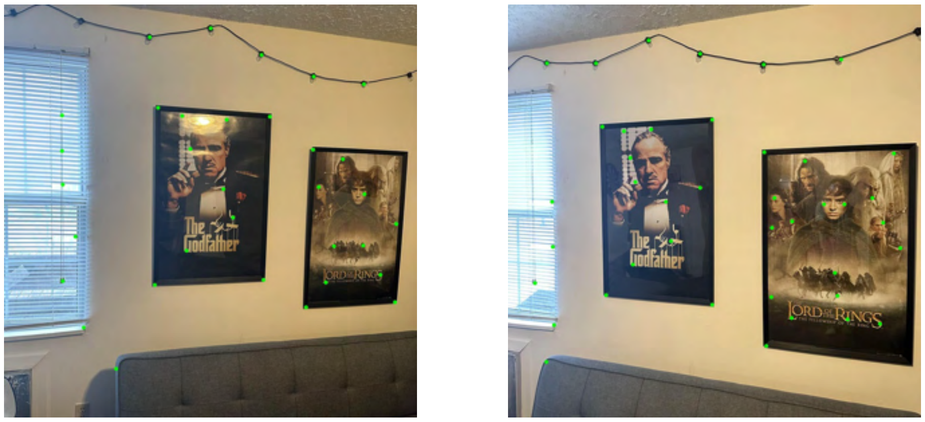



Harris Corner Points for Sigma = 1.2

Task 2 Results



Harris Corner Points for Sigma = 1.6

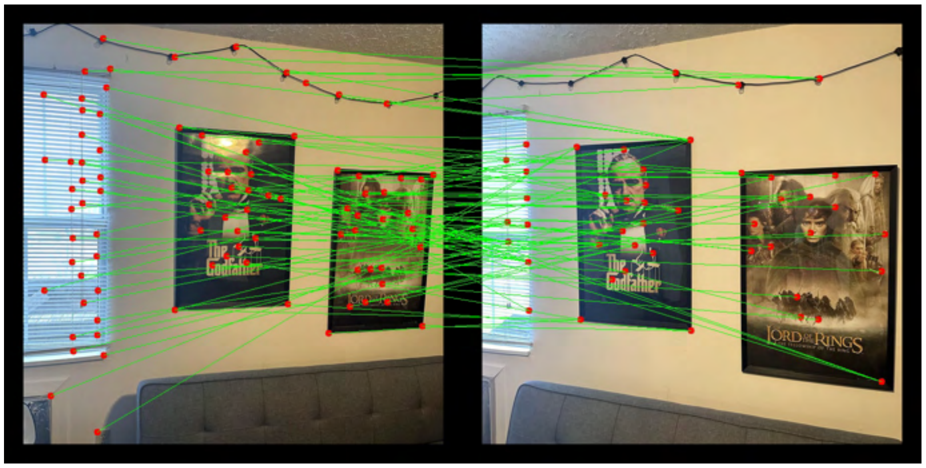


Harris Corner Points for Sigma = 2

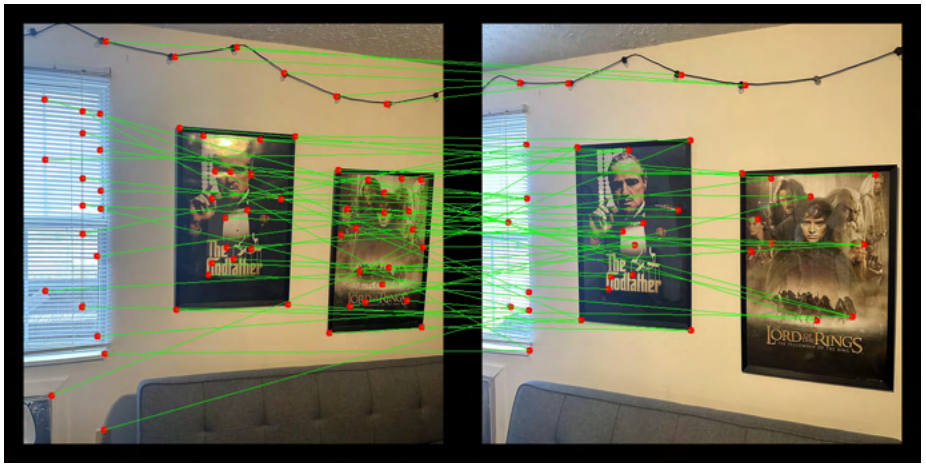
Task 2 Results

NCC Results - Task 2 My Image 2

In the images below I will display the NCC Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for My Image 2.

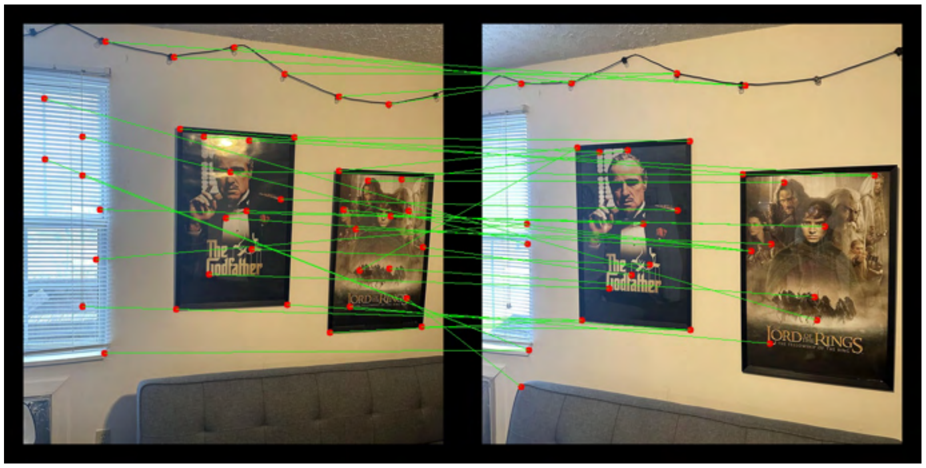


NCC Point Correspondences for Sigma = 1

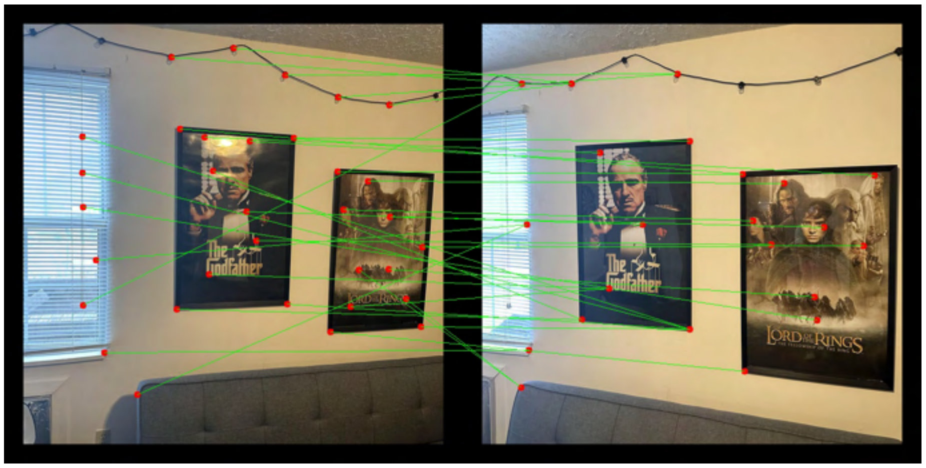


NCC Point Correspondences for Sigma = 1.2

Task 2 Results



NCC Point Correspondences for Sigma = 1.6

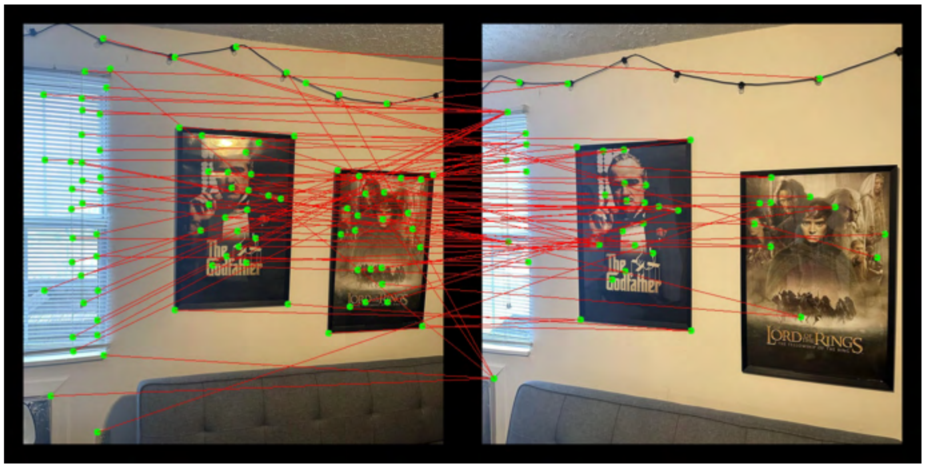


NCC Point Correspondences for Sigma = 2

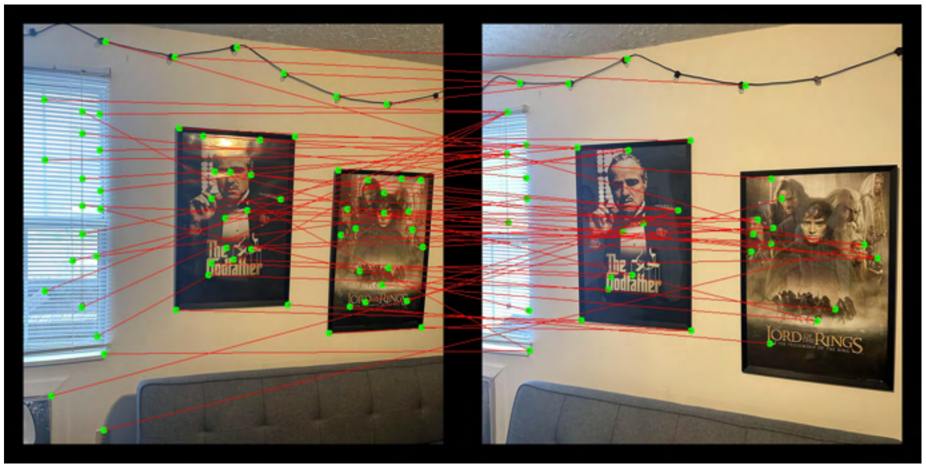
Task 2 Results

SSD Results - Task 2 My Image 2

In the images below I will display the SSD Point Correspondences for the sigma values of 1, 1.2, 1.6, 2 for My Image 2.

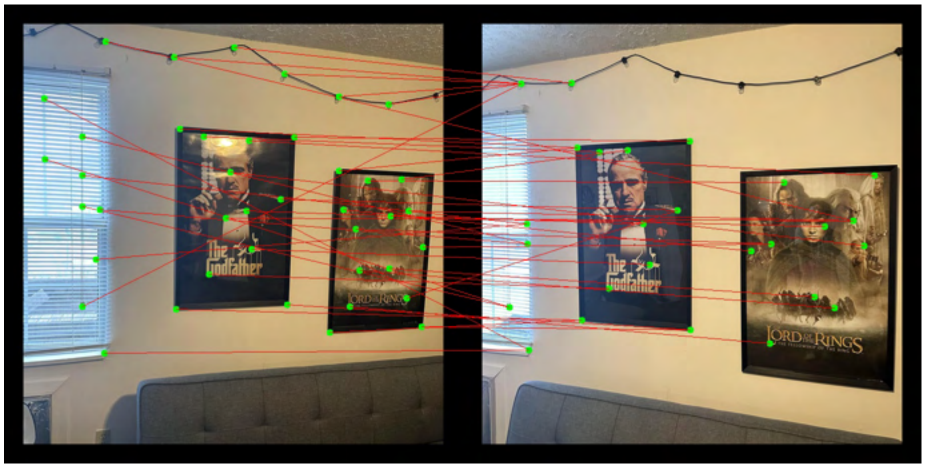


SSD Point Correspondences for Sigma = 1

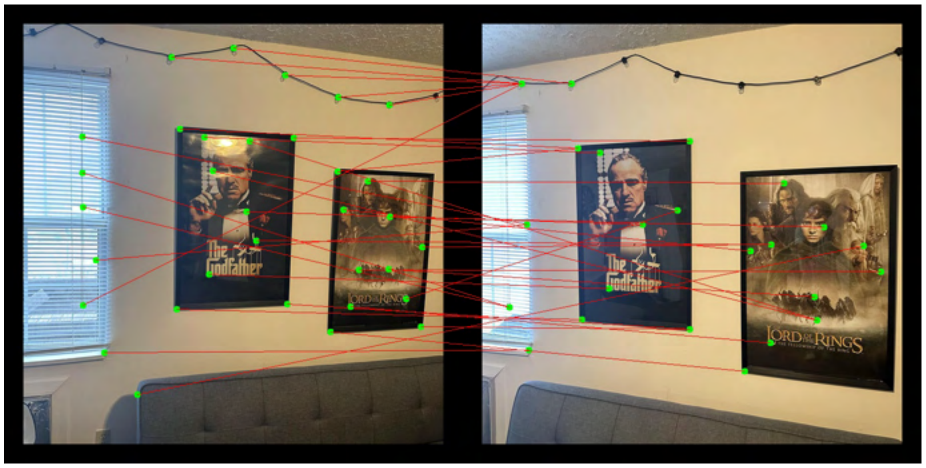


SSD Point Correspondences for Sigma = 1.2

Task 2 Results



SSD Point Correspondences for Sigma = 1.6

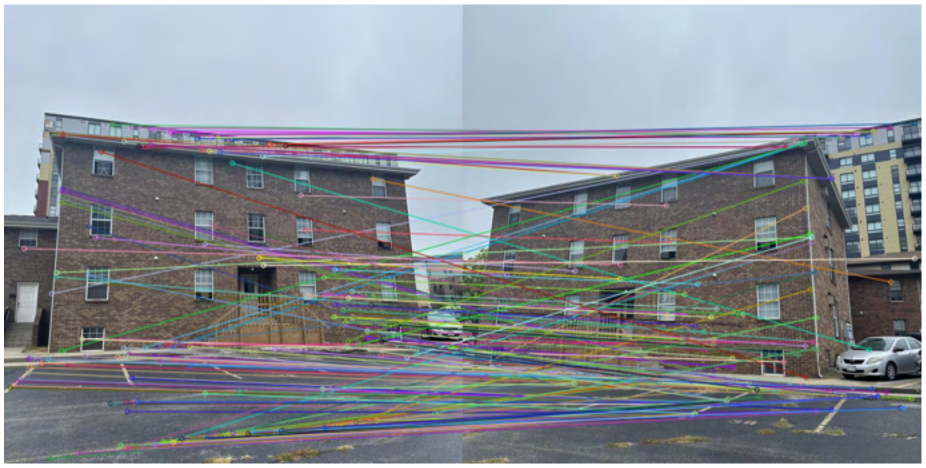


SSD Point Correspondences for Sigma = 2

Task 2 Results

SIFT Results - Task 2 My Image 1

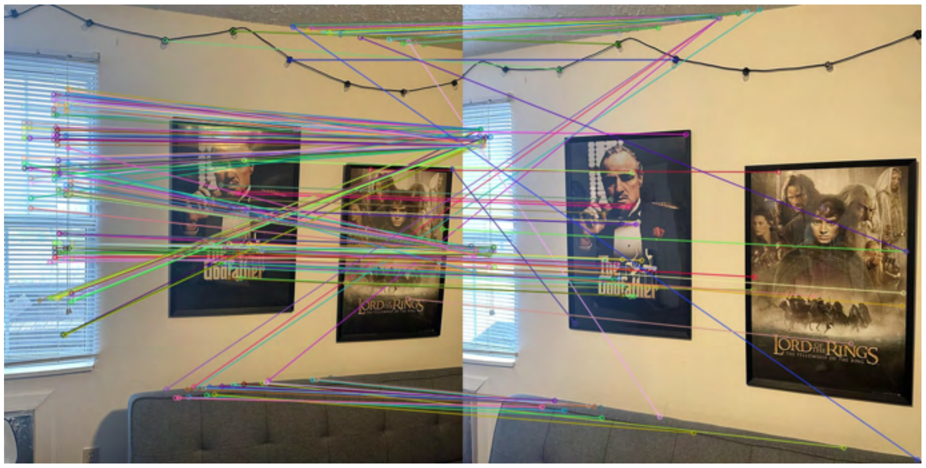
In the image below, I will display the SIFT algorithm result for My Image 1.



SIFT Result for My Image 1

SIFT Results - Task 2 My Image 2

In the image below, I will display the SIFT algorithm result for My Image 2.

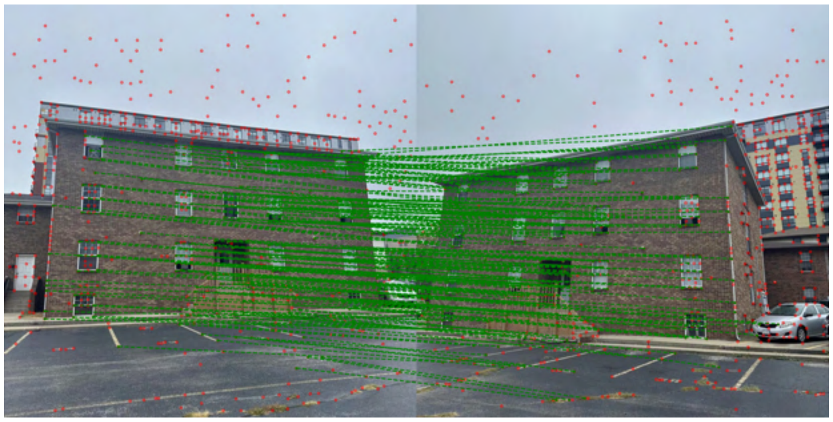


SIFT Result for My Image 2

Task 2 Results

SuperPoint and SuperGlue Results - Task 2 My Image 1

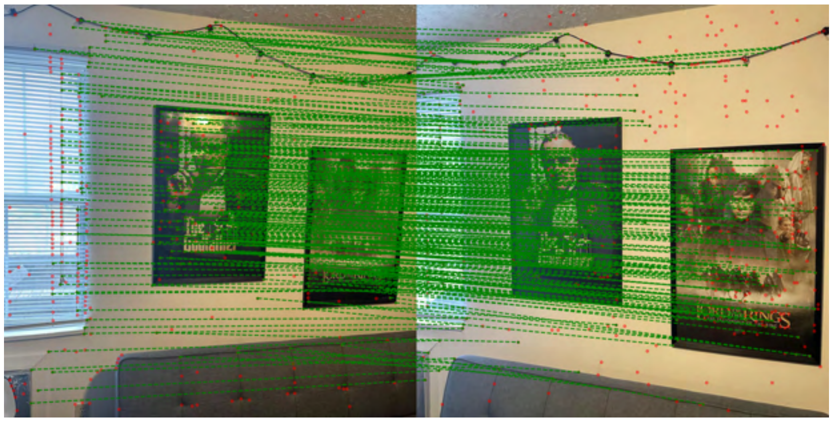
In the image below, I will display the Superpoint and Superglue result for the My Image 1.



Superpoint and Superglue Result for My Image 1

SuperPoint and SuperGlue Results - Task 2 My Image 2

In the image below, I will display the Superpoint and Superglue result for the My Image 2.



Superpoint and Superglue Result for My Image 2

Results Disccusion

Results Discussion

First I will talk about my results on the Harris Corner Detector. The most obvious observation is that as the omega value increased less amount of interest points were detected. In almost all of the images, you can see that at the lowest omega value of one, many interest points were being detected. The interest points being detected were very sensitive because they would find even the smallest change in grayscale to be a corner. Now the results shown have been through some point suppression, so you do not see all of the points, but this pattern of many points in the lower sigma values is still evident.

A side note concerning the images that I took. Initially, the resolution of the images I took was much greater than the images that were given to us in the assignment. When I put the higher-resolution image through my Harris Corner detector there were many more interest points than when I reduced the resolution to something similar to the given images. So, if the resolution of the image is higher then you can increase the sigma value and get a similar amount of interest points if you had a lower sigma value and lower resolution image. This conclusion in hindsight is not a surprise because in the Gaussian pyramid, we know that when you double the sigma the size of the image decreases by 2 fold.

The next point of discussion is the difference in finding corresponding points using SSD versus NCC. I observed that SSD yielded slightly better results than NCC, as there were fewer incorrect matches between points in the Hovde Image and My Image 1. However, the difference in quality between the two methods was less noticeable in the Temple Image and My Image 2. Despite SSD generally producing better results in the images used for the report, neither method was sufficient for accurately identifying corresponding points. In all the images there were portions of them where an object was not present in the other image, but there still were interest points being mapped to the other image. This was the largest cause of the false assignments of corresponding points. This problem is mostly fixed in the SIFT and SuperPoint and SuperGlue implementations.

The SIFT implementation was much better than the Harris Corner Detector. It eliminated many of the false corresponding point allocations which was caused because of the angle of the image having a portion of the building/object not in the other image. For example, looking at the SIFT Result of the Temple

Results Disccusion

Image we can see in the left image, half of the picture is not visible in the right image. So, the SIFT implementation will stop mapping interest points in that region which is a huge improvement from the Harris Corner Detection Method. Not only that it seems that the SIFT algorithm only maps one interest point to another, unlike the Harris Corner Detection method which had many one-to-many corner mappings. This highlights the limitation of SSD and NCC because they only look at the distance between points, which is not extensive enough of a mapping technique.

SIFT is more successful than Harris Corner Detection because it calculates a dominant local orientation, making it easier to map points under planar rotation. While Harris Corner Detection can successfully identify interest points when the correct sigma value is chosen, it struggles with accurate point mapping.

The final method implemented in this report is the SuperPoint and SuperGlue approach, which proved to be the most effective in identifying matching point pairs. To illustrate why this method excels, let's examine My Image 1. In previous implementations, including SIFT, the algorithms struggled to correctly match points on the building behind the brown building. Both Harris Corner Detection and SIFT mistakenly mapped points from the top of this building in the left image to the far right side of the same building in the right image. However, SuperPoint and SuperGlue recognized that the images were showing different parts of the building and correctly flagged these points in red, indicating no valid corresponding points. This level of accuracy sets SuperPoint and SuperGlue apart from the classical methods.

The best parameters for the best feature extraction and matching is very dependent on the image. However, I found general success with using the window size, M, equal to 29, a sigma value of 1.6, k value of 0.5 (for Harris Response Calculation function), and a non-max suppression window of 15*σ.

Source Code

```
import cv2
import numpy as np

def get_haar_wavelet_filter(sigma):

    #get M value for size of matrix
    M = int(np.ceil(4*sigma))
    if(M % 2 == 1):
        M += 1

    #create haar matrix for x
    haar_x_left = -np.ones((M, M // 2))
    haar_x_right = np.ones((M, M // 2))
    haar_x = np.concatenate((haar_x_left, haar_x_right), axis=1)

    #create haar matrix for y
    haar_y_top = np.ones((M //2 , M))
    haar_y_bottom = -np.ones((M //2 , M))
    haar_y = np.concatenate((haar_y_top, haar_y_bottom), axis=0)

    return haar_x, haar_y


def harris_corner_detection(img1, img2, sigma, file_name):

    #convert images to grayscale
    img1_grayscale = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) / 255
    img2_grayscale = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY) / 255

    #Use Haar Waveler Filter to get derivative
    haar_x, haar_y = get_haar_wavelet_filter(sigma)

    #convolve the haar matrix on each image
    img1_dx = cv2.filter2D(img1_grayscale,ddepth=-1,kernel=haar_x)
    img1_dy = cv2.filter2D(img1_grayscale,ddepth=-1,kernel=haar_y)

    img2_dx = cv2.filter2D(img2_grayscale,ddepth=-1,kernel=haar_x)
    img2_dy = cv2.filter2D(img2_grayscale,ddepth=-1,kernel=haar_y)

    #get values without the summation in c matrix
    img1_dx_2 = np.square(img1_dx)
    img1_dy_2 = np.square(img1_dy)
    img1_dx_dy = img1_dx * img1_dy

    img2_dx_2 = np.square(img2_dx)
    img2_dy_2 = np.square(img2_dy)
    img2_dx_dy = img2_dx * img2_dy

    #apply summation to the values in c matrix
    five_sigma = int(np.ceil(5*sigma))
    five_sigma_matrix = np.ones((five_sigma, five_sigma))

    img1_dx_2_summation = cv2.filter2D(img1_dx_2,ddepth=-1,kernel=five_sigma_matrix)
    img1_dy_2_summation = cv2.filter2D(img1_dy_2,ddepth=-1,kernel=five_sigma_matrix)
```

Source Code

```
img1_dx_dy_summation = cv2.filter2D(img1_dx_dy,ddepth=-1,kernel=five_sigma_matrix)

img2_dx_2_summation = cv2.filter2D(img2_dx_2,ddepth=-1,kernel=five_sigma_matrix)
img2_dy_2_summation = cv2.filter2D(img2_dy_2,ddepth=-1,kernel=five_sigma_matrix)
img2_dx_dy_summation = cv2.filter2D(img2_dx_dy,ddepth=-1,kernel=five_sigma_matrix)

#using the summation values you can find trace and determinant
img1_determiant = (img1_dx_2_summation*img1_dy_2_summation) - (img1_dx_dy_summation)**2
img1_trace = img1_dx_2_summation + img1_dy_2_summation

img2_determiant = (img2_dx_2_summation*img2_dy_2_summation) - (img2_dx_dy_summation)**2
img2_trace = img2_dx_2_summation + img2_dy_2_summation

#make Harris Response Calculation
k = 0.05 #picked the middle of allowed range
img1_R = img1_determiant - (k*(img1_trace**2))
img2_R = img2_determiant - (k*(img2_trace**2))

#now filter out the non-corner or weak corners by calcualted threshold
img1_threshold = np.mean(np.abs(img1_R))
img2_threshold = np.mean(np.abs(img2_R))

#Do no maximum suppression with area of five*sigma
img1_corner_list = []
img2_corner_list = []
five_sigma = five_sigma*3

for x in range(five_sigma, img1.shape[1] - five_sigma):
    for y in range(five_sigma, img1.shape[0] - five_sigma):
        img1_R_window = img1_R[y-five_sigma:y+five_sigma, x-five_sigma:x+five_sigma]
        # Check if the current response is the maximum and exceeds the threshold
        if img1_R[y, x] == img1_R_window.max() and img1_R[y,x] > img1_threshold:
            img1_corner_list.append((x, y))

for x in range(five_sigma, img2.shape[1] - five_sigma):
    for y in range(five_sigma, img2.shape[0] - five_sigma):
        img2_R_window = img2_R[y-five_sigma:y+five_sigma, x-five_sigma:x+five_sigma]
        # Check if the current response is the maximum and exceeds the threshold
        if img2_R[y, x] == img2_R_window.max() and img2_R[y,x] > img2_threshold:
            img2_corner_list.append((x, y))

# Convert the list to a NumPy array
img1_corner_list = np.array(img1_corner_list)
img2_corner_list = np.array(img2_corner_list)

#create image with all points
img1_output = np.copy(img1)
img2_output = np.copy(img2)

for x, y in img1_corner_list:
    cv2.circle(img1_output, (x, y), radius=3, color=(0, 255, 0), thickness=-1)
cv2.imwrite(file_name + '_' + str(sigma) + '_img1_points.jpeg', img1_output)

for x, y in img2_corner_list:
    cv2.circle(img2_output, (x, y), radius=3, color=(0, 255, 0), thickness=-1)
```

Source Code

```
    cv2.imwrite(file_name + '_' + str(sigma) + '_img2_points.jpeg', img2_output)

    return img1_corner_list, img2_corner_list


def harris_corner_detection_with_ncc(img1, img2, corner1, corner2, sigma, M, file_name):

    #convert images to grayscale
    img1_grayscale = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) / 255
    img2_grayscale = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY) / 255

    #add boarder to images
    img1_border = cv2.copyMakeBorder(img1_grayscale,M,M,M,M, cv2.BORDER_CONSTANT, value=0)
    img2_border = cv2.copyMakeBorder(img2_grayscale,M,M,M,M, cv2.BORDER_CONSTANT, value=0)

    point_pairs = []

    for window1 in corner1:
        distance_list  = np.zeros((len(corner2),2))
        for i, window2 in enumerate(corner2):
            #get window
            img1_window = img1_border[window1[1]+M-M//2 : window1[1]+M+M//2,
                                      window1[0]+M-M//2 : window1[0]+M+M//2]
            img2_window = img2_border[window2[1]+M-M//2 : window2[1]+M+M//2,
                                      window2[0]+M-M//2 : window2[0]+M+M//2]
            #get NCC
            mean_1 = np.mean(img1_window)
            mean_2 = np.mean(img2_window)
            distance_list[i] = np.array([(np.sum((img1_window - mean_1) *
                                                 (img2_window - mean_2))) /
                              (np.sqrt((np.sum((img1_window - mean_1) ** 2))
                                        * (np.sum((img2_window - mean_2) ** 2)))), i])

        #in all NCC find the largest one
        max_val = np.argmax(distance_list[:, 0], axis=0)
        point_pairs.append((window1, corner2[max_val]))

    #put points and lines on a picture
    img1_color = cv2.copyMakeBorder(img1,M,M,M,M,cv2.BORDER_CONSTANT, value=0)
    img2_color = cv2.copyMakeBorder(img2,M,M,M,M,cv2.BORDER_CONSTANT, value=0)
    image_combined = np.concatenate((img1_color, img2_color), axis=1)
    width_img1 = img1.shape[1]

    for i, j in point_pairs:
        img1_plot = (i[0] + M, i[1] + M)
        img2_plot = (j[0] + M*3 + width_img1, j[1] + M)

        # draw poins and lines
        cv2.circle(image_combined, img1_plot, radius=5, color=(0,0,255), thickness=-1)
        cv2.circle(image_combined, img2_plot, radius=5, color=(0,0,255), thickness=-1)
        cv2.line(image_combined, img1_plot, img2_plot, color=(0,255,0), thickness=1)

    cv2.imwrite(file_name + '_' + str(sigma) + '_img_ncc.jpeg', image_combined)

    return
```

Source Code

```python
def harris_corner_detection_with_ssd(img1, img2, corner1, corner2, sigma, M, file_name):

    #convert images to grayscale
    img1_grayscale = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY) / 255
    img2_grayscale = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY) / 255

    #add boarder to images
    img1_border = cv2.copyMakeBorder(img1_grayscale,M,M,M,M, cv2.BORDER_CONSTANT, value=0)
    img2_border = cv2.copyMakeBorder(img2_grayscale,M,M,M,M, cv2.BORDER_CONSTANT, value=0)

    point_pairs = []

    #loop thorugh all points and calculate the min distance
    for window1 in corner1:
        distance_list  = np.zeros((len(corner2),2))
        for j, window2 in enumerate(corner2):
            #get window
            img1_window = img1_border[window1[1]+M-M//2 : window1[1]+M+M//2,
                                      window1[0]+M-M//2 : window1[0]+M+M//2]
            img2_window = img2_border[window2[1]+M-M//2 : window2[1]+M+M//2,
                                      window2[0]+M-M//2 : window2[0]+M+M//2]
            #get SSD
            distance_list[j] = np.array([np.sum((img1_window - img2_window) ** 2), j])

        #in all the SSD find the smallest one
        min_distance = int(np.argmin(distance_list[:,0]))
        point_pairs.append((window1, corner2[min_distance]))

    #put points and lines on a picture
    img1_color = cv2.copyMakeBorder(img1,M,M,M,M,cv2.BORDER_CONSTANT, value=0)
    img2_color = cv2.copyMakeBorder(img2,M,M,M,M,cv2.BORDER_CONSTANT, value=0)
    image_combined = np.concatenate((img1_color, img2_color), axis=1)
    width_img1 = img1.shape[1]

    for i, j in point_pairs:
        img1_plot = (i[0] + M, i[1] + M)
        img2_plot = (j[0] + M*3 + width_img1,j[1] + M)

        #points and lines
        cv2.circle(image_combined, img1_plot, radius=5, color=(0,255,0), thickness=-1)
        cv2.circle(image_combined, img2_plot, radius=5, color=(0,255,0), thickness=-1)
        cv2.line(image_combined, img1_plot, img2_plot, color=(0,0,255), thickness=1)

    cv2.imwrite(file_name + '_' + str(sigma) + '_img_ssd.jpeg', image_combined)

    return

def SIFT(img1, img2, file_name):

    #convert images to grayscale
    img1_grayscale = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_grayscale = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    #make sift
```

Source Code

```
    sift = cv2.SIFT_create()

    #get keypoints and descriptors
    kp1, descriptors1 = sift.detectAndCompute(img1_grayscale, None)
    kp2, descriptors2 = sift.detectAndCompute(img2_grayscale, None)

    #create matcher
    bf = cv2.BFMatcher()

    #match descriptots and get lowest distances matches
    matches = matches = bf.match(descriptors1,descriptors2)
    matches = sorted(matches, key=lambda val: val.distance)

    #draw the first 50 matches
    out = cv2.drawMatches(img1, kp1, img2, kp2, matches[:200], None, flags=2)

    #save the image
    cv2.imwrite(file_name + '_SIFT.jpeg', out)

    return


def main():

    #load all the task 1 images
    temple_1 = cv2.imread('temple_1.jpg')
    temple_2 = cv2.imread('temple_2.jpg')
    hovde_1 = cv2.imread('hovde_2.jpg')
    hovde_2 = cv2.imread('hovde_3.jpg')

    ######## Harris Corner Detector #############

    #list of all 4 sigma values
    sigma_list = [1, 1.2, 1.6, 2]


    for i in sigma_list:
        # get corners
        temple_img1_points, temple_img2_points = harris_corner_detection(
            temple_1, temple_2, i, 'temple')
        hovde_img1_points, hovde_img2_points = harris_corner_detection(
            hovde_1, hovde_2, i, 'hovde')
        #ncc
        M = 40
        harris_corner_detection_with_ncc(temple_1, temple_2, temple_img1_points,
                                         temple_img2_points, i, M, 'temple')
        harris_corner_detection_with_ncc(hovde_1, hovde_2, hovde_img1_points,
                                         hovde_img2_points, i, M, 'hovde')
        #ssd
        harris_corner_detection_with_ssd(temple_1, temple_2, temple_img1_points,
                                         temple_img2_points, i, M, 'temple')
        harris_corner_detection_with_ssd(hovde_1, hovde_2, hovde_img1_points,
                                         hovde_img2_points, i, M, 'hovde')

    ####### SIFT ############################
```

Source Code

```
    SIFT(temple_1, temple_2, 'temple')
    SIFT(hovde_1, hovde_2, 'hovde')


    ######### TASK 2 ######################

    #load all the task 2 images
    img1_1 = cv2.imread('my_img1_1.jpg')
    img1_2 = cv2.imread('my_img1_2.jpg')
    img2_1 = cv2.imread('my_img2_1.jpg')
    img2_2 = cv2.imread('my_img2_2.jpg')

    for i in sigma_list:
        # get corners
        img1_1_points, img1_2_points = harris_corner_detection(
            img1_1, img1_2, i, 'my_img1')
        img2_1_points, img2_2_points = harris_corner_detection(
            img2_1, img2_2, i, 'my_img2')
        #ncc
        M = 30
        harris_corner_detection_with_ncc(img1_1, img1_2, img1_1_points,
                                         img1_2_points, i, M, 'my_img1')
        harris_corner_detection_with_ncc(img2_1, img2_2, img2_1_points,
                                         img2_2_points, i, M, 'my_img2')
        #ssd
        harris_corner_detection_with_ssd(img1_1, img1_2, img1_1_points,
                                         img1_2_points, i, M, 'my_img1')
        harris_corner_detection_with_ssd(img2_1, img2_2, img2_1_points,
                                         img2_2_points, i, M, 'my_img2')

    ####### SIFT ###########################
    SIFT(img1_1, img1_2, 'my_img1')
    SIFT(img2_1, img2_2, 'my_img2')



if __name__=="__main__":
    main()
```