ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

October 2, 2014

Contents

1 Introduction

In this assignment my own version of Harris corner detector will be implemented. The correspondences of interest points between two images (the same object with views from different angles) would be established based on SSD (Sum of Squared Differences) and NCC (Normalized Cross Correlation) method. Then, we will check the quality of Harris corner detector by applying the SIFT operator to the same sets of images.

Based on our experiment, it can be concluded although Harris corner detector can detect those obvious corners easily and accurately, it is not a good method when the features are not strict corners/more robust features.

It has been found in the experiment that the NCC based SIFT works better than anything else regarding those robust features. Please refer to figure 26 and figure 39 for great output from NCC based SIFT matching.

2 Harris Corner Detector

2.1 Overview of Harris Corner Detection

Before SIFT and SURF operators were invented, Harris Corner Detector was widely used in digital image interest points detection. The idea of Harris corner detection is based on that the characterization of a corner pixel should be invariant to rotations of images.

Although scale was not introduced when Harris corner detector was first introduced, we now can implement the Harris corner detector with variable scale.

2.2 Harris Corner Detector Implementation

1. We first need to calculate the gradient along x and y directions in the image. However, since we need to make our Harris corner detector scalable, we can not use Sobel Operator as Sobel Operator can not take care of scales properly. Instead, Haar Filter was implemented to replace Sobel Operator by finding the d_x and d_y. Below we will give an example of Haar filter with $\sigma = 1.2$.

$$Haar\ Filter\ for\ \frac{\partial}{\partial x},\ with\ \sigma = 1.2 : \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

$$Haar\ Filter\ for\ \frac{\partial}{\partial y},\ with\ \sigma = 1.2 : \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Please note the above form of Haar filter is based on the expansion of Haar wavelet at basic form. We have to make sure that the forms are scaled up to an M by M operator where M is the smallest even integer greater than $4 \times \sigma$. (Similarly we can easily prove that while $\sigma = 1.2$, M = 6. And when $\sigma = 1.4$, M = 8)

$\sigma = 1.4$.

$$Haar\ Filter\ for\ \frac{\partial}{\partial x},\ with\ \sigma = 1.4: \begin{bmatrix} -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1-1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$Haar\ Filter\ for\ \frac{\partial}{\partial y},\ with\ \sigma = 1.4: \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Furthermore, in order to minimize the noise in the pictures, before gradient calculation was performed, we also need to filter our images with Gaussian smoothing filter.

2. After the d_x and d_y was obtained, we then create a neighbourhood window of size $5\sigma \times 5\sigma$. Note that the σ should be consistent of that used in the first part when we are filtering the image using Haar filter. The C matrix could then be constructed:

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

3. While C in the previous step is a 2×2 matrix, we will first check the rank of $C_{i,j}$ at pixel location (i,j). As long as $rank(C) \neq 2$, we will remove the pixel locations from our candidates list of corners/interest points. **The computation efficiency could be improved significantly if we can first eliminate majority of candidates points.**

4. For the remaining corner candidates, we than need to determine the corner strength. Define

$$Conrner\ Response = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

While k is defined as a constant: 0.04, λ_1 and λ_2 is the eigenvalues of matrix C. Obviously if C is not rank 2 matrix the candidate point would not worth investigating. In order to simplify the calculation,

$$\det(C) = \lambda_1 \lambda_2$$

$$trace(C) = \lambda_1 + \lambda_2$$

therefore, SVD of matrix C would then not be required.

5. After corner response at each candidates pixel has been calculated, we then set up a threshold to filter out those points whose corner responses are not strong enough. However, in practical we will notice that even after threshold, at certain regions there would still be too many corner candidates. In order to solve the problem, we will perform non-maxima suppression to extract **only those points with local maxima values**.

6. Now all the Harris corner detection technique has been performed and we have certain amount of interest points. Save the interest points extracted from each images separately for corner correspondence estimation.

3 Establishing Correspondences Between Image Pairs for Harris Corner Detector

3.1 SSD: Sum of Squared Differences

In order to use SSD to establish the correspondences between interest points of an image pair, we first need to define a window $(M+1)\times(M+1)$. For the Harris corner detector, let $f_1(i,j)$ denote the pixel values in image 1 within the $(M+1)\times(M+1)$ window, and let $f_2(i,j)$ denote the pixel values in image 2 within the $(M+1)\times(M+1)$ window. Pairwise SSD is defined as:

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

3.2 NCC: Normalized Cross Correlation

Similarly as SSD, In order to use NCC to establish the correspondences between interest points of an image pair, we first need to define a window $(M+1)\times(M+1)$. For the Harris corner detector, let $f_1(i,j)$ denote the pixel values in image 1 within the $(M+1)\times(M+1)$ window, and let $f_2(i,j)$ denote the pixel values in image 2 within the $(M+1)\times(M+1)$ window. Pairwise NCC is defined as:

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - \mu_1)(f_2(i,j) - \mu_2)}{\sqrt{[\sum_i \sum_j (f_1(i,j) - \mu_1)^2][\sum_i \sum_j (f_2(i,j) - \mu_2)^2]}}$$

while μ_1 is the mean of window $f_1(i,j)$ and μ_2 is the mean of window $f_2(i,j)$.

3.3 False Matching Elimination

In general, a lot of pairs were matched incorrectly if we do not have any systematic way to avoid/reduce false matching.

SSD Case: As SSD is defined as the sum of squared errors, an ideal match would obviously have SSD = 0. However, based on our practical experiment we know that is almost impossible. Hence we use the following method to reduce/avoid false matching.

1. If SSD value of a certain pair is smaller than 5 $\times$ (the absolute minima values of SSD across all SSD matrix, we proceed, otherwise will dump the point. **This step will actually dump a lot of good candidates**.

2. If the $\frac{minimum\ of\ SSD}{second\ minimum\ of\ SSD}$ is smaller than a certain ratio (denoted as Rssd), we will establish correspondence between this specific pair. Otherwise we will again dump the point as candidate.

NCC Case: As NCC is defined as the normalized cross correlation, an ideal match would obviously have NCC = 1. However, based on our practical experiment we know that is almost impossible. Hence we use the following method to reduce/avoid false matching.

1. If NCC value of a certain pair is smaller than 0.9 × (the absolute maxima values of NCC across all SSD matrix, we proceed, otherwise will dump the point. **This step will actually dump a lot of good candidates**.
2. If the $\frac{maxima\ of\ NCC}{second\ maxima\ of NCC}$ is larger than a certain ratio (denoted as Rncc), we will establish correspondence between this specific pair. Otherwise we will again dump the point as candidate.
3. Of course, if NCC value is negative, which mean two pixel is anti-correlated, they can not be a pair.

3.4 Parameters Table For Harris Corner Detector

Image	W_{Haar}	W_{SSD}	W_{NCC}	TH_{SSD}	TH_{NCC}	R_{SSD}	R_{NCC}
pic1.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.85	1.1
pic2.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.85	1.1
pic6.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.8	1.01
pic7.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.8	1.01
my1.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.85	1.01
my2.jpg	$5\sigma \times 5\sigma$	$10\sigma \times 10\sigma$	$10\sigma \times 10\sigma$	$\leq 40 \times SSD_{Minima}$	$\geq 0.3 \times NCC_{Maxima}$	0.85	1.01

Note that the threshold values for corner responses is defined as:

$$(CR1_{Maxima} + CR2_{Maxima})/20$$

($CR1_{Maxima}$ is the Maxima for Corner Response of Image 1

($CR2_{Maxima}$ is the Maxima for Corner Response of Image 2

4 SIFT Algorithm: Scale Invariant Feature Transform

For SIFT algorithm, we first need to find all the local extrema from the DoG pyramid. Note that extrema include both maxima and minima. To be more detailed, each point in the DoG pyramid should be compared to:

1. 8 points in the 3 by 3 neighbourhood at the same scale
2. 9 points in the 3 by 3 neighbourhood at the next scale
3. 9 points in the 3 by 3 neighbourhood at the previous scale

Usually, those points in original image that the grey levels change rapidly in several directions are likely to be the DoG extrema.

In order to locate the extrema in the sub-pixel accuracy, we need to estimate the second-order derivatives of $D(x, y, \sigma)$ at the sampling points in the DoG pyramid. First, find the Taylor series expansion of $D(x, y, \sigma)$ in the vicinity of $\vec{x_0} = (x_0, y_0, \sigma_0)^T$:

$$D(\vec{x}) \approx D(\vec{x_0}) + J^T(\vec{x_0})\vec{x} + \frac{1}{2}\vec{x}^T H(\vec{x_0})\vec{x}$$

where $\vec{x}$ is a incremental of $\vec{x_0}$.

Easily, J is the gradient vector estimated at $\vec{x}_0$:

$$J(\vec{x}_0) = (\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial \sigma})^T|_{\vec{x}_0}$$

And Hessian matrix is:

$$H(\vec{x}_0) = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial \sigma} \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial \sigma} \\ \frac{\partial^2 D}{\partial \sigma \partial x} & \frac{\partial^2 D}{\partial \sigma \partial y} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix}$$

For the true locations of extrema:

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

As the extrema points are found, we need to threshold out those extremas who are week. For example we can set a hard cut off at:

$$D(\vec{x}) \geq 0.03$$

to be qualified for an extrema candidate.

After the candidates of the local extrema are found, we then need to establish the dominant local orientation for each candidate point found in previous step. To find the local dominant orientation we need to calculate the gradient vector of the Gaussian-smoothed image $f(x, y, \sigma)$ at the scale σ of the extrema. It magnitude is defined as:

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

While the orientation is:

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

Finally, we divide the 16 by 16 neighbourhood of point into 4 by 4 cells (each cell with 4 by 4 points and totally we have 16 cells). Now, for each of the cell, an 8-bin orientation histogram is calcualted from the gradient-magnitude-weighted values of $\theta(x, y)$ at 16 pixels. That is, total of $8 \times 16 = 128$. Hence,for each interest point, we will have 128-element descriptor.

In next section, we will explain how to establish correspondences based on features extracted by SIFT operator

5 Establishing Correspondences Between Image Pairs for SIFT

5.1 SSD: Sum of Squared Differences

In order to use SSD to establish the correspondences between interest points of an image pair yield by SIFT, we need:

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

While $f_1(i,j)$ is the 128-elements descriptor obtained at each interest points location.

5.2 NCC: Normalized Cross Correlation

Similarly as SSD, In order to use NCC to establish the correspondences between interest points of an image pair, we need :

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - \mu_1)(f_2(i,j) - \mu_2)}{\sqrt{[\sum_i \sum_j (f_1(i,j) - \mu_1)^2][\sum_i \sum_j (f_2(i,j) - \mu_2)^2]}}$$

While $f_1(i,j)$ is the 128-elements descriptor obtained at each interest points location.

5.3 False Matching Elimination

In general, a lot of pairs were matched incorrectly if we do not have any systematic way to avoid/reduce false matching.

SSD Case: As SSD is defined as the sum of squared errors, an ideal match would obviously have SSD = 0. However, based on our practical experiment we know that is almost impossible. Hence we use the following method to reduce/avoid false matching.

1. If SSD value of a certain pair is smaller than 5 $\times$ (the absolute minima values of SSD across all SSD matrix, we proceed, otherwise will dump the point. **This step will actually dump a lot of good candidates**.
2. If the $\frac{minimum\ of\ SSD}{second\ minimum\ of\ SSD}$ is smaller than a certain ratio (denoted as Rssd), we will establish correspondence between this specific pair. Otherwise we will again dump the point as candidate.

Euclidean Distance Case: Euclidean distance case is almost identical as SSD, the only difference is $Euclidean\ Distance = \sqrt{SSD}$

NCC Case: As NCC is defined as the normalized cross correlation, an ideal match would obviously have NCC = 1. However, based on our practical experiment we know that is almost impossible. Hence we use the following method to reduce/avoid false matching.

1. If NCC value of a certain pair is smaller than 0.9 $\times$ (the absolute maxima values of NCC across all SSD matrix, we proceed, otherwise will dump the point. **This step will actually dump a lot of good candidates**.

2. If the $\frac{maxima\ of\ NCC}{second\ maxima\ of NCC}$ is larger than a certain ratio (denoted as Rncc), we will establish correspondence between this specific pair. Otherwise we will again dump the point as candidate.

3. Of course, if NCC value is negative, which mean two pixel is anti-correlated, they can not be a pair.

5.4 Parameters Table For SIFT

Image	$R_{Euclidean}$	R_{SSD}	R_{NCC}
pic1.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$
pic2.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$
pic6.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$
pic7.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$
my1.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$
my2.jpg	$\leq 5 \times Euclidean_{Minima}$	$\leq 5 \times SSD_{Minima}$	$\geq 0.9 \times NCC_{Maxima}$

Based on our empirical data, dynamic threshold method works great for SIFT.

6 Dynamic Threshold for Euclidean Distance, SSD and NCC

It has already proven useful and convenient in this experiment using dynamic threshold. The idea of dynamic threshold is to avoid manually change each threshold value for every single experiment. Because the threshold values would vary hugely based on the image quality, illumination, feature descriptors strength, corner response, etc.

For example, in order to qualify for a match SSD value should be at least smaller than 5 times the smallest SSD value. Or, similarly, in order to qualify for a match NCC value should be at least larger than 0.9 times the largest NCC value.

Great Result: NCC for SIFT

From the experiment, we have actually concluded that when the images/features are robust, NCC based on SIFT features can still work great. **For more details please refer to each of the conclusion/discussion session in next section and figure 26, figure 39.**

7 Results: Very Important Conclusions At End of Each Subsections

7.1 Set 1: Harris Operator/SIFT Comparison

Figure 1. Set1: pic1.jpg

Figure 2. Set1: pic2.jpg

Figure 3. Harris: The SSD matching with $\sigma = 0.6$

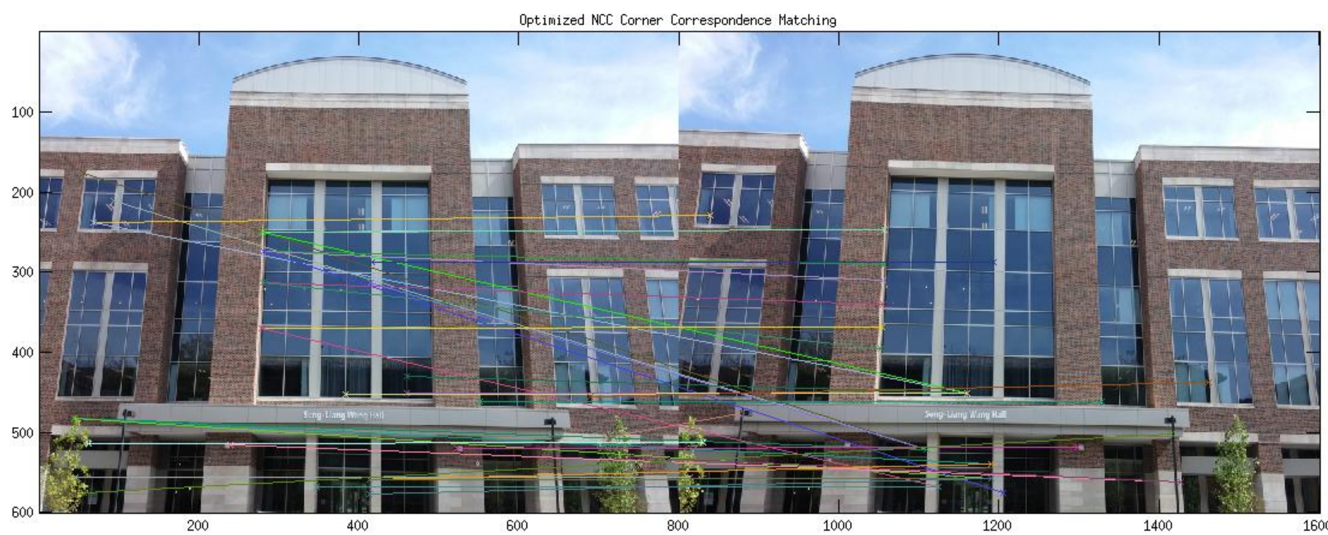

Figure 4. Harris: The NCC matching with $\sigma = 0.6$

Figure 5. Harris: The SSD matching with $\sigma = 1$

Figure 6. Harris: The NCC matching with $\sigma = 1$

Figure 7. Harris: The SSD matching with $\sigma = 1.4$

Figure 8. Harris: The NCC matching with $\sigma = 1.4$

Figure 9. Harris: The SSD matching with $\sigma = 2.2$

Figure 10. Harris: The NCC matching with $\sigma = 2.2$

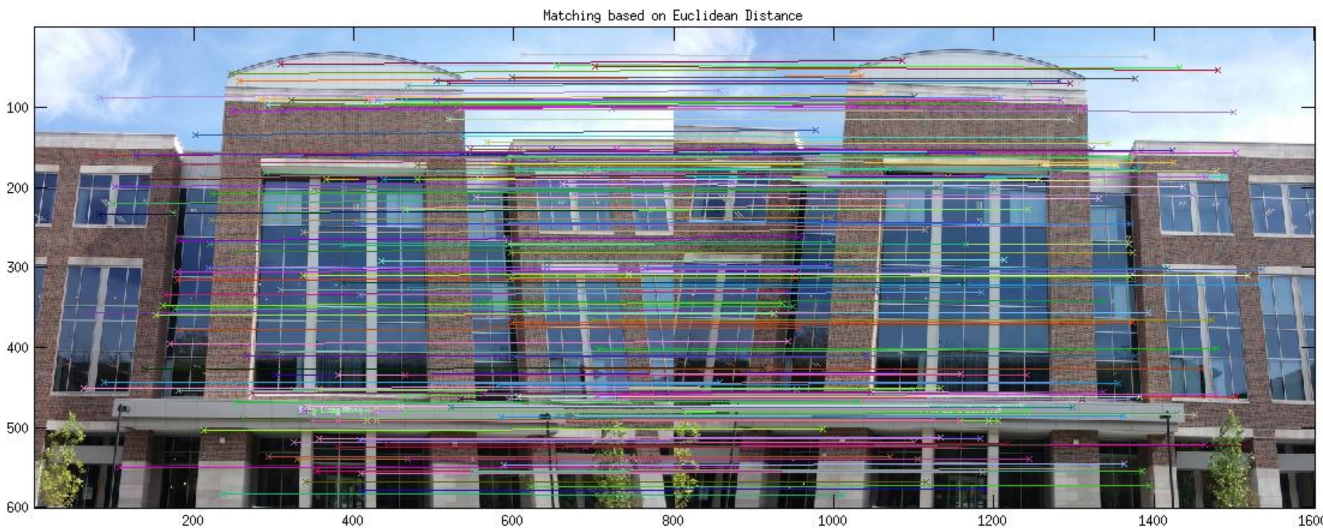

Figure 11. SIFT: Interest Points Matching Based on Euclidean Distance

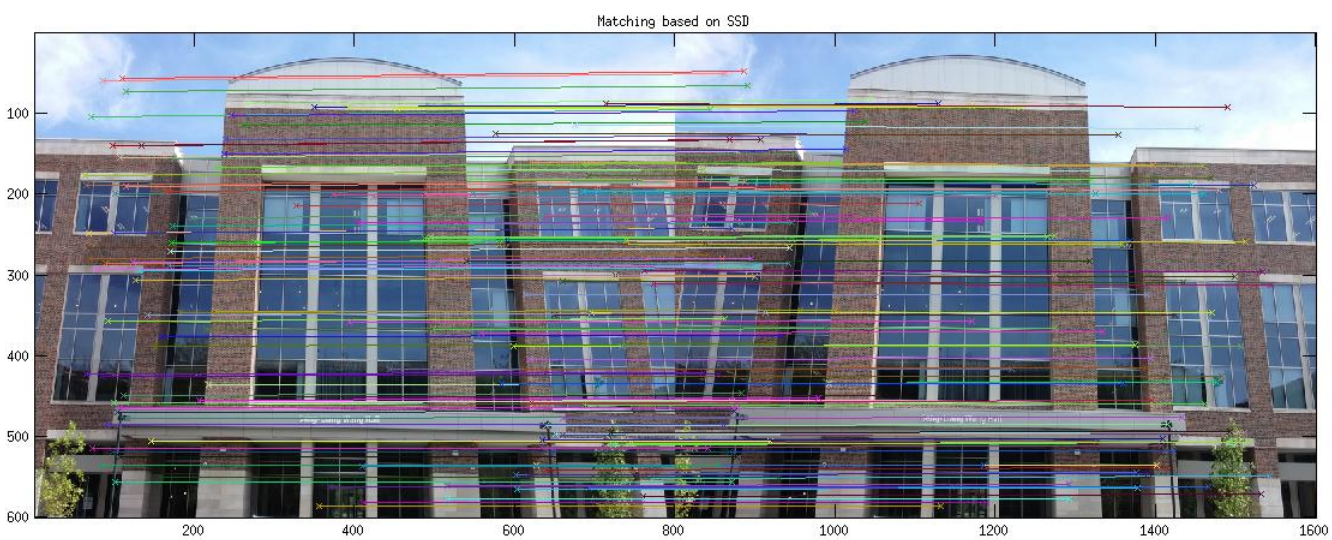

Figure 12. SIFT: Interest Points Matching Based on SSD

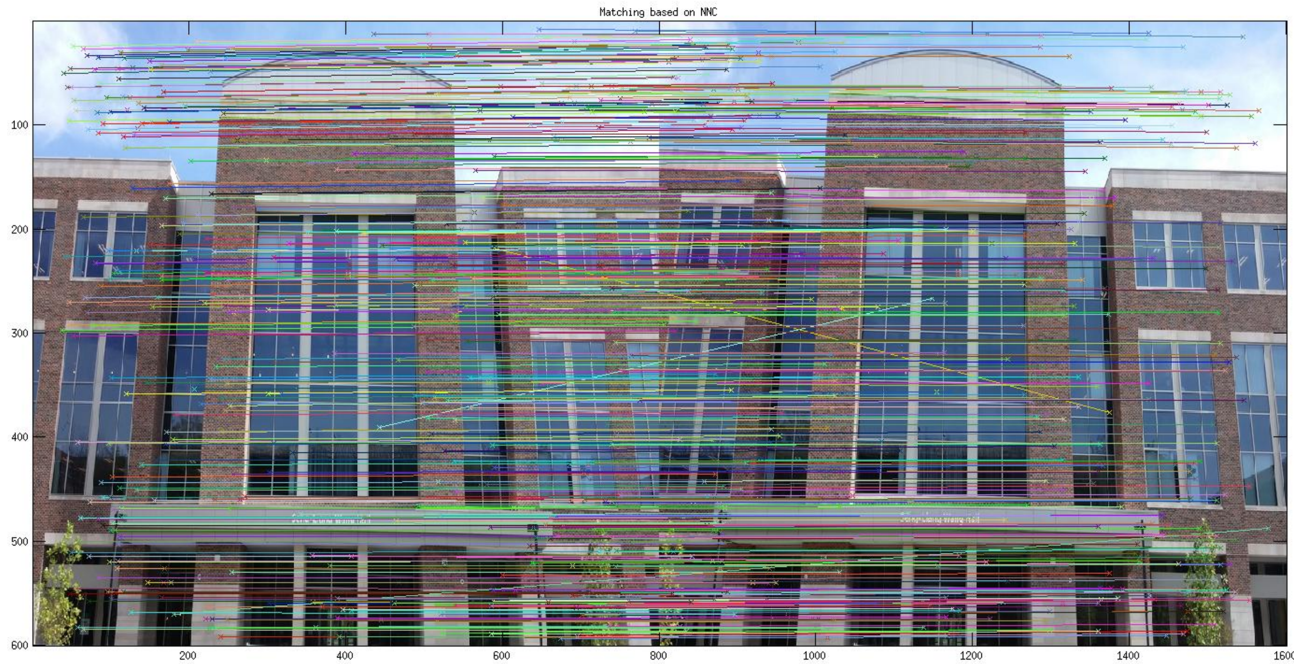

Figure 13. SIFT: Interest Points Matching Based on NCC

Conclusion for Set 1:

For this set of images as the view angle (also the lighting conditions, color saturation, etc) didn't change that much, Harris corner detector works pretty well. For the correspondences established based on SSD and NCC, except for a very few mismatch the overall correct matching rate is very high.

It can also be concluded that larger the σ, less sensitive the Harris Corner detector is (less interest points is not necessarily bad). Those points in the lower part of the image could always be detected by Harris Corner detector. As the Harris Corner detector already work pretty well, SIFT operator would not improve our result that much. (of course we will easily have a lot more interest points).

7.2 Set 2: Harris Operator/SIFT Comparison

Figure 14. Set2: pic6.jpg

Figure 15. Set1: pic7.jpg

Figure 16. Harris: The SSD matching with $\sigma = 0.6$

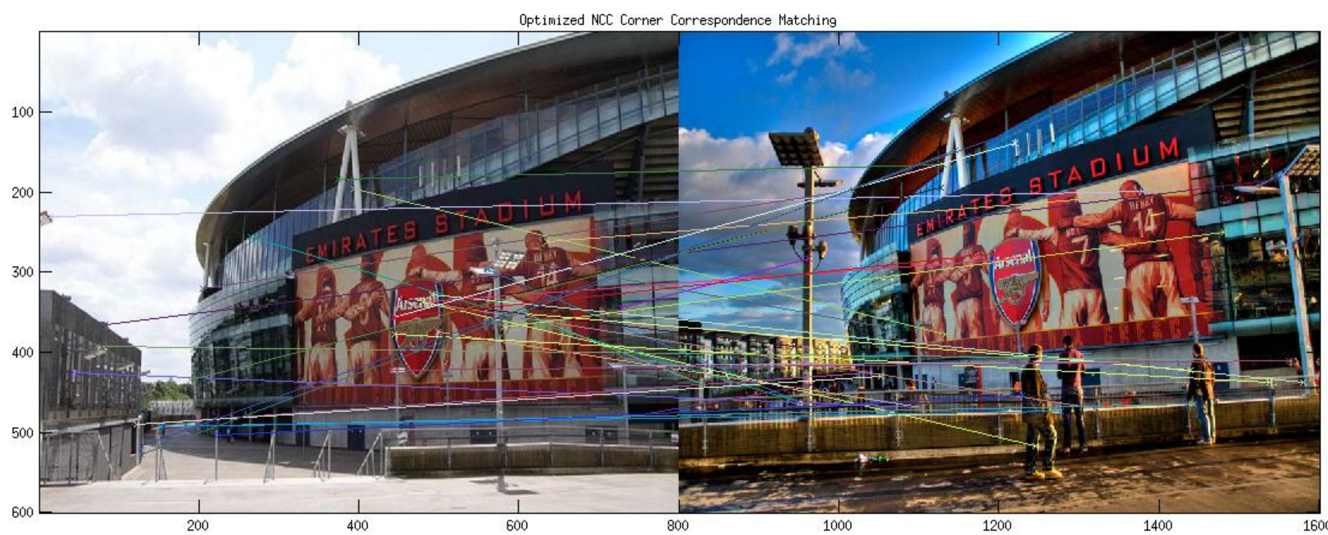

Figure 17. Harris: The NCC matching with $\sigma = 0.6$

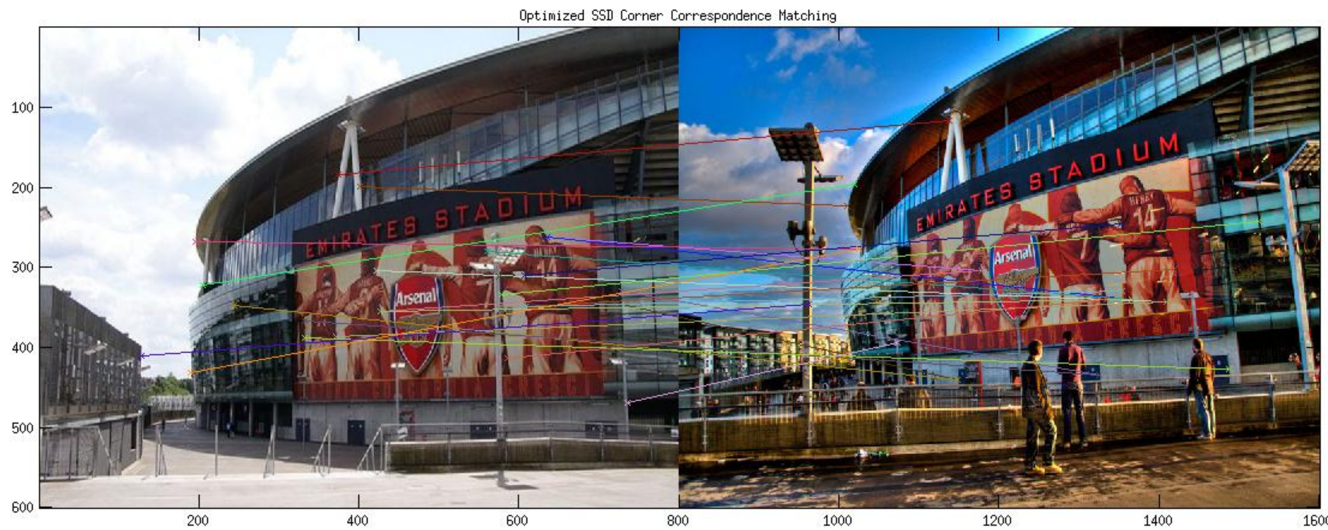

Figure 18. Harris: The SSD matching with $\sigma = 1$

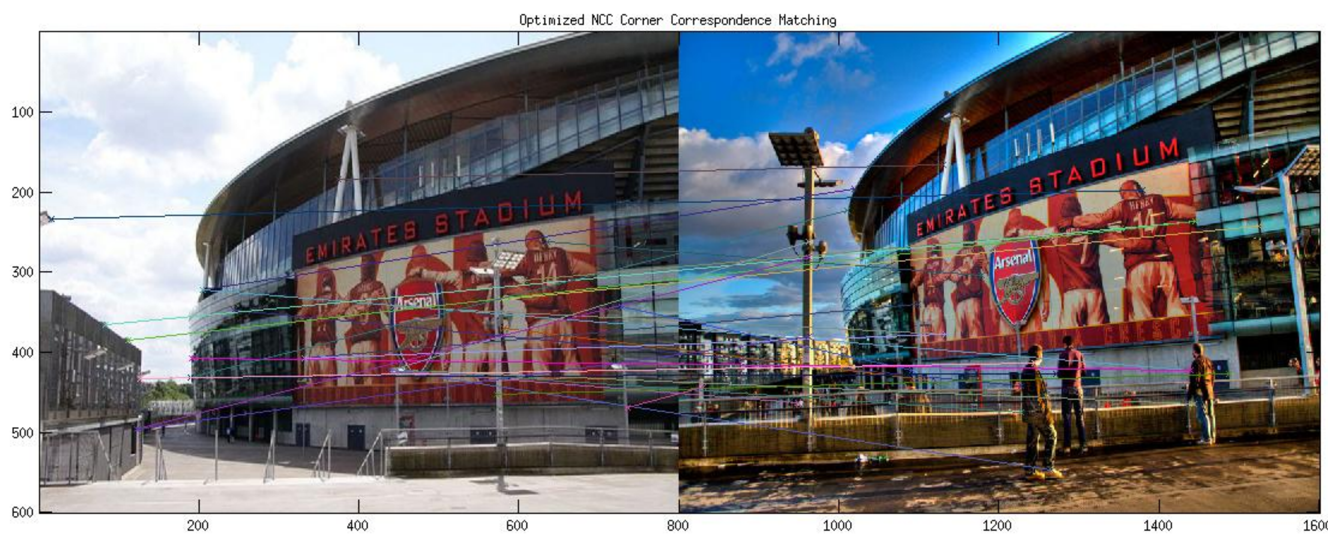

Figure 19. Harris: The NCC matching with $\sigma = 1$

Figure 20. Harris: The SSD matching with $\sigma = 1.4$

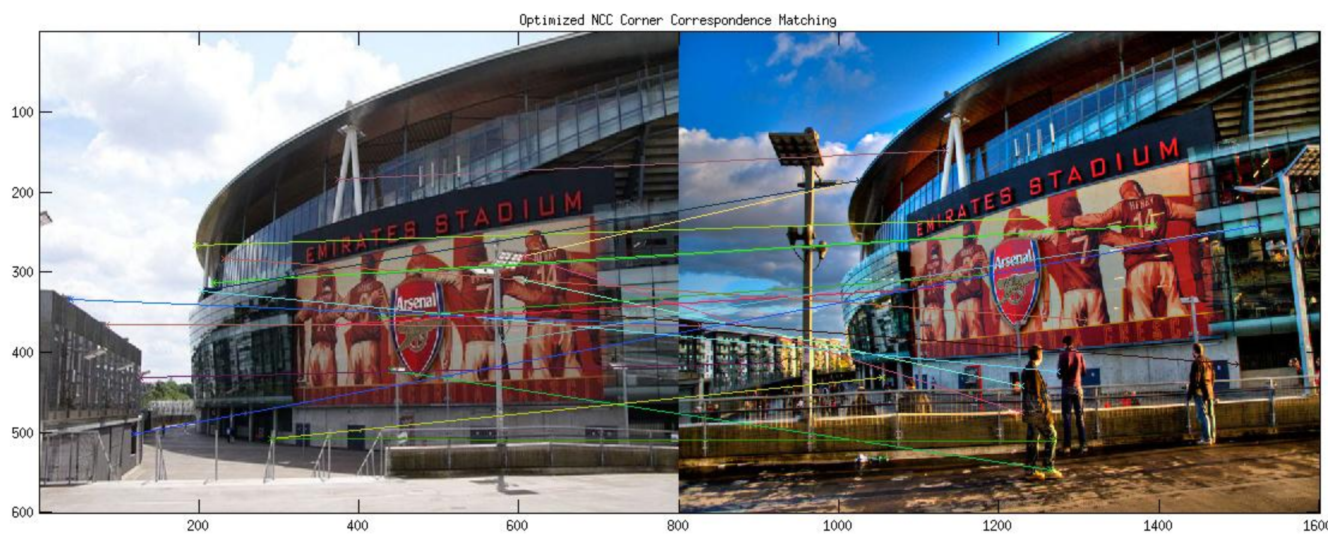

Figure 21. Harris: The NCC matching with $\sigma = 1.4$

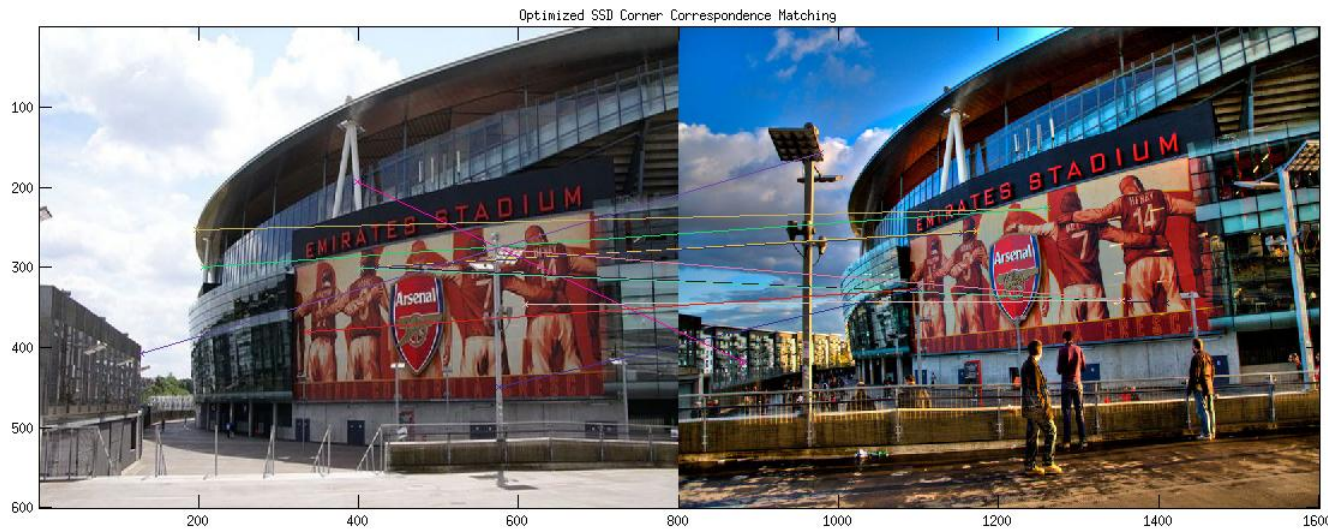

Figure 22. Harris: The SSD matching with $\sigma = 2.2$

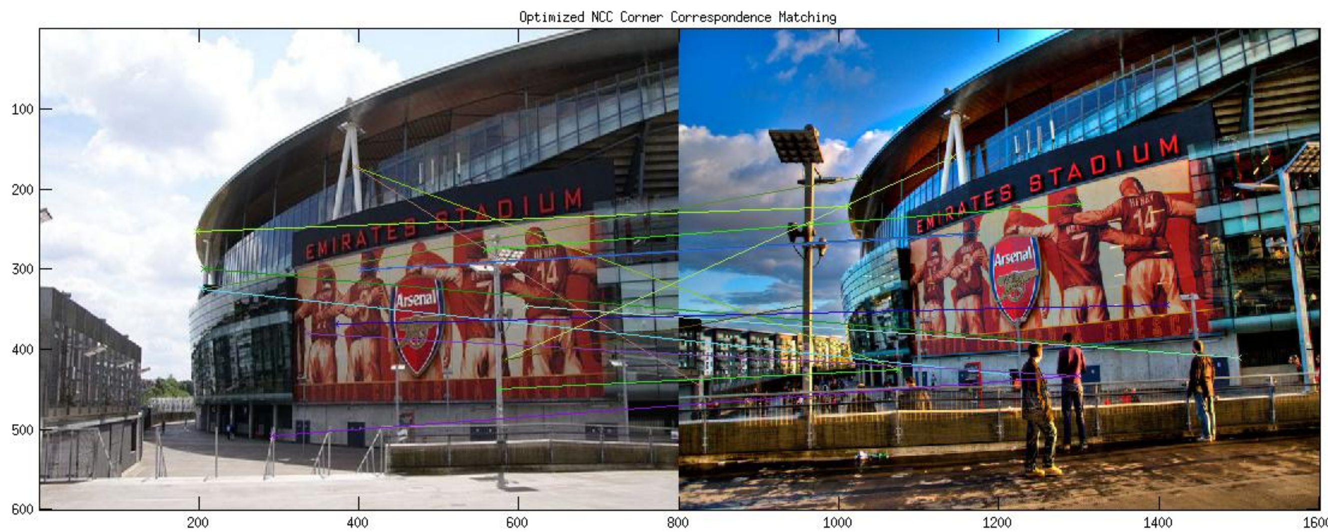

Figure 23. Harris: The NCC matching with $\sigma = 2.2$

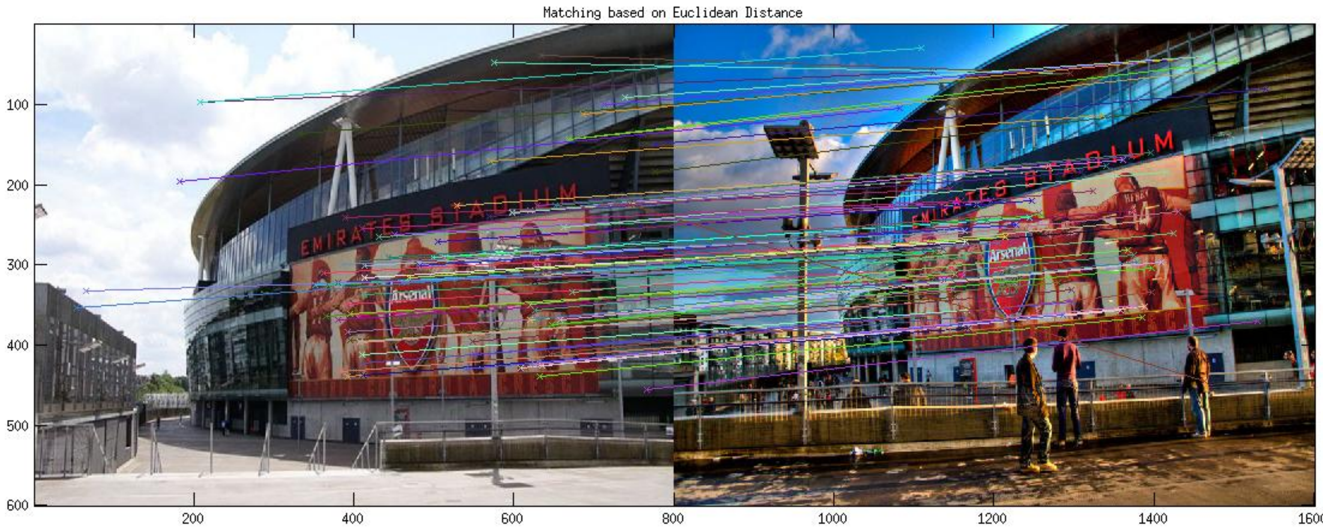

Figure 24. SIFT: Interest Points Matching Based on Euclidean Distance

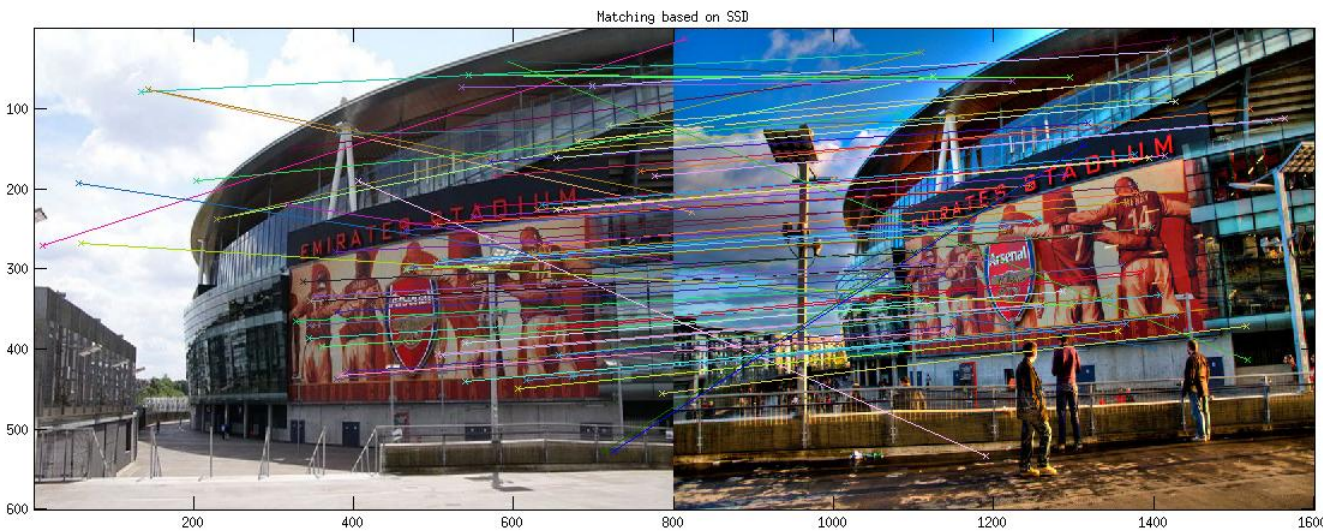

Figure 25. SIFT: Interest Points Matching Based on SSD

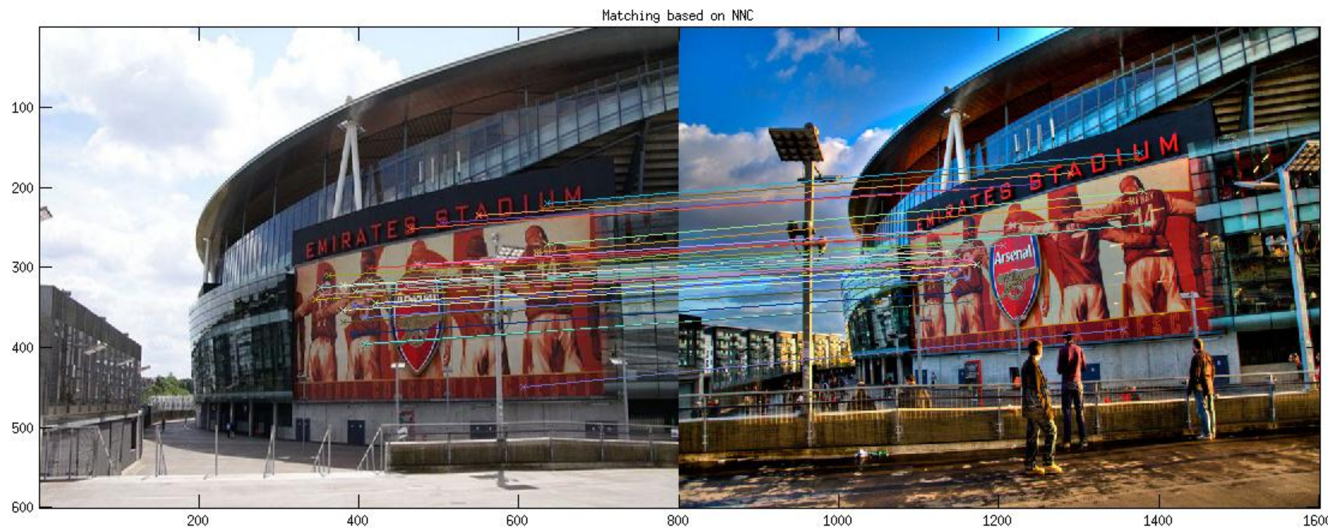

Figure 26. SIFT: Interest Points Matching Based on NCC

Conclusion for Set 2:

For this set of images as the view angle (**ESPECIALLY** the lighting conditions, color saturation, etc) changed **significantly**, Harris corner detector did not perform as well as in the previous set. For the correspondences established based on SSD and NCC, the matching rate decreased significantly.

Although larger the σ, less sensitive the Harris Corner detector is (less interest points detected is not necessarily bad). Those points detected with larger σ actually tend to be more accurately matched across the images.

As the Harris Corner detector yield bad results for this pair, SIFT operator actually works a lot better! The results based on Euclidean and SSD are great, but the result based on NCC is greater! With NCC, SIFT actually succeeded in matching the Arsenal Player Poster, while there are not so many significant corners in there (by human visual).

Based on the result from this part, we have concluded when the features are more robust, SIFT with NCC would improve our matching rate significantly.

7.3 My Own Set: Harris Operator/SIFT Comparison

Figure 27. My Set: my1.jpg

Figure 28. My Set: my2.jpg

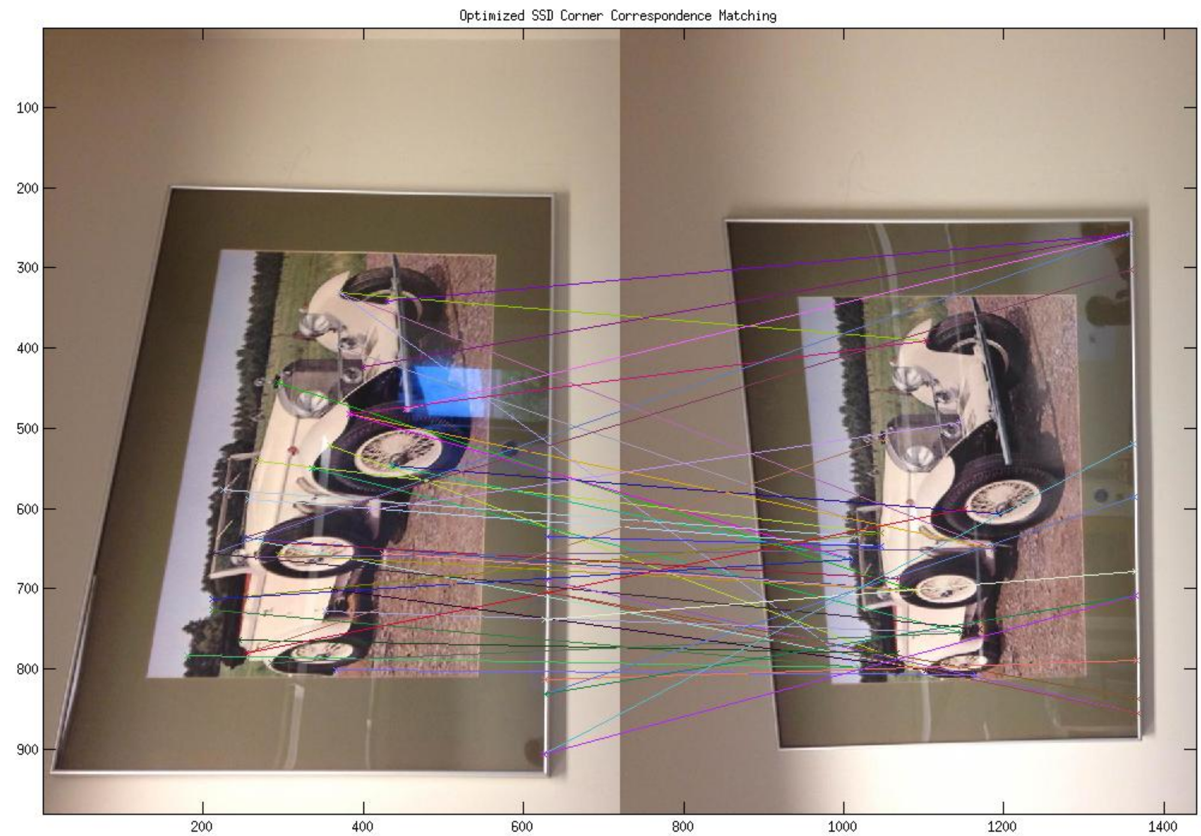

Figure 29. Harris: The SSD matching with $\sigma = 0.6$

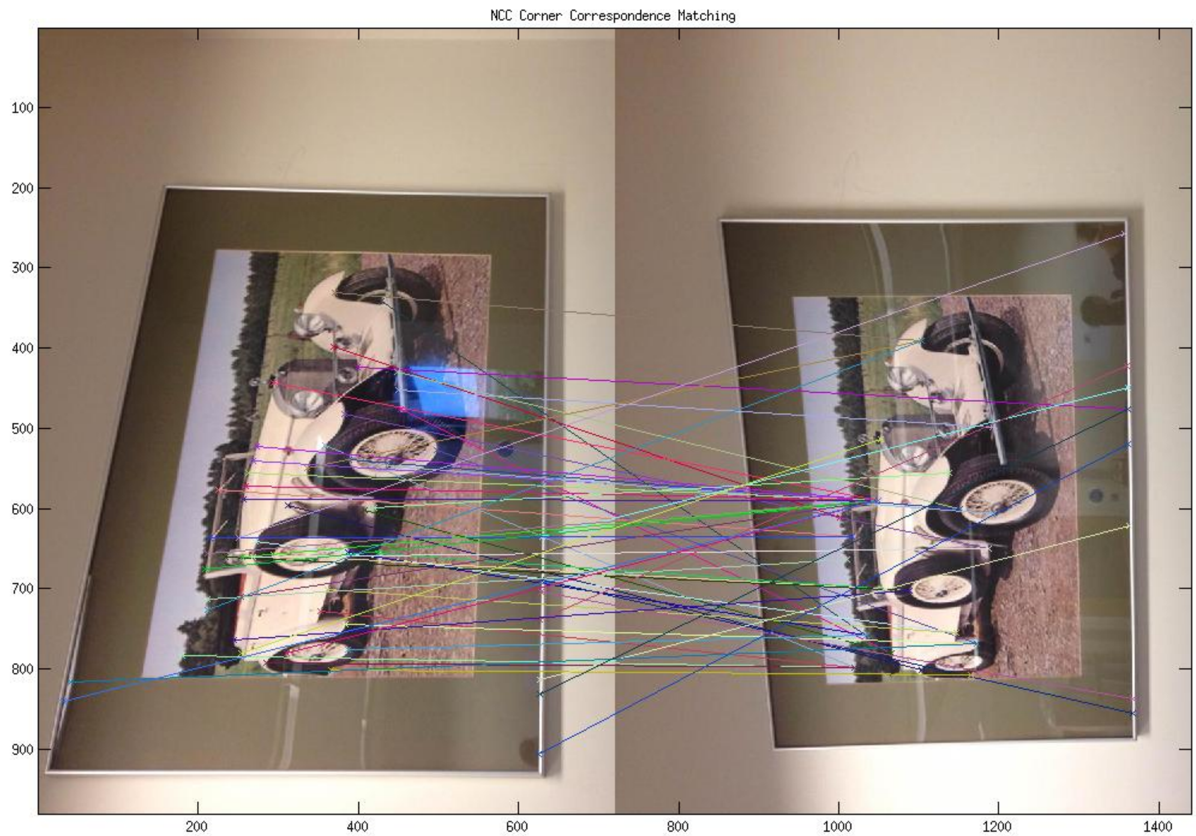

Figure 30. Harris: The NCC matching with $\sigma = 0.6$

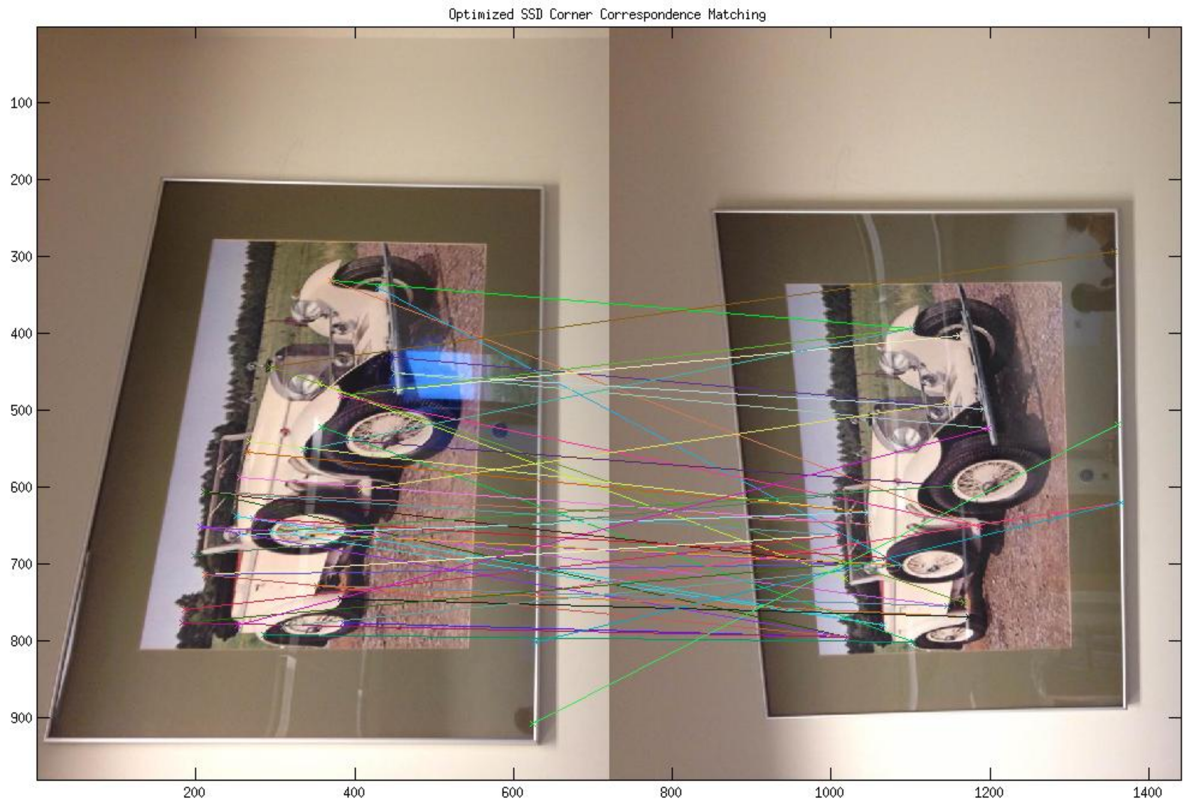

Figure 31. Harris: The SSD matching with $\sigma = 1$

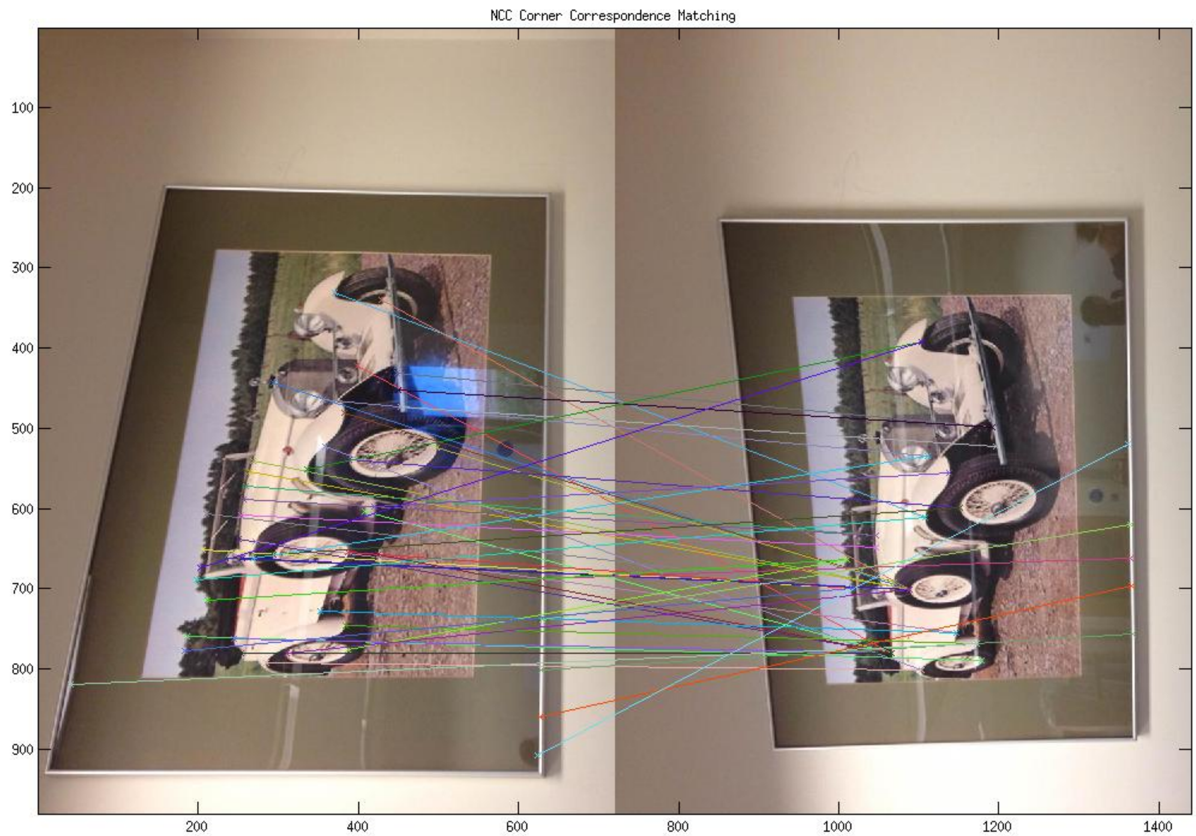

Figure 32. Harris: The NCC matching with $\sigma = 1$

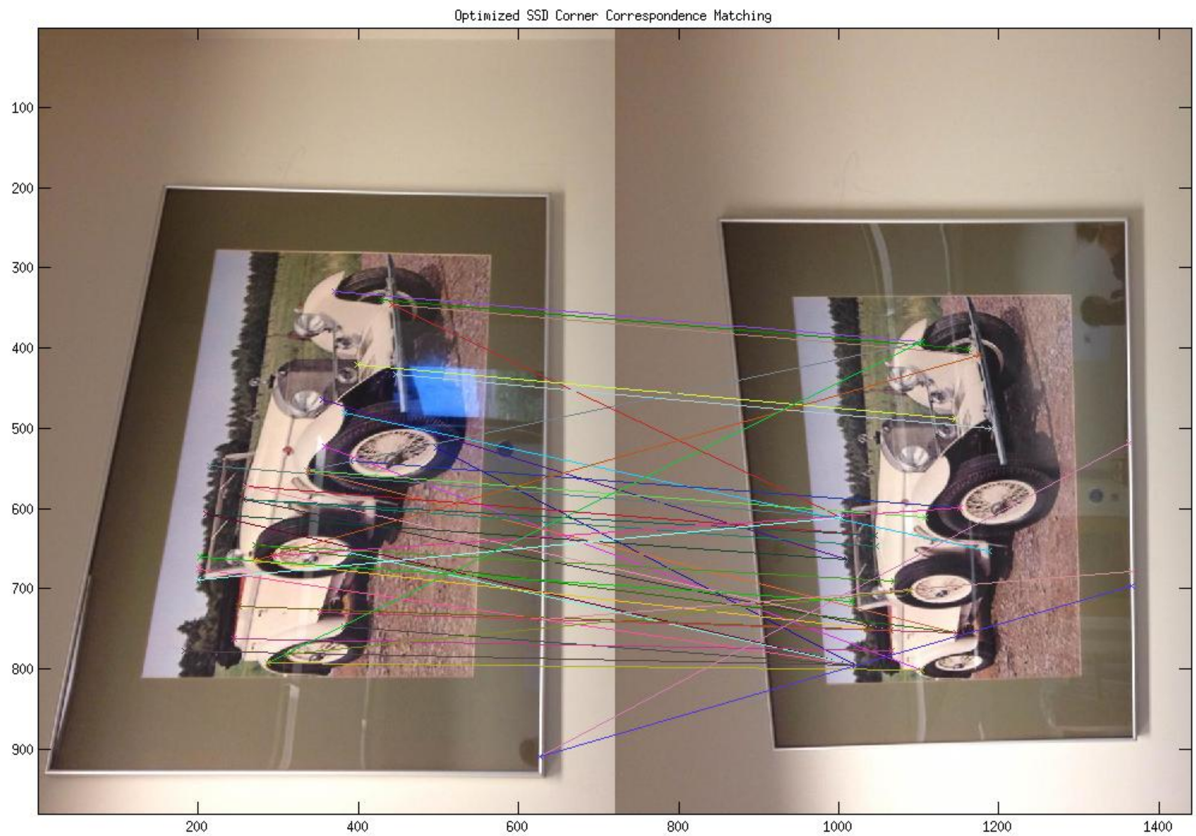

Figure 33. Harris: The SSD matching with $\sigma = 1.4$

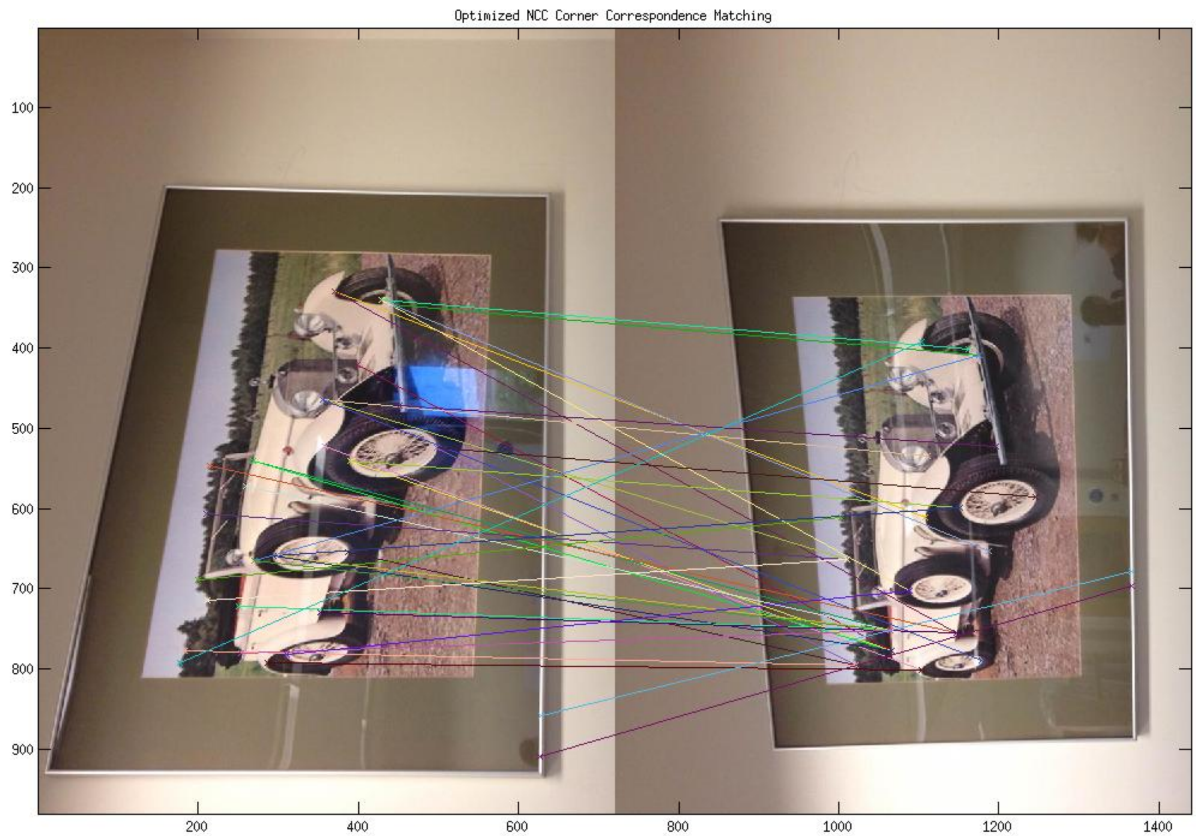

Figure 34. Harris: The NCC matching with $\sigma = 1.4$

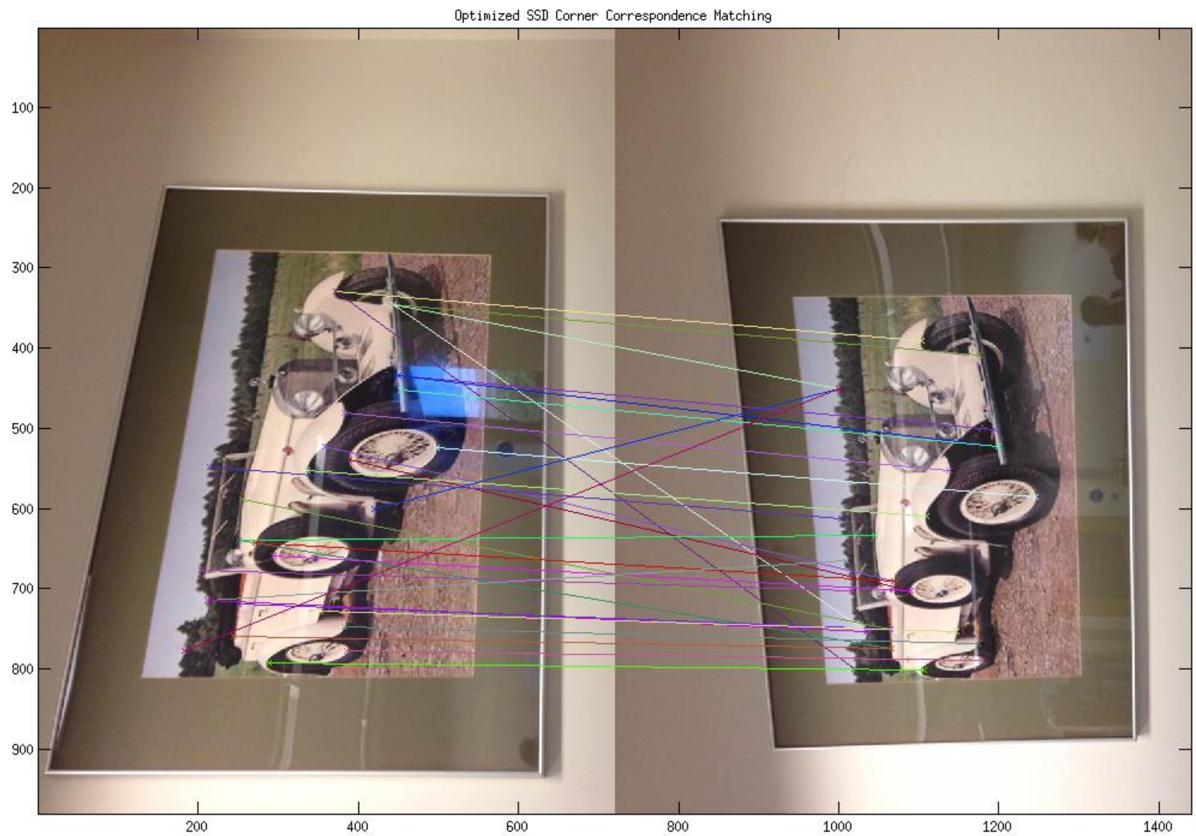

Figure 35. Harris: The SSD matching with $\sigma = 2.2$

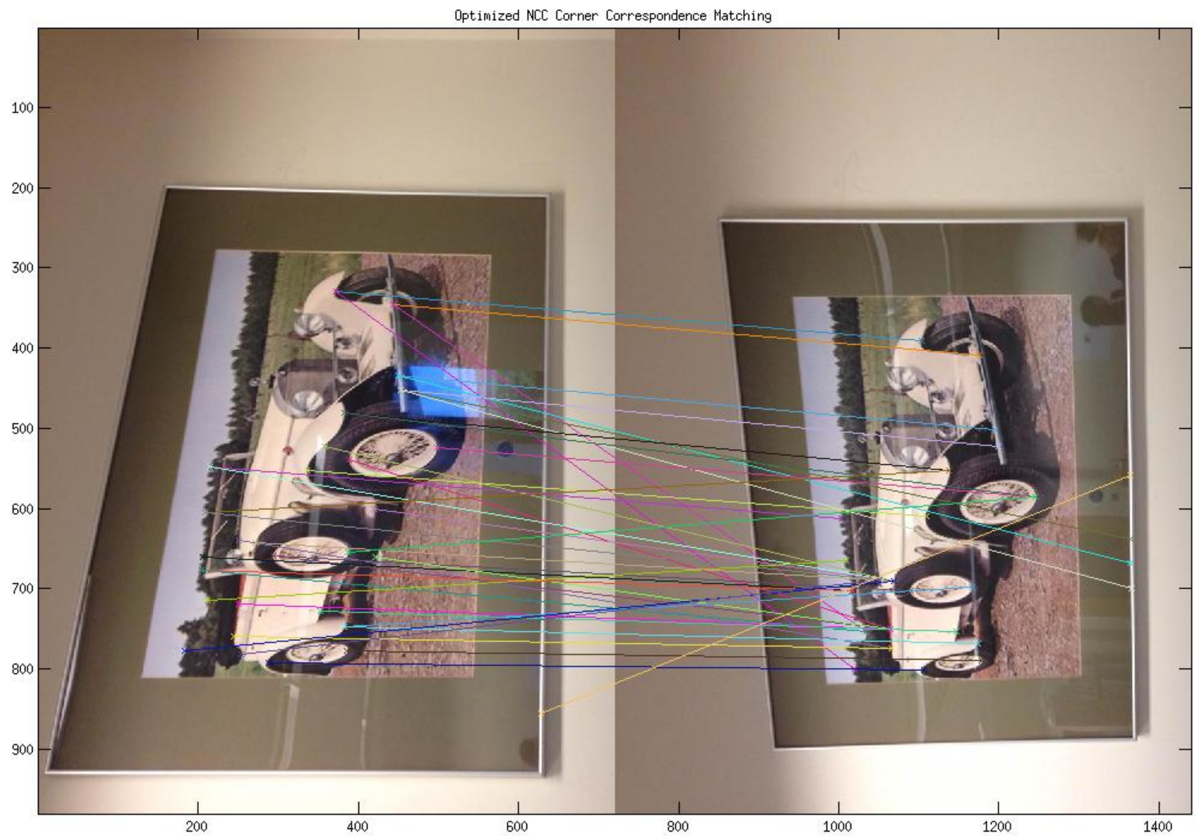

Figure 36. Harris: The NCC matching with $\sigma = 2.2$

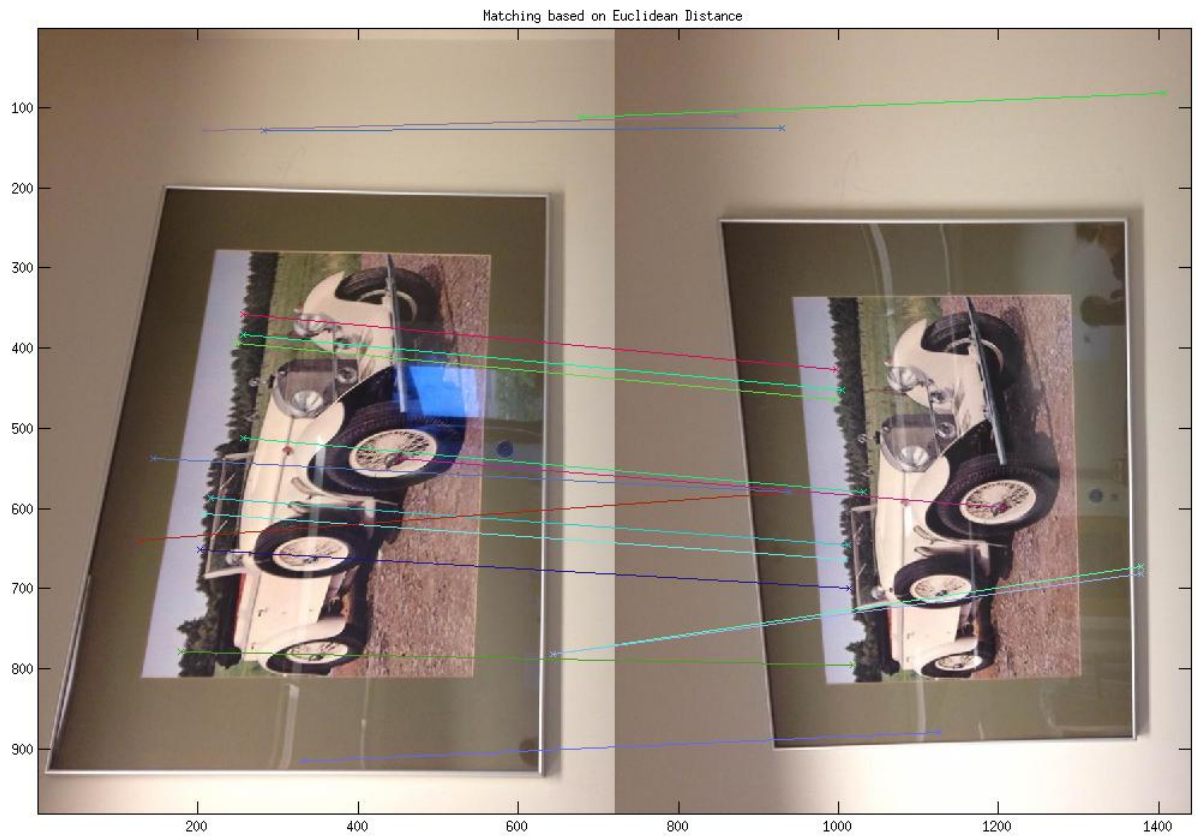

Figure 37. SIFT: Interest Points Matching Based on Euclidean Distance

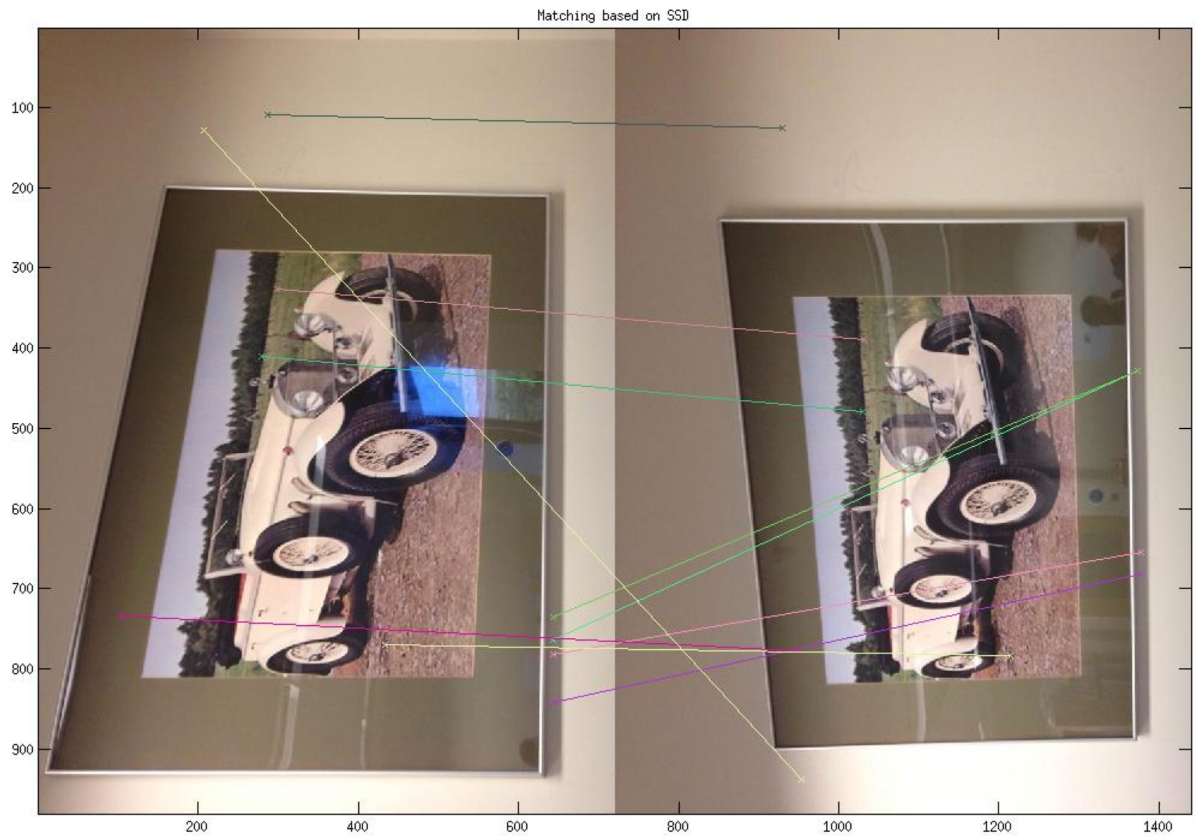

Figure 38. SIFT: Interest Points Matching Based on SSD

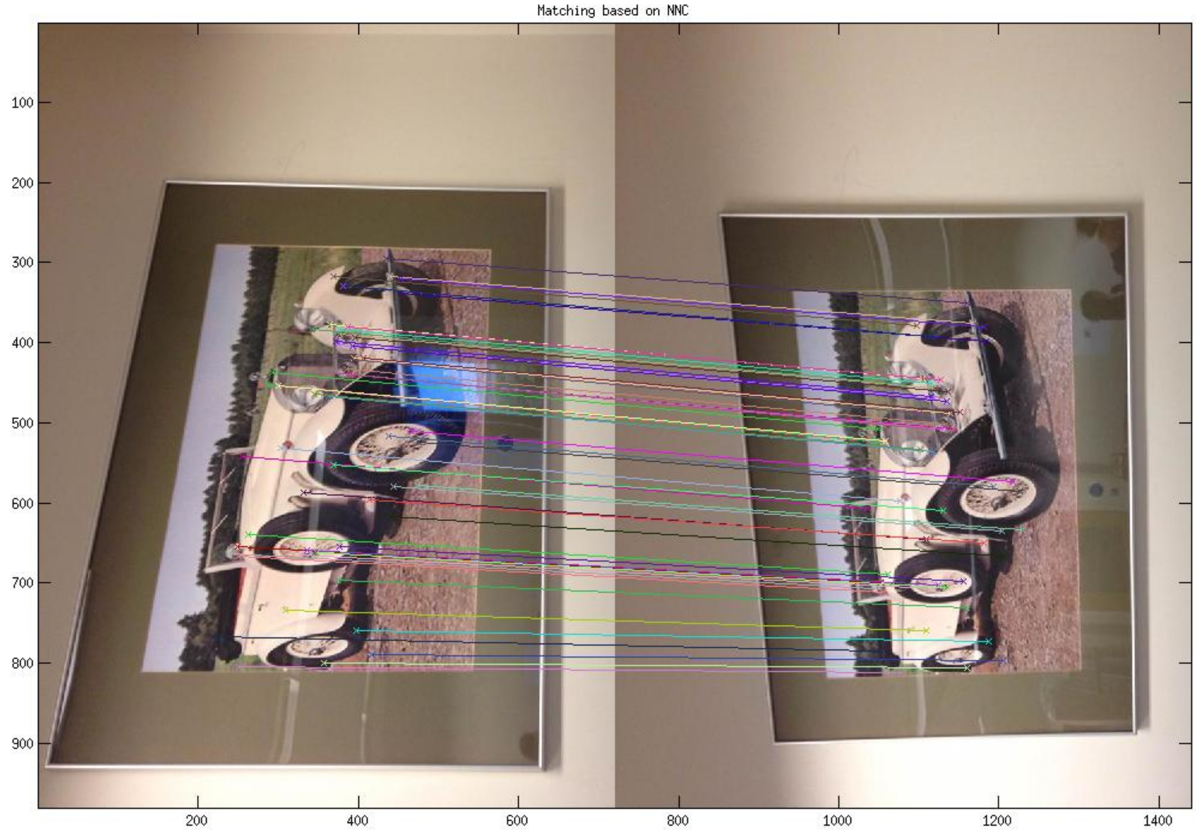

Figure 39. SIFT: Interest Points Matching Based on NCC

Conclusion for Set 1:

Although for this set of images the view angle changed slightly as in Set 1, Harris corner detector did not perform as well as in the previous set. For the correspondences established based on SSD and NCC, the matching rate decreased significantly compared to those in set 1.

Challenge: If we look closely into this pair, we will find there is a huge trouble. Unlike Set 1, a lot of features in the images are very similar. For example: The Frames, The Car White Paint, The Tyres, Those Trees. Even by human visual if we only look into the small details we can not distinguish one object from another. However, SIFT has proven to be more accurate/sensitive than human visual in this case.

Again, similar as to those in Set 2 larger the σ, less sensitive the Harris Corner detector is (less interest points detected is not necessarily bad). Those points detected with larger σ actually tend to be more accurately matched across the images.

As the Harris Corner detector did not yield ideal results for this pair, again, SIFT operator actually works better! The results based on Euclidean and SSD are great (there are only a few points because the threshold and other limiting conditions are strict in order to avoid mismatch as much as possible), but the result based on NCC is greater! With NCC, SIFT actually succeeded in matching the Old Car in the images, while there are not so many significant corners in there (by human visual).

Based on the result from this part, once more we have concluded when the features are more robust, SIFT with NCC would improve our matching rate significantly.

7.4 Intermediate Results: Gradient, Corners

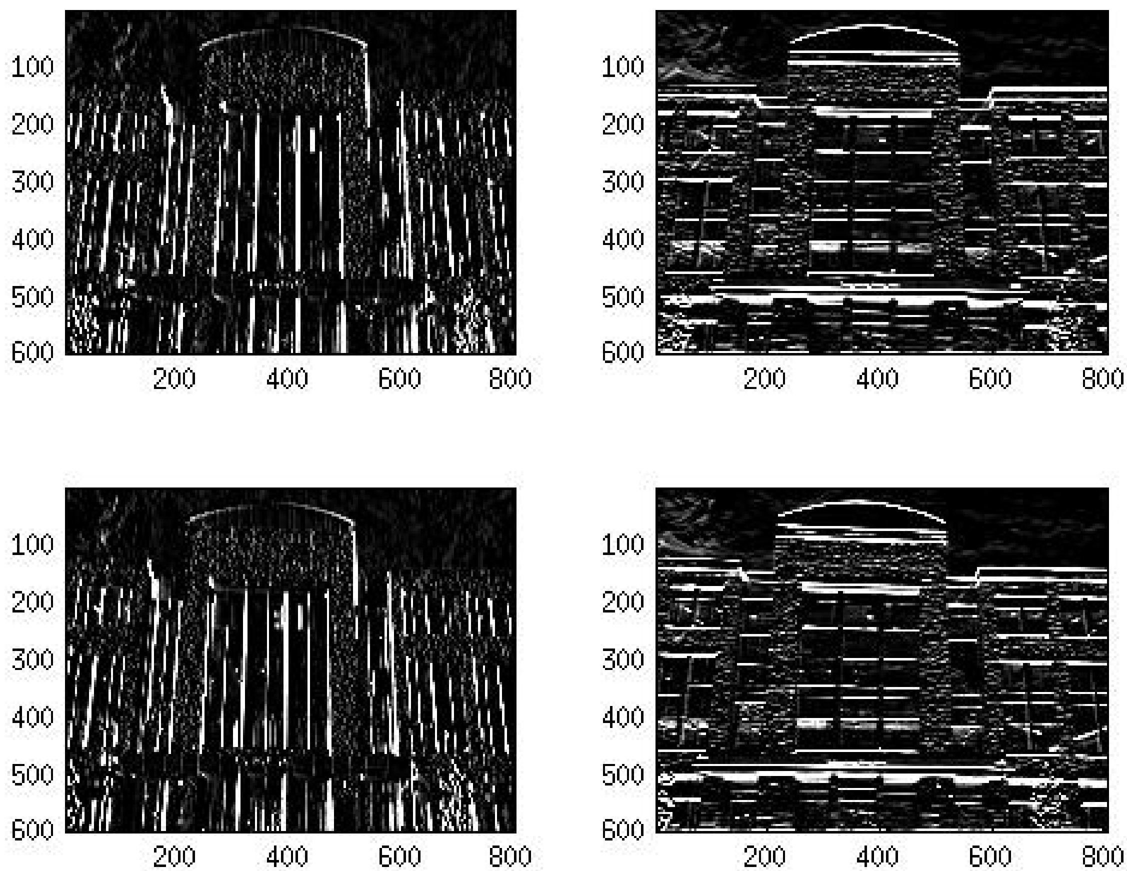

Fig 40. The x-gradient (left) and y-gradient (right) for pic1.jpg (upper) and pic2.jpg (lower) using $\sigma = 1$

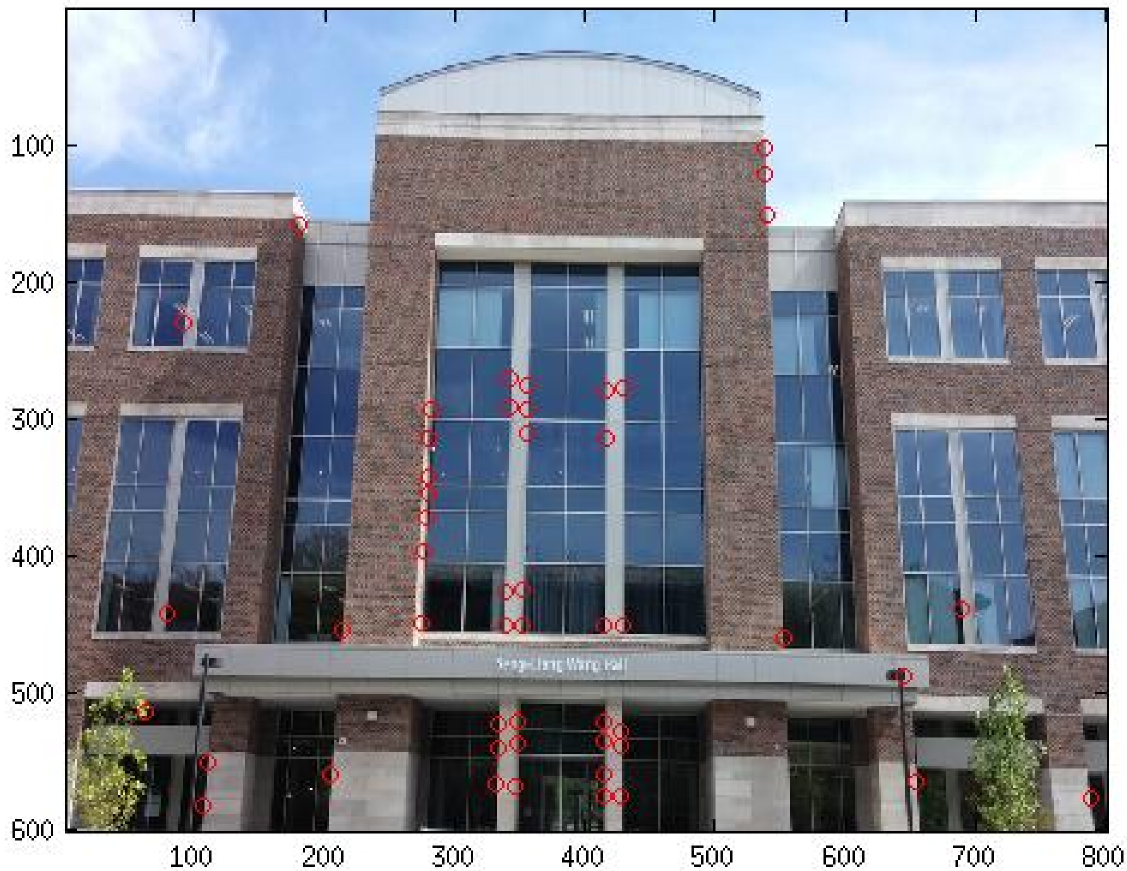

Fig 41. The corner points detected on pic1.jpg using $\sigma = 1$

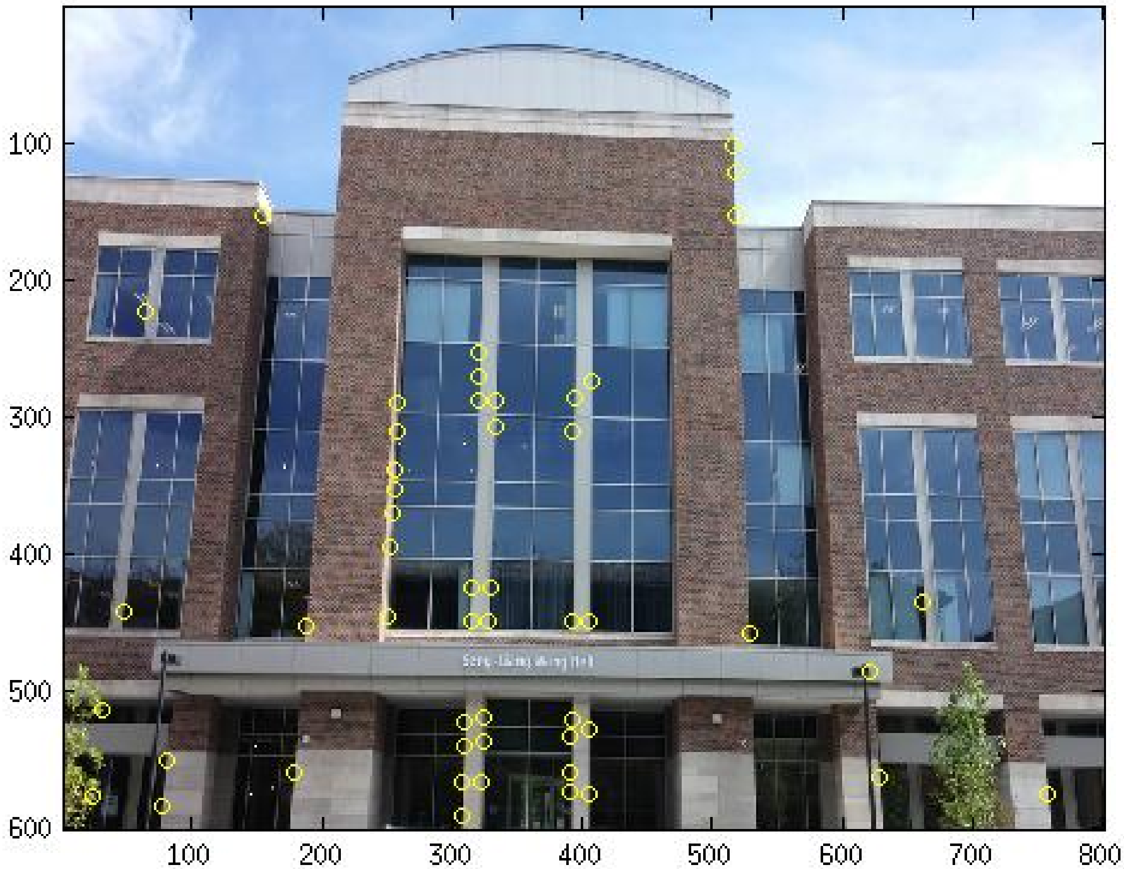

Fig 42. The corner points detected on pic2.jpg using $\sigma = 1$

7.5 Appendix A: Harris Corner Detection Matlab Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read the test images, and seek user input for a scale sigma
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc
pic1 = imread('your_image_name_1.jpg');  % your_image_name_1 = the image1 want to be processed
pic2 = imread('your_image_name_2.jpg');  % your_image_name_2 = the image2 want to be processed

scale = input('Enetr a scale:');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Change the RGB images into gray scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pic1_gray = rgb2gray(pic1);
pic2_gray = rgb2gray(pic2);
```

```
pic1_gray = double(pic1_gray);
pic2_gray = double(pic2_gray);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up some coefficient, Rssd = ratio for SSD, Rncc = ratio for NCC
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Rssd = 0.85
Rncc = 1.01
k = 0.04;


I1 = rgb2gray(pic1);
I2 = rgb2gray(pic2);

size_I1 = size(I1);
size_I2 = size(I2);
size_I1 = size(I1);
size_I2 = size(I2);

I1x = zeros(size_I1(1),size_I1(2));
I1y = zeros(size_I1(1),size_I1(2));
I2x = zeros(size_I2(1),size_I2(2));
I2y = zeros(size_I2(1),size_I2(2));
I1 = double(I1);
I2 = double(I2);

haar_size = round(round((4*scale+1))/2)*2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Smooth the image a bit before processing to make sure those noise would
% not be detected as corners (to improve computational efficiency)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
smooth_filter = fspecial('gaussian', 5*scale, scale);
I1 = imfilter(I1,smooth_filter);
I2 = imfilter(I2,smooth_filter);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Applying 'Haar' Filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Hx(1:haar_size,1:haar_size/2) = -1;
Hx(1:haar_size,haar_size/2+1:haar_size) = 1;
Hy(1:haar_size/2,1:haar_size) = 1;
Hy(haar_size/2+1:haar_size,1:haar_size) = -1;

I1x = imfilter(I1,Hx);
I1y = imfilter(I1,Hy);
I2x = imfilter(I2,Hx);
I2y = imfilter(I2,Hy);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This part no longer useful. The part is for the initial implementation of Sobel filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% This part is for sobel operator
%
%
% for i = 2:1:size_I1(1)-1;
%     for j = 2:1:size_I1(2)-1;
%         I1x(i,j) = I1(i-1,j+1) + 2*I1(i, j+1) + I1(i+1, j+1) -I1(i-1,j-1) ...
%             - 2*I1(i,j-1) - I1(i+1,j-1);
%         I1y(i,j) = I1(i-1,j-1) + 2*I1(i-1,j) + I1(i-1,j+1)  - I1(i+1,j-1) ...
%             -2*I1(i+1, j) - I1(i+1, j+1);
%     end
% end
%
% for i = 2:1:size_I2(1)-1;
%     for j = 2:1:size_I2(2)-1;
%         I2x(i,j) = I2(i-1,j+1) + 2*I2(i, j+1) + I2(i+1, j+1) -I2(i-1,j-1) ...
%             - 2*I2(i,j-1) - I2(i+1,j-1);
%         I2y(i,j) = I2(i-1,j-1) + 2*I2(i-1,j) + I2(i-1,j+1)  - I2(i+1,j-1) - ...
%             2*I2(i+1, j) - I2(i+1, j+1);
%     end
% end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot the gradient as intermediate result to make sure Haar filter was correctly implemented
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
subplot(2,2,1)
image(I1x)
colormap(gray(256))
subplot(2,2,2)
image(I1y)
colormap(gray(256))
subplot(2,2,3)
image(I2x)
colormap(gray(256))
subplot(2,2,4)
image(I2y)
colormap(gray(256))

tic
disp('Compute the C matrix for Image 1...')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the C matrix for image 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =1:1:size_I1(1)
    for j = 1:1:size_I1(2)
        C_Matrix_I1 = [0,0;0,0];
        for m = -(5*scale-1)/2:1:(5*scale-1)/2
```

```
            for n = -(5*scale-1)/2:1:(5*scale-1)/2
                if (i+m>0) && (i+m < size_I1(1)) && (j+n>0) && (j+n < size_I1(2))
                C_Matrix_I1(1,1) = C_Matrix_I1(1,1) + I1x(i+m,j+n)*I1x(i+m,j+n);
                C_Matrix_I1(2,2) = C_Matrix_I1(2,2) + I1x(i+m,j+n)*I1x(i+m,j+n);
                C_Matrix_I1(1,2) = C_Matrix_I1(1,2) + I1x(i+m,j+n)*I1y(i+m,j+n);
                C_Matrix_I1(2,1) = C_Matrix_I1(2,1) + I1x(i+m,j+n)*I1y(i+m,j+n);
                else
                end
                end
        end
        C_I1{i,j} = C_Matrix_I1;
    end
end
toc
disp('Check the rank of C matrix for Image 1... ')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check the rank of C matrix. If rank not euqal to 2 then dump the points
% If rank(C) = 2 then save the points for further processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =1:1:size_I1(1)
    for j = 1:1:size_I1(2)
        C_Matrix_I1 = C_I1{i,j};
        if (rank(C_Matrix_I1) == 2)
            I_corner_I1(i,j) = 1;
%             plot(j,i,'b*');
        else
            I_corner_I1(i,j) = 0;
        end
    end
end
toc
disp('Evaluating the corner strength for Image 2... ')

Corner_strength_I1 = zeros(size_I1(1),size_I1(2));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimate the corner strength at the remaining cadidate locations for image 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =1:1:size_I1(1)
    for j = 1:1:size_I1(2)
        if (I_corner_I1(i,j) == 1)
            %[U,S,V] = svd(C_I1{i,j});   %%%No need to use SVD
            %Corner_strength_I1(i,j) = S(1,1)*S(2,2) - k*(S(1,1)+S(2,2))^2;   %%%No need to use SVD
            Corner_strength_H_I1(i,j) = det(C_I1{i,j}) - k*(trace(C_I1{i,j}))^2; %%%This is better way to calculate corner strength
        end
    end
end

```

```
toc
disp('Compute the C matrix for Image 2...')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate the C matrix for image 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =1:1:size_I2(1)
    for j = 1:1:size_I2(2)
        C_Matrix_I2 = [0,0;0,0];
        for m = -(5*scale-1)/2:1:(5*scale-1)/2
            for n = -(5*scale-1)/2:1:(5*scale-1)/2
                if (i+m>0) && (i+m < size_I2(1)) && (j+n>0) && (j+n < size_I2(2))
                C_Matrix_I2(1,1) = C_Matrix_I2(1,1) + I2x(i+m,j+n)*I2x(i+m,j+n);
                C_Matrix_I2(2,2) = C_Matrix_I2(2,2) + I2x(i+m,j+n)*I2x(i+m,j+n);
                C_Matrix_I2(1,2) = C_Matrix_I2(1,2) + I2x(i+m,j+n)*I2y(i+m,j+n);
                C_Matrix_I2(2,1) = C_Matrix_I2(2,1) + I2x(i+m,j+n)*I2y(i+m,j+n);
                else
                end
                end
        end
        C_I2{i,j} = C_Matrix_I2;
    end
end

toc
disp('Check the rank of C matrix for Image 2... ')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check the rank of C matrix. If rank not euqal to 2 then dump the points
% If rank(C) = 2 then save the points for further processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i =1:1:size_I2(1)
    for j = 1:1:size_I2(2)
        C_Matrix_I2 = C_I2{i,j};
        if (rank(C_Matrix_I2) == 2)
            I_corner_I2(i,j) = 1;
        else
            I_corner_I2(i,j) = 0;
        end
    end
end

Corner_strength_I2 = zeros(size_I2(1),size_I2(2));


toc
disp('Evaluating the corner strength for Image 2... ')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimate the corner strength at the remaining cadidate locations for image 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
for i =1:1:size_I2(1)
    for j = 1:1:size_I2(2)
        if (I_corner_I2(i,j) == 1)
            Corner_strength_H_I2(i,j) = det(C_I2{i,j}) - k*(trace(C_I2{i,j}))^2;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up a dynamic threshold, thus if a candidate has a corner strength lower than
% the threshold, it would be filtered out (to improve computational efficiency)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

threshold = (max(max(Corner_strength_H_I1)) + max(max(Corner_strength_H_I2)))/20;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Counting the corners detected in both image and plot those corners
% This is intermediate results and will not appear on homework report
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure
image(I1)
colormap(gray(256))
hold on;
Actual_Corner_I1 = zeros(size_I1(1),size_I1(2));

toc
disp('Thresholding the corner candidates for Image 1... ')


cnt_cor1 = 0;
for i = 11:1:size_I1(1)-10
    for j = 11:1:size_I1(2)-10
        if (Corner_strength_H_I1(i,j) > threshold) && ...
        (Corner_strength_H_I1(i,j) == max(max(Corner_strength_H_I1(i-10:1:i+10,j-10:1:j+10))))
    Actual_Corner_I1(i,j) = 1;
    plot(j,i,'rx');
    cnt_cor1 = cnt_cor1 + 1;
    corner_loc1(cnt_cor1,1:2) = [i;j];
        else
        end
    end
end

figure
image(I2)
colormap(gray(256))
hold on;
Actual_Corner_I2 = zeros(size_I2(1),size_I2(2));

```

```
toc
disp('Thresholding the corner candidates for Image 2... ')
cnt_cor2 = 0;
for i = 11:1:size_I2(1)-10
    for j = 11:1:size_I2(2)-10
        if (Corner_strength_H_I2(i,j) > threshold) && ...
        (Corner_strength_H_I2(i,j) == max(max(Corner_strength_H_I2(i-10:1:i+10,j-10:1:j+10))))
    Actual_Corner_I2(i,j) = 1;
    plot(j,i,'bx');
    cnt_cor2 = cnt_cor2 + 1;
    corner_loc2(cnt_cor2,1:2) = [i;j];
        else
        end
    end
end

corner_count1 = sum(sum(Actual_Corner_I1))
corner_count2 = sum(sum(Actual_Corner_I2))

window_size = scale*20;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%              Optimized    SSD    %%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set a window so the the SSD of each candidate could be found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for m = 1:1:cnt_cor1
    i1 = corner_loc1(m,1);
    j1 = corner_loc1(m,2);
    for n = 1:1:cnt_cor2
        i2 = corner_loc2(n,1);
        j2 = corner_loc2(n,2);
        SSD_Win = pic1_gray(i1-window_size/2:1:i1+window_size/2,j1-window_size/2:1:j1+window_size/2) - ...
            pic2_gray(i2-window_size/2:1:i2+window_size/2,j2-window_size/2:1:j2+window_size/2);
        SSD(m,n) = sumsqr(SSD_Win);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set a new image prepared for displaying the matched interest points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
new_image(1:(max(size_I1(1),size_I2(1))),1:size_I1(2)+size_I2(2),1:3) = ...
    zeros(max(size_I1(1),size_I2(1)), size_I1(2)+size_I2(2),3);
new_image(1:size_I1(1),1:size_I1(2),:) = pic1;
new_image(1:size_I2(1),1+size_I1(2):size_I2(2)+size_I1(2),:) = pic2;
new_image = uint8(new_image);
```

```
figure
image(new_image)
truesize
hold on;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If an interest point with SSD:
% 1) Smaller than a threshold
% 2) Minima
% 3) Minima/Second Minama < Rssd (ratio)
% Correspondence Established
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for m = 1:1:cnt_cor1
    i1 = corner_loc1(m,1);
    j1 = corner_loc1(m,2);
    for n = 1:1:cnt_cor2
         if (SSD(m,n) == min(SSD(m,:))) && (SSD(m,n) < 40* min(min(SSD(:,:))))
             local_minima = SSD(m,n);
             SSD(m,n) = max(SSD(m,:));
             if (local_minima/min(SSD(m,:)) < Rssd)
             i2 = corner_loc2(n,1);
             j2 = corner_loc2(n,2);
             rand_color = rand(1, 3);
             plot([j1;size_I1(2)+j2],[i1;i2],'-x','Color',rand_color(1,:));
             n = cnt_cor2;
             else
             end
        else
        end

    end
end
title('Optimized SSD Corner Correspondence Matching')



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Optimized NCC %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set a window so the the NCC of each candidate could be found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:cnt_cor1
    i1 = corner_loc1(m,1);
    j1 = corner_loc1(m,2);
    for n = 1:1:cnt_cor2
        i2 = corner_loc2(n,1);
        j2 = corner_loc2(n,2);

        f1_m1 = pic1_gray(i1-window_size/2:1:i1+window_size/2,j1-window_size/2:1:j1+window_size/2) - ...
            mean(mean(pic1_gray(i1-window_size/2:1:i1+window_size/2,j1-window_size/2:1:j1+window_size/2)));
```

```
        f2_m2 = pic2_gray(i2-window_size/2:1:i2+window_size/2,j2-window_size
            /2:1:j2+window_size/2) - ...
            mean(mean(pic2_gray(i2-window_size/2:1:i2+window_size/2,j2-
                window_size/2:1:j2+window_size/2)));

        NNC(m,n) = sum(sum(f1_m1.*f2_m2))/((sumsqr(f1_m1)*sumsqr(f2_m2))^(1/2)
            );

    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set a new image prepared for displaying the matched interest points
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
new_image(1:(max(size_I1(1),size_I2(1))),1:size_I1(2)+size_I2(2),1:3) = ...
    zeros(max(size_I1(1),size_I2(1)), size_I1(2)+size_I2(2),3);
new_image(1:size_I1(1),1:size_I1(2),:) = pic1;
new_image(1:size_I2(1),1+size_I1(2):size_I2(2)+size_I1(2),:) = pic2;
new_image = uint8(new_image);
figure
image(new_image)
truesize
hold on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If an interest point with SSD:
% 1) Larger than a threshold
% 2) Maxima
% 3) Maxima/Second Maxima > Rncc (ratio)
% Correspondence Established
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for m = 1:1:cnt_cor1
    i1 = corner_loc1(m,1);
    j1 = corner_loc1(m,2);
    for n = 1:1:cnt_cor2
        if (NNC(m,n) == max(NNC(m,:))) && (NNC(m,n) > 0.3*max(max(NNC(:,:))))
            local_maxima = NNC(m,n);
            NNC(m,n) = min(NNC(m,:));
            if (local_maxima/max(NNC(m,:)) > Rncc)
            i2 = corner_loc2(n,1);
            j2 = corner_loc2(n,2);
            rand_color = rand(1, 3);
            plot([j1;size_I1(2)+j2],[i1;i2],'-x','Color',rand_color(1,:));
            n = cnt_cor2;
            else
            end
        else
        end

    end
end
title('Optimized NCC Corner Correspondence Matching')
```

7.6 Appendix B: SIFT Matlab Script

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read the test images, and seek user input for a scale sigma
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc
pic1 = imread('your_image_name_1.jpg');  % your_image_name_1 = the image1 want to be processed
pic2 = imread('your_image_name_2.jpg');  % your_image_name_2 = the image2 want to be processed
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set up some coefficient, Rssd = ratio for SSD, Rncc = ratio for NCC
% Reuc = ratio for Euclidean Distance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Reuc = 0.7;
Rssd = 0.7;
Rncc = 1.4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Change the RGB images into gray scale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pic1_gray = rgb2gray(pic1);
pic2_gray = rgb2gray(pic2);
pic1_gray = double(pic1_gray);
pic2_gray = double(pic2_gray);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform SIFT feature extration and extract both locations and descriptors
% for each candidates interest point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[I1,sift_vec_I1] = vl_sift(im2single(rgb2gray(pic1)));
[I2,sift_vec_I2] = vl_sift(im2single(rgb2gray(pic2)));
size_image_I1 = size(pic1);
size_I1 = size(I1);
size_I2 = size(I2);
points_size_I1 = size_I1(2);
points_size_I2 = size_I2(2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Round up those sub-pixel returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:1:points_size_I1
    corner_loc_I1(1,i) = round(I1(2,i));
    corner_loc_I1(2,i) = round(I1(1,i));
end

for i = 1:1:points_size_I2
    corner_loc_I2(1,i) = round(I2(2,i));
    corner_loc_I2(2,i) = round(I2(1,i));
end


disp('Calculating Euclidean Distance Matrix')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Calculating the Euclidean Distance of vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:points_size_I1
    i1 = corner_loc_I1(1,m);
    j1 = corner_loc_I1(2,m);
    for n = 1:1:points_size_I2
        i2 = corner_loc_I2(1,n);
        j2 = corner_loc_I2(2,n);
        sift_diff = double(sift_vec_I1(:,m)) - double(sift_vec_I2(:,n));
        sift_mat(m,n) = sumsqr(sift_diff);
    end
end
disp('Matching Interest Points Based on Euclidean Distance Matrix')
figure
image(cat(2, pic1, pic2));
truesize
hold on;
title('Matching based on Euclidean Distance')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Macthing images the Euclidean Distance of vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:points_size_I1
    i1 = corner_loc_I1(1,m);
    j1 = corner_loc_I1(2,m);
    for n = 1:1:points_size_I2
        if (sift_mat(m,n) == min(sift_mat(m,:)) && (sift_mat(m,n)<5*min(min(sift_mat))))
            loc_minimum = sift_mat(m,n);
            sift_mat(m,n) = max(sift_mat(m,:));
            if (loc_minimum/min(sift_mat(m,:)) < Reuc)
            i2 = corner_loc_I2(1,n);
            j2 = corner_loc_I2(2,n);
            rand_color = rand(1, 3);
            plot([j1;size_image_I1(2)+j2],[i1;i2],'-x','Color',rand_color(1,:));
            n = points_size_I2;
            else
            end
        else
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the SSD from vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Calculating SSD Matrix')
SSD = sift_mat.^2;
disp('Matching Interest Points Based on SSD Matrix')
figure
image(cat(2, pic1, pic2));
truesize
hold on;
title('Matching based on SSD')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matching images based on the SSDf vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:points_size_I1
    i1 = corner_loc_I1(1,m);
    j1 = corner_loc_I1(2,m);
    for n = 1:1:points_size_I2
        if (SSD(m,n) == min(SSD(m,:)) && (SSD(m,n)<5*min(min(SSD))))
            loc_minimum = SSD(m,n);
            SSD(m,n) = max(SSD(m,:));
            if (loc_minimum/min(SSD(m,:)) < Rssd)
            i2 = corner_loc_I2(1,n);
            j2 = corner_loc_I2(2,n);
            rand_color = rand(1, 3);
            plot([j1;size_image_I1(2)+j2],[i1;i2],'-x','Color',rand_color(1,:)
                );
            n = points_size_I2;
            else
            end
        else
        end
    end
end

disp('Calculating the NCC matrix')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculating the NCC from vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:points_size_I1
    i1 = corner_loc_I1(1,m);
    j1 = corner_loc_I1(2,m);
    for n = 1:1:points_size_I2
        i2 = corner_loc_I2(1,n);
        j2 = corner_loc_I2(2,n);
        f1m1 = double(sift_vec_I1(:,m)) - double(mean(sift_vec_I1(:,m)));
        f2m2 = double(sift_vec_I2(:,n)) - double(mean(sift_vec_I2(:,n)));
        NCC(m,n) = sum(f1m1.*f2m2)/((sumsqr(f1m1)*(sumsqr(f2m2)))^(1/2));

    end
end

figure
image(cat(2, pic1, pic2));
truesize
hold on;
title('Matching based on NNC')
disp('Matching Interest Points Based on NCC Matrix')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Matching images based on the NCC from vectors returned from SIFt operator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m = 1:1:points_size_I1
    i1 = corner_loc_I1(1,m);
    j1 = corner_loc_I1(2,m);
    m
```

```
    for n = 1:1:points_size_I2
        if (NCC(m,n) == max(NCC(m,:))) && (NCC(m,n) > max(max(NCC))*(0.9))
            local_maxima = max(NCC(m,n));
            NCC(m,n) = min(NCC(m,:));
            if (local_maxima/max(NCC(m,:)) > Rncc)
            i2 = corner_loc_I2(1,n);
            j2 = corner_loc_I2(2,n);
            rand_color = rand(1, 3);
            plot([j1;size_image_I1(2)+j2],[i1;i2],'-x','Color',rand_color(1,:)
                );
            n = points_size_I2;
            else
            end
        else
        end
    end
end
```