ECE 661 Homework #4

Soonam Lee

October 2, 2014

1 Introduction

The purpose of this project is detecting interest points from images and using simple comparisons to establish correspondences between the interest points extracted from two different images of the same scene. To achieve this, we develop our own version of the Harris corner detector. Using 4 different scales of the Harris corner detector, we extract interest points from two different viewpoint images. In addition, we use SIFT feature extractor from $VLFeat$ library to compare the performance with the Harris corner detector. After extracting the interest points, we should compute correspondence of each interest points and bind them as pairs. Using two metrics such that the NCC (Normalized Cross Correlation) and the SSD (Sum of Squared Differences) metrics, we can establish correspondences between the interest points in the two images of the same scene.

This document is structured as follows. The theoretical background of Harris corner detector and SIFT feature descriptors are introduced in Section 2. Then, brief explanation of feature matching methods are followed in Section 3. The experimental results with given pairs of image sets are shown in detail in Section 4. Lastly, the MATLAB codes that use for these experiments are attached in the end.

2 Theoretical Background

In this section, we discuss two different feature extraction methods, Harris corner detector and SIFT feature descriptor. Each method will be described as step by step with relevant equations.

2.1 Harris Corner Detector

The goal of the Harris corner detector is finding the points that can be defined as corners. A corner can be defined such that the pixel in the vicinity of which the gray levels show significant variation in at least two different directions. In order to perform the Harris corner detectors, we have following steps:

1. Take input as a simple 2D image I and calculate intensity variations at different scales along the x and y direction. If scale is not important, we can use the Sobel operator to calculate these variations. In this project, however, different from previous year projects, we need to consider different scales. In order to compute different scales, we apply different size of Haar filters and denote $I_x^{(i)}$ and $I_y^{(i)}$ where x- and y-oriented filters at new sigma σi.

   The basic form of the Haar wavelets are $(-1, 1)$ along x and $(1, -1)^T$ along y. These forms are scaled up to an $S \times S$ such that S is the smallest even integer greater than 4σ. For example, when $\sigma = 1.4$, two Haar wavelets should be expressed as

$$\begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix},$$

because greater but smallest even number of 4σ is 6. Different scale gives us different size of these Haar wavelet filter matrix. After that, convolve these filters with given images, we can get $I_x^{(i)}$ and $I_y^{(i)}$ respectively.

2. Construct matrix $M^{(i)}$ such that

$$M^{(i)} = \begin{bmatrix} \sum (I_x^{(i)})^2 & \sum I_x^{(i)} I_y^{(i)} \\ \sum I_x^{(i)} I_y^{(i)} & \sum (I_y^{(i)})^2 \end{bmatrix} = U^{-1} \begin{bmatrix} \lambda_1^{(i)} & 0 \\ 0 & \lambda_2^{(i)} \end{bmatrix} U,$$

in a $5\sigma \times 5\sigma$ neighborhood window of the pixel where we would like to check the presence or absence of a corner. We prefer the size of this $5\sigma \times 5\sigma$ neighborhood window as odd number to make sure the center point is a pixel. This 2×2 matrix has full rank at a genuine corner. However, if the pixel on which the $5\sigma \times 5\sigma$ window is on a straight edge, its rank reduces to 1. From this characteristics, we need to compute eigenvalues $(\lambda_1^{(i)}, \lambda_2^{(i)})$ of $M^{(i)}$. As observed from Figure 1, there are four different cases depending on these eigenvalues.

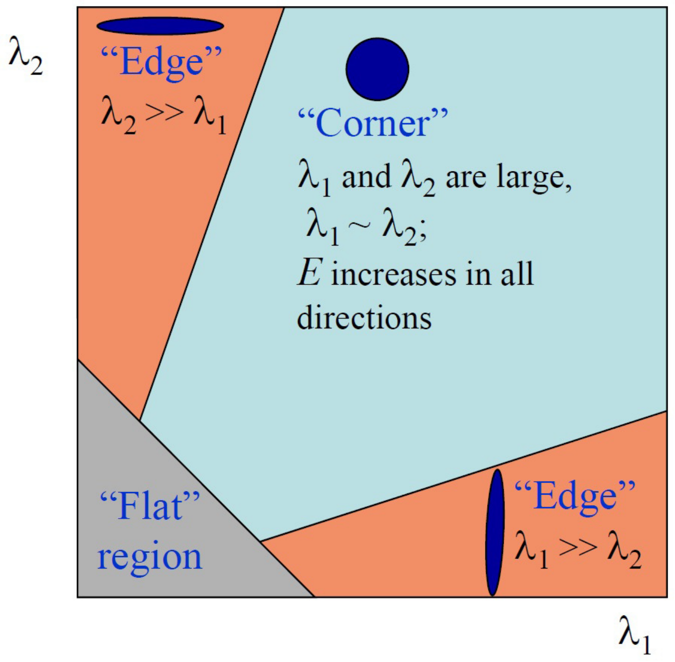

Figure 1: Relationship between eigenvalues and features

3. Calculate the determinant and trace of matrix $M^{(i)}$ and get corner response $R^{(i)}$. Note that determinant and trace are expressed as

$$det(M^{(i)}) = \lambda_1^{(i)} \lambda_2^{(i)}, \quad tr(M^{(i)}) = \lambda_1^{(i)} + \lambda_2^{(i)}.$$

The corner response $R^{(i)}$ can be computed as

$$R^{(i)} = det(M^{(i)}) - k(tr(M^{(i)}))^2 = \lambda_1^{(i)}\lambda_2^{(i)} - k(\lambda_1^{(i)} + \lambda_2^{(i)})^2,$$

where k should be determined empirically. In this project, this k is set to 0.04.

4. Using non-maximum suppression as we perform for the Canny edges, get rid of the pixels with corner response near pixels using certain size of window.

5. Find the points with large corner response with respect to threshold T_{Harris}. This means if $R^{(i)} > T_{Harris}$, we consider the pixel location as a corner. Otherwise, although it was classified with corner, we do not consider it as a corner.

2.2 SIFT Feature Descriptor

We extract SIFT feature points using $VLFeat$ embedded function. The details of theory of SIFT is already done in class, so we briefly introduce some key steps as follow:

1. **Scale-space extreme detection**
Using DoG (Difference-of-Gaussian) pyramid, find all the local extrema. The DoG can be computed as

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma) * I(x, y)) \\ &= ff(x, y, k\sigma) - ff(x, y, \sigma) \end{aligned}$$

where $ff(x, y, \sigma)$ is LoG (Laplacian-of-Gaussian) of image I(x, y). In addition, $G(x, y, \sigma)$ is a variable-scale Gaussian expressed as

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

This means we find points that are locally maximum or locally minimum in the three dimensional space (x, y, σ). From one point, we need to consider 26 neighbors in a 3D neighborhoods around $3 \times 3 \times 3$.

2. **Keypoint localization**
Since we use different scales (σ) across DoG pyramid, we should match the top extreme points to bottom image. In order to do this, using the second-order derivatives at the sampling points in the DoG pyramid, you can localize the extremum with sub-pixel accuracy in the vicinity of where the extremum was found in Step 1. We compute Taylor expansion such that

$$D(\mathbf{x}) = D(x_0) + J^T(x_0)\mathbf{x} + \frac{1}{2}\mathbf{x}^T H(x_0)\mathbf{x},$$

where $\mathbf{x}$ is the incremental deviation from x_0. The gradient vector J and the Hessian H at x is defined as

$$\begin{aligned} J(x_0) &= \left(\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial z}\right)^T_{x_0} \\ H(x_0) &= \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial \sigma} \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial \sigma} \\ \frac{\partial^2 D}{\partial \sigma \partial x} & \frac{\partial^2 D}{\partial \sigma \partial y} & \frac{\partial^2 D}{\partial \sigma^2} \end{bmatrix}. \end{aligned}$$

Using this Taylor expansion, SIFT feature can be interpolated near local extremum and find accurate position of original image.

3. **Discarding low-contrast keypoints**
To discard the keypoints with low contrast, thresholding $|D(\mathbf{x})|$ at the locations $(x, y, \sigma)^T = \mathbf{x}$ of the extremums. Typically, an extremum is rejected if $|D(\mathbf{x})| < 0.03$. We also reject those exrema in the scale space that draw their support from an edge in the image. The way that we eliminate edge is the same as we do in the Harris corner detector using trace and determinant of eigenvalues.

4. **Orientation assignment**
Now, we want to find the dominant local orientation. First, we calculate the gradient vector of the Gaussian-smoothed image $ff(x, y, \sigma)$ at the scale σ of the extremum. At each point in a $S \times S$ neighborhood around the extremum, we compute the gradient magnitude $(m(x, y))$ and the gradient orientation $(\theta(x, y))$ such that

$$\begin{aligned} m(x,y) &= \sqrt{|ff(x+1,y,\sigma) - ff(x,y,\sigma)|^2 + |ff(x,y+1,\sigma) - ff(x,y,\sigma)|^2} \\ \theta(x,y) &= \arctan\left(\frac{ff(x+1,y,\sigma) - ff(x,y,\sigma)}{ff(x,y+1,\sigma) - ff(x,y,\sigma)}\right). \end{aligned}$$

After weighting $\theta(x, y)$ with $m(x, y)$, we construct a histogram of $\theta(x, y)$ values using 36 bins spanning the full $360°$ range.

5. **Keypoint descriptor**
At the scale of the extremum, the extremum point is surrounded by a 16×16 neighborhood of points that is divided into 4×4 cells, each cell consisting of a 4×4 block of points. For each of the 16 cells, an 8-bin orientation histogram is calculated from the gradient-magnitude-weighted values of $\theta(x, y)$ at the 16 pixels in the cell. Finally, stringing together the 16 8-bin histograms yields a 128-element descriptor at each retained extremum in the DoG pyramid. This 128-element descriptor as a vector in a 128-dimensional space, its length is normalized to unity to make it invariant to change in illumination.

3 Feature Matching Methods

Once we got the low-level features such as corners from two different images, we can establish correspondences by directly comparing the gray levels. We use two metrics so called SSD (Sum of Squared Differences) and NCC (Normalized Cross-Correlation). However, if we got the features from scale-space operator such as SIFT, we can simply measuring the Euclidean distance between their descriptor vector.

3.1 SSD (Sum of Squared Differences)

After finding the corners using the Harris corner detector, one possible way of establishing correspondences between image pairs is to use the sum of squared differences. That is, for each pair of interest points in both input images I_1 and I_2 we compare their gray levels around an $(S+1) \times (S+1)$ window such that

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

Using this SSD, we can compute the differences between each pairs and take minimal one as corresponding features.

3.2 NCC (Normalized Cross-Correlation)

Another technique to establish correspondences between image pairs is the normalized cross-correlation. In this case, the gray levels of each pair of interest points in both input images I_1 and I_2 are compared around an $(S+1) \times (S+1)$ window in the following manner:

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

where m_i is the mean value along the $(S+1) \times (S+1)$ window around the interest point i. In this case, the NCC value will be between -1 and 1, being 1 a perfect match between two interest points around the comparison window. Therefore, we can set a threshold to filter the pairs of interest points that are more likely to match.

3.3 Euclidean Distance

When using the SIFT feature descriptors, the correspondences between image pairs can be done by simply compare the Euclidean distance between each pair of possible interest points in the two input images I_1 and I_2. Since each interest point is represented by a 128-element descriptor vector, we can compare two of them as follows:

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

where a and b are the two descriptors to be compared. Therefore, we can establish a threshold to filter the best matches. Note that this part is included in $VLFeat$ embedded SIFT function.

4 Comparison of Harris Corner Detector and SIFT Descriptors

1. As we learned from class, Harris corner detector is useful when there is not any scale space movement between pairs of image. Therefore, Harris corner detector can find many matched points in set 1 as SIFT did. However, except from set 1, SIFT feature extraction methods are always outperformed. Meanwhile, SIFT feature already considered scale parameters, it is much robust to extract corresponding feature points under scale space movement.

2. Harris corner detector is too sensitive to noise. In order to reduce this sensitive, we perform non-maxima suppression to extract just one corners inside certain window.

3. SSD doesn't need many computations, but the results from SSD matching is very dependent on thresholding. The NCC is more robust when there are affine distortion between a pair of images.

5 Experimental Results

In this section, we show feature extraction results using various methods (Harris corner detector, SIFT) and matching correspondences between these feature pairs with three different methods that we introduced from previous section. Note that we use two methods such as SSD, NCC for matching Harris corner detectors and use Euclidean distance for SIFT descriptors. Since we need to apply four different scales for the Harris corner detector, each image pairs produces nine different image matching results. We apply these methods for two given image pairs and two image pairs taken on my own.

Before moving on to the results, we introduce several parameters that we use for feature extraction methods and feature matching methods. The detail parameters are explained in Table 1. We generally use same values for different image pairs except from scale value (i) and thresholds of feature matching methods. More specifically, we use four different scale values from 1 to 4. Since this scale values make σ values changes, window size of Haar filter depends on this scale value. Moreover, in order to decide true corners, we should have threshold for Harris corner detector, $T_{Harris}^{(i)}$. Since the size of neighborhood window $M^{(i)}$ is changeable, fixed $T_{Harris}^{(i)}$ did not work. Therefore, we set $T_{Harris}^{(i)}$ as average of corresponding corner response $R^{(i)}$. Note that if certain pixel location has very large corner response, the corner might detect across all different scales.

In addition, we have p which control threshold values of SSD, NCC and Euclidean distance metrics. In particular, we compute the average of each metric value between two correspondences, and multiplying p to the average of metric values to control thresholds.

Harris parameter	**Description**	**Value**
k	sensitivity parameter	0.04
σ	neighborhood window size to build C	$1.2i$
W_{nms}	Window size of non-maxima suppression	29
SSD/NCC parameters		
W_{SSD}	Comparison window size of SSD	20
W_{NCC}	Comparison window size of NCC	20
T_{SSD}	SSD threshold	$p \cdot T_{ave}^{SSD}$
T_{NCC}	NCC threshold	$p \cdot T_{ave}^{NCC}$
SIFT parameter		
$PeakThresh$	Peak selection threshold	3
Euclidean distance parameters		
T_{distE}	Euclidean distance threshold	$p \cdot T_{ave}^{distE}$

Table 1: Parameter description and value

5.1 Set 1

Since a pair of image from set 1 has only small moves, the correspondence results are quite good. We use scale for threshold p as 1 for all metric in this set. As we mentioned, we apply four different scales such that $i = 1, 2, 3, 4$. As can be seen from Figures, the number of Harris corner features decreases when scale value increases. This is expected because we convolve larger Haar wavelet to given image pair and it makes Harris corner detect sparser corner when we increase scale. As scale goes up, fewer but better matched corresponding corner pairs are left. Different from Harris corner detector, SIFT found rich feature points from both image pair and matched them nicely. Note that we use Euclidean distance of 128-vector of each feature points and select the point that produces minimum distance .

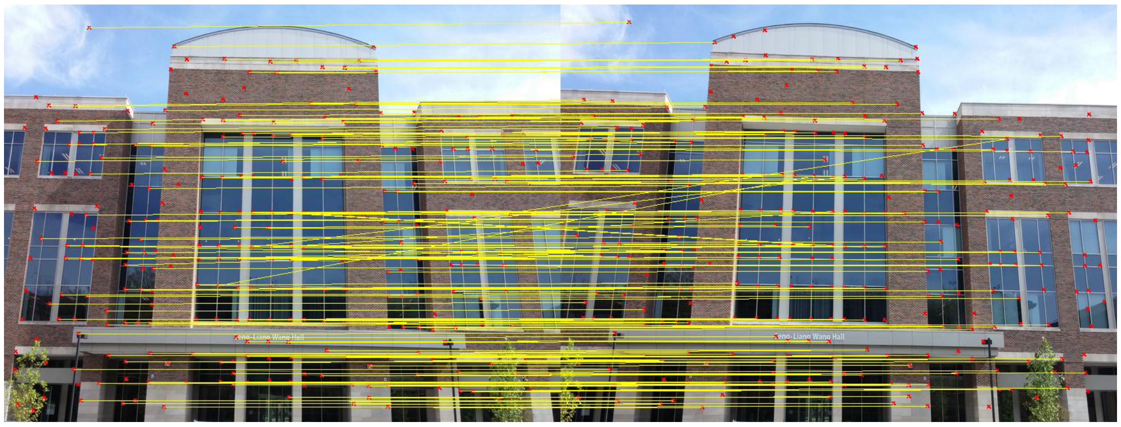

Figure 2: SSD correspondences found with scale 1 in set 1.

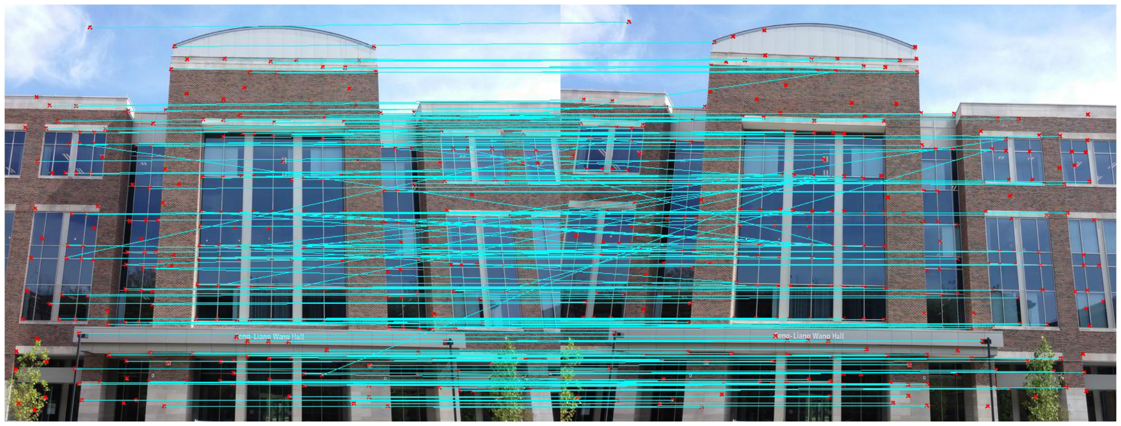

Figure 3: NCC correspondences found with scale 1 in set 1.

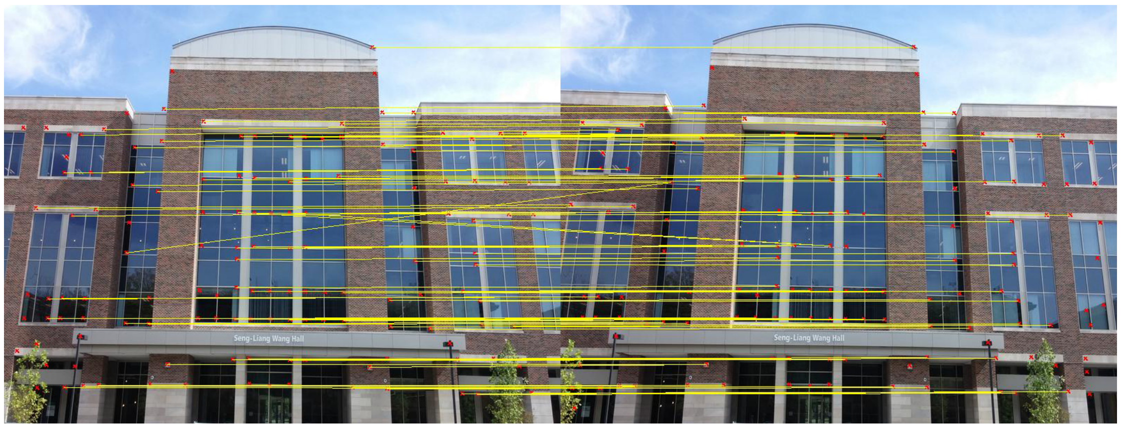

Figure 4: SSD correspondences found with scale 2 in set 1.

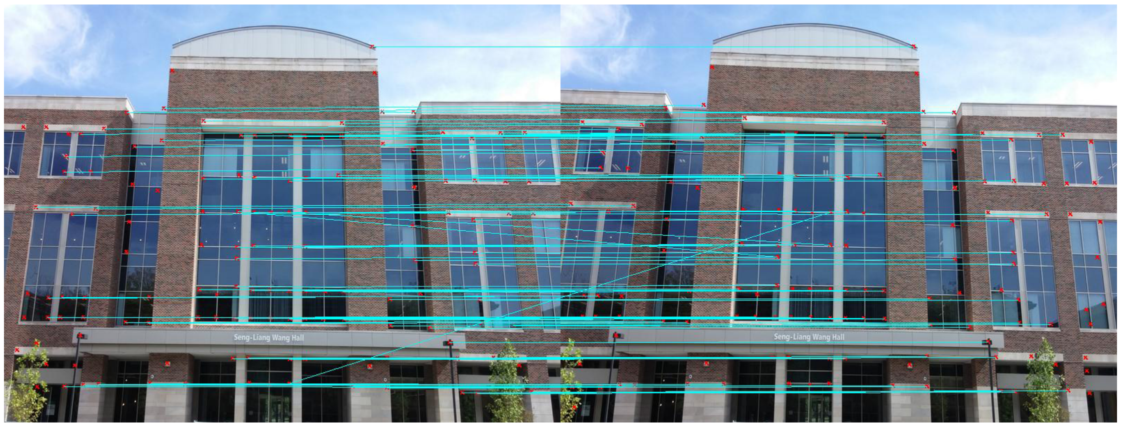

Figure 5: NCC correspondences found with scale 2 in set 1.

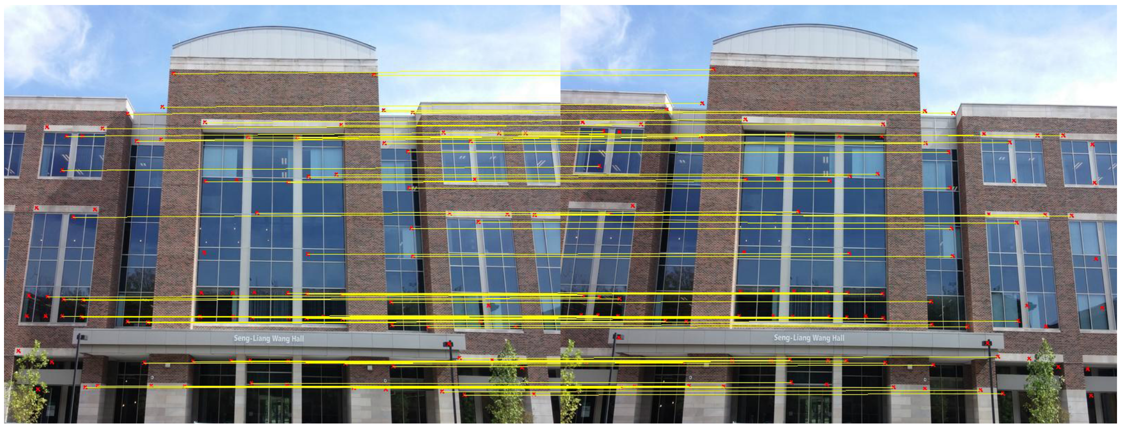

Figure 6: SSD correspondences found with scale 3 in set 1.

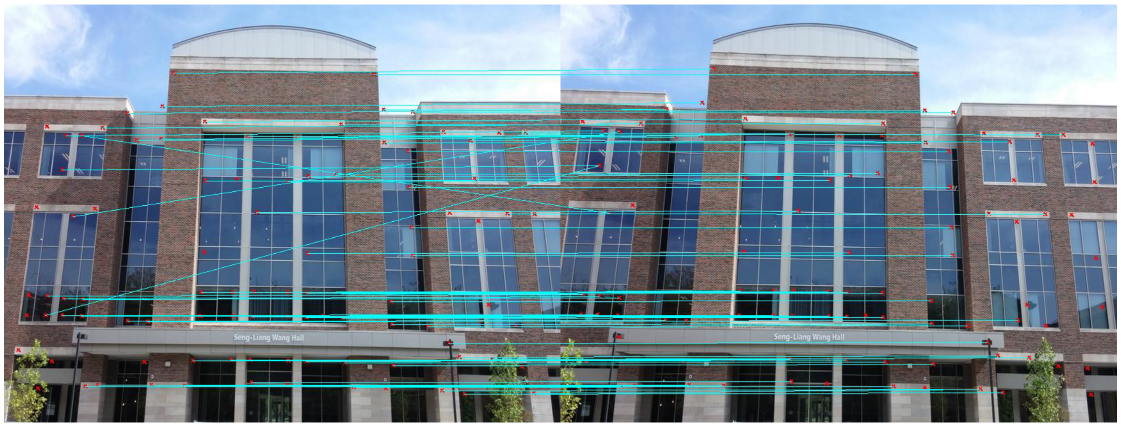

Figure 7: NCC correspondences found with scale 3 in set 1.

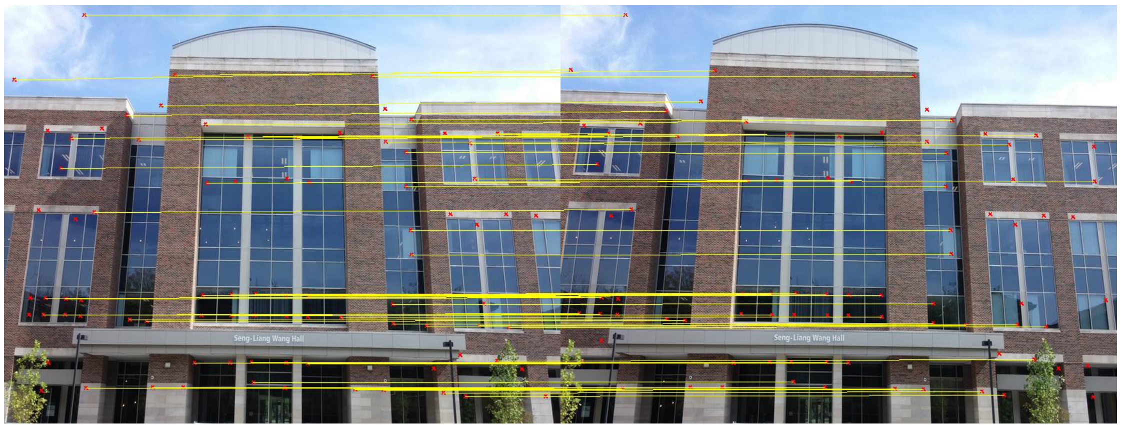

Figure 8: SSD correspondences found with scale 4 in set 1.

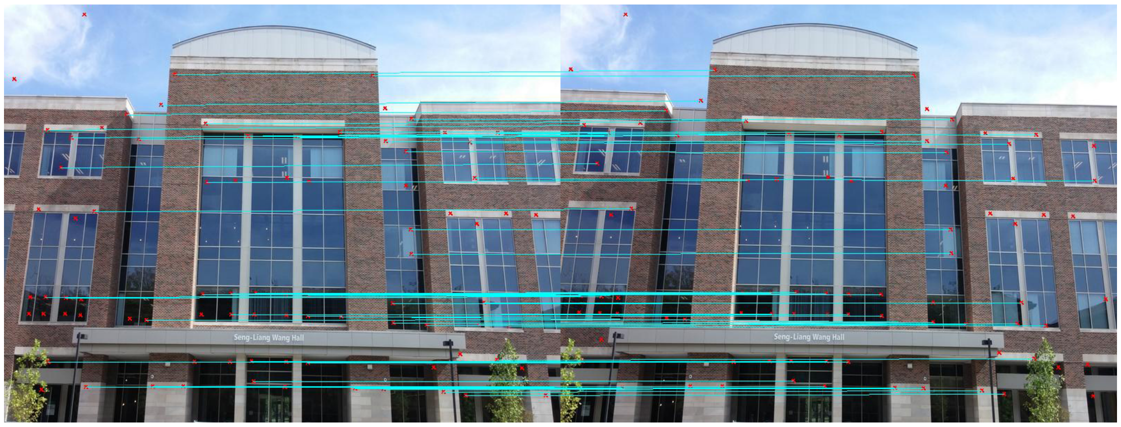

Figure 9: NCC correspondences found with scale 4 in set 1.

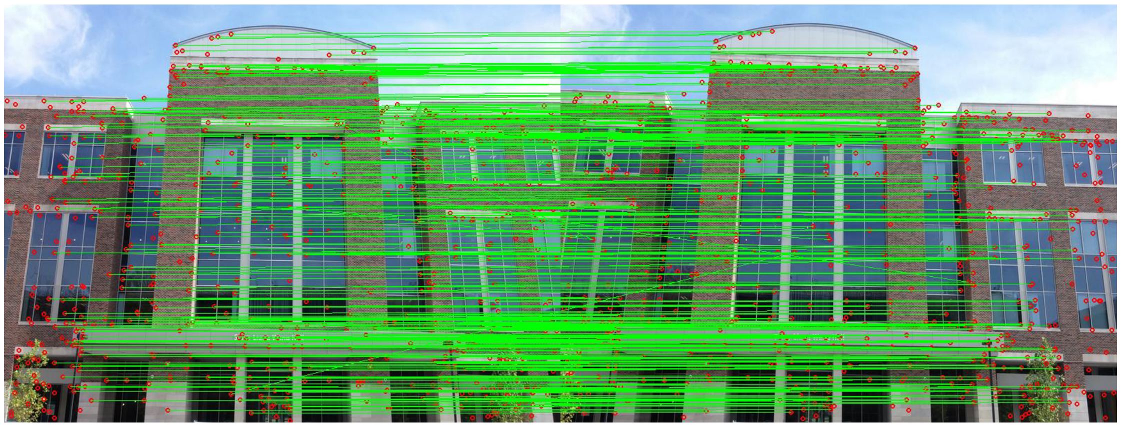

Figure 10: SIFT correspondences found with Euclidean distance metric in set 1.

5.2 Set 2

Different from set 1, image pair of set 2 is very difficult to find matched points due to different illumination condition and different objects. We decide scale for threshold p as 0.6 empirically for all metric in this set. As we mentioned, we apply four different scales such that $i = 1, 2, 3, 4$. As we observed already, larger scale produces less corner points. Note that SSD correspondences with less corner points did not get better. However, NCC correspondences with less corner points clearly shows better matched.

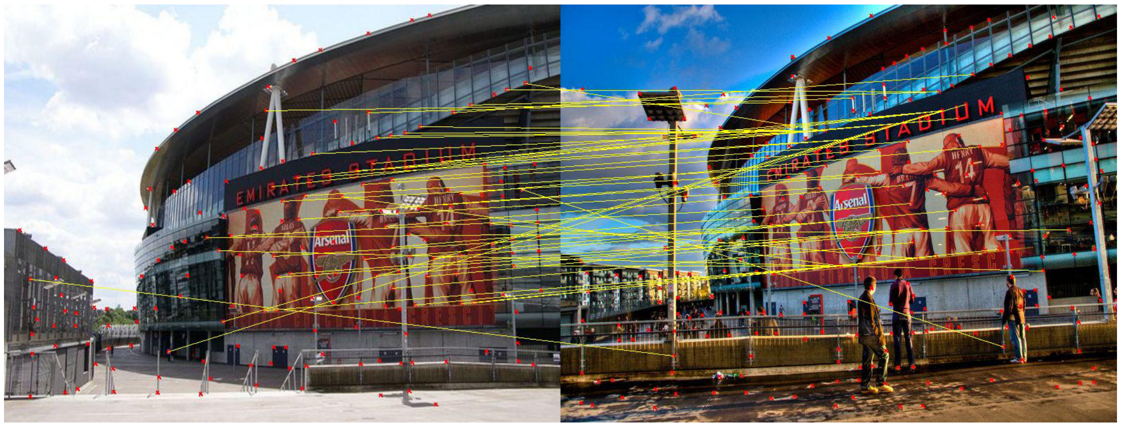
Figure 11: SSD correspondences found with scale 1 in set 2.

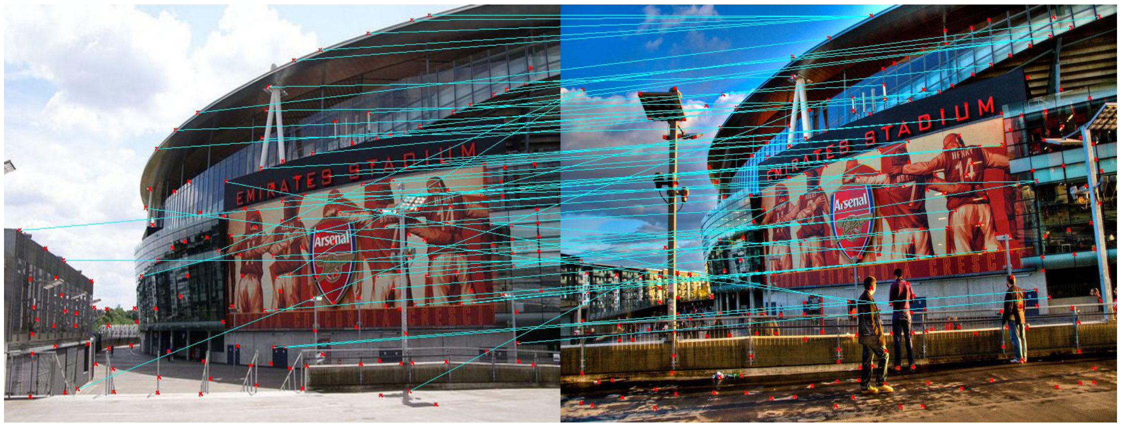
Figure 12: NCC correspondences found with scale 1 in set 2.

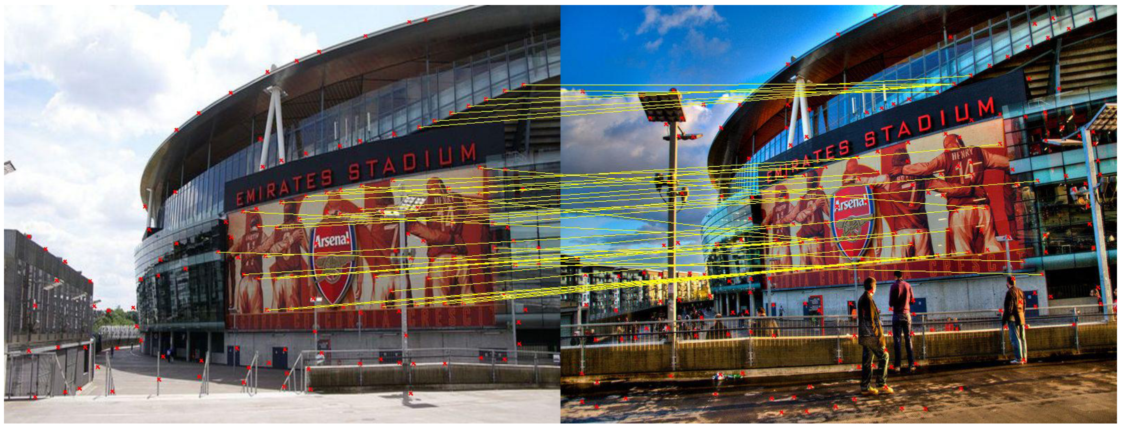

Figure 13: SSD correspondences found with scale 2 in set 2.

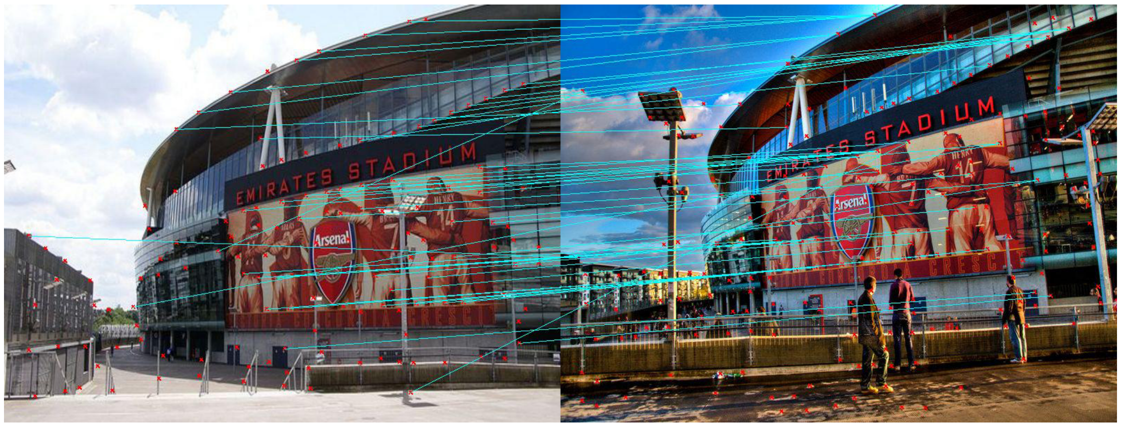

Figure 14: NCC correspondences found with scale 2 in set 2.

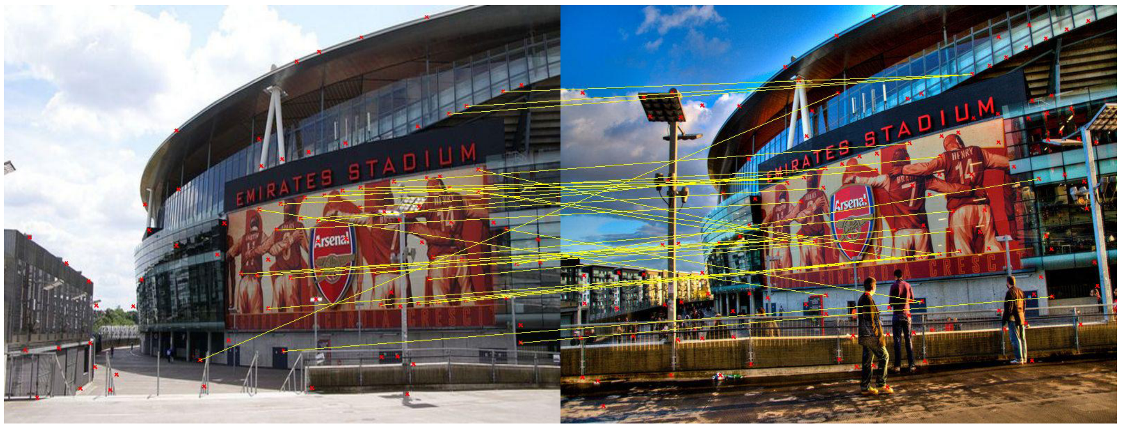

Figure 15: SSD correspondences found with scale 3 in set 2.

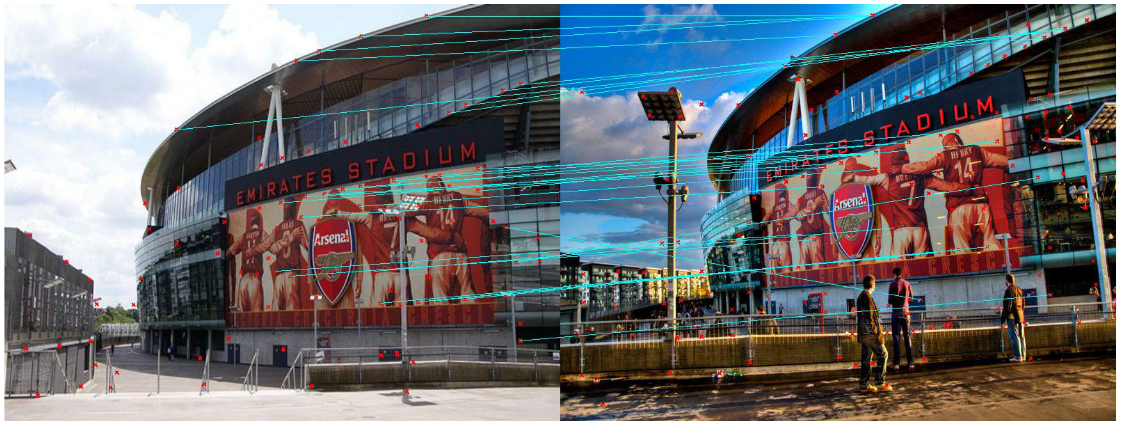

Figure 16: NCC correspondences found with scale 3 in set 2.

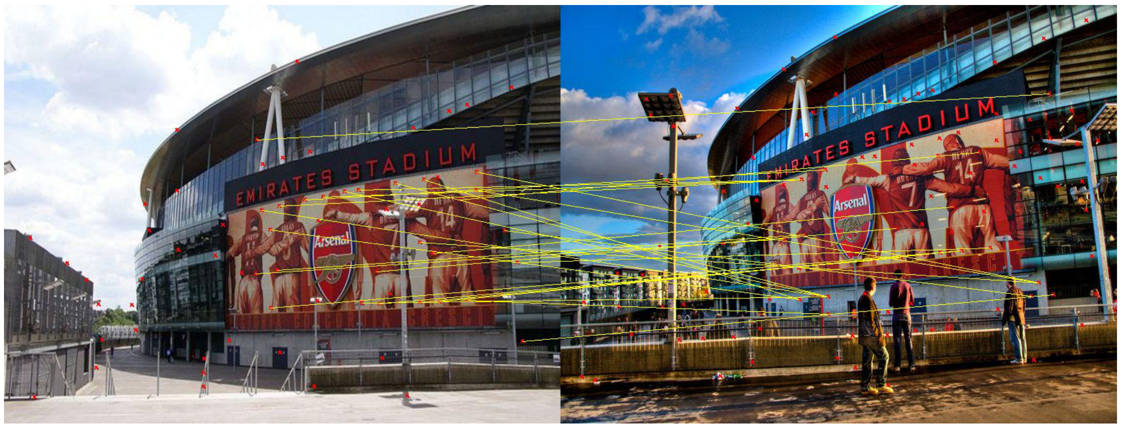

Figure 17: SSD correspondences found with scale 4 in set 2.

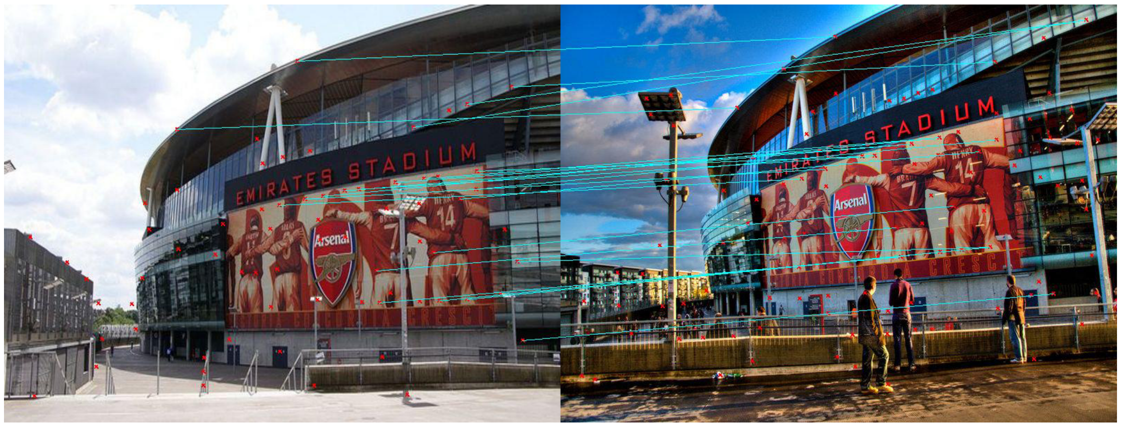

Figure 18: NCC correspondences found with scale 4 in set 2.

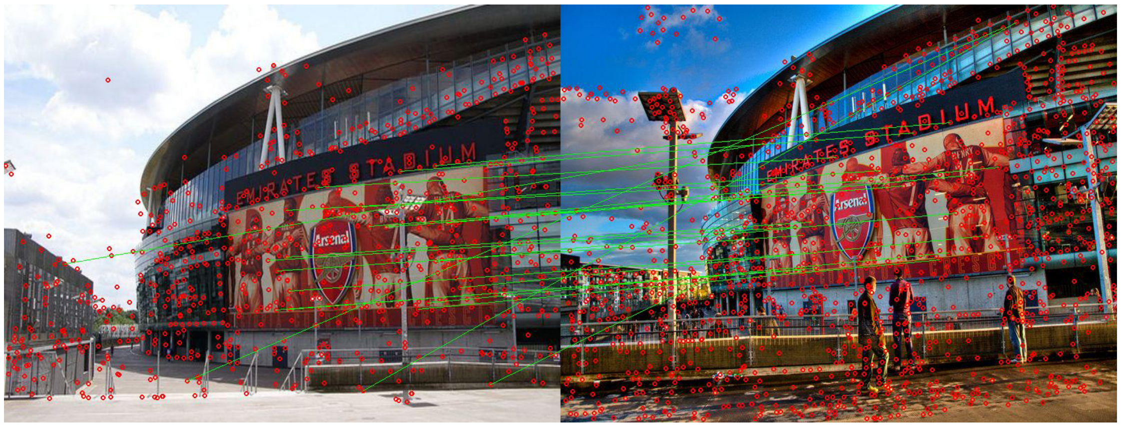

Figure 19: SIFT correspondences found with Euclidean distance metric in set 2.

5.3 Set 3

Set 3 and set 4 are my personal set to apply feature extraction and matched feature points methods. More specifically, we took images from our lab and image pair of set 3 contains letter and time table. Since these images are taken from different orientation, this pair of image has projective distortion. We decide scale for threshold p as 0.8 empirically for all metric in this set. As we mentioned, we apply four different scales such that $i = 1, 2, 3, 4$.

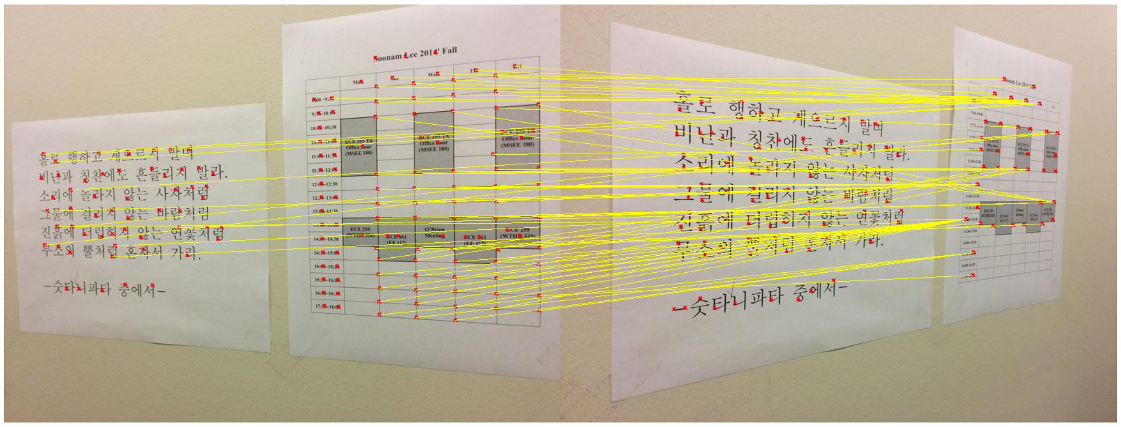

Figure 20: SSD correspondences found with scale 1 in set 3.

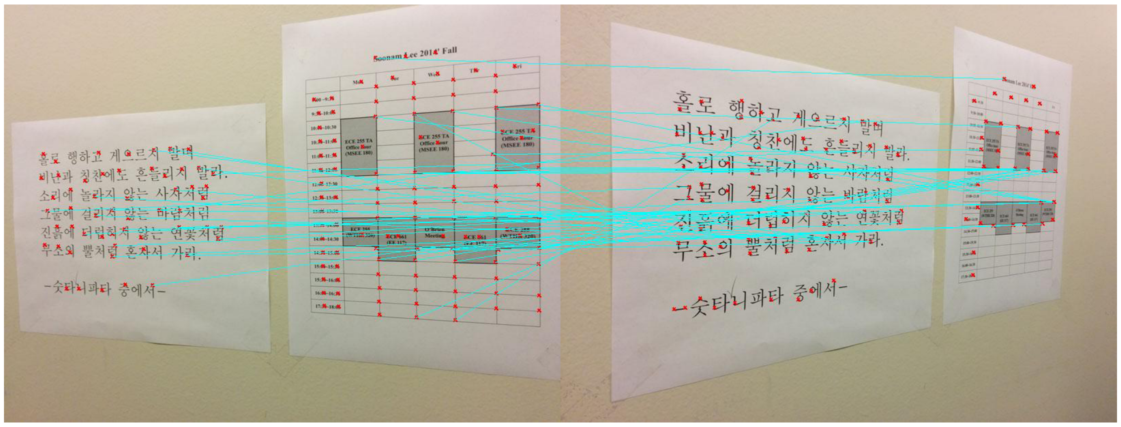

Figure 21: NCC correspondences found with scale 1 in set 3.

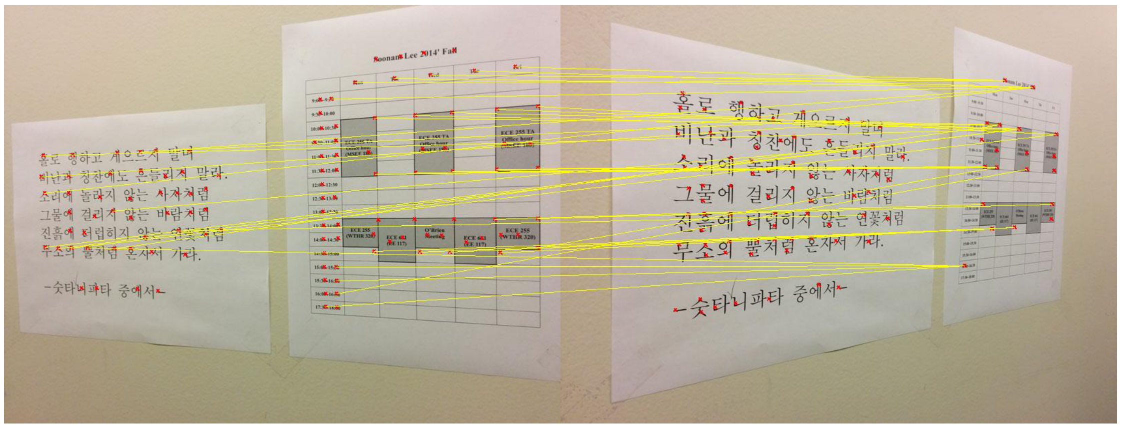
Figure 22: SSD correspondences found with scale 2 in set 3.

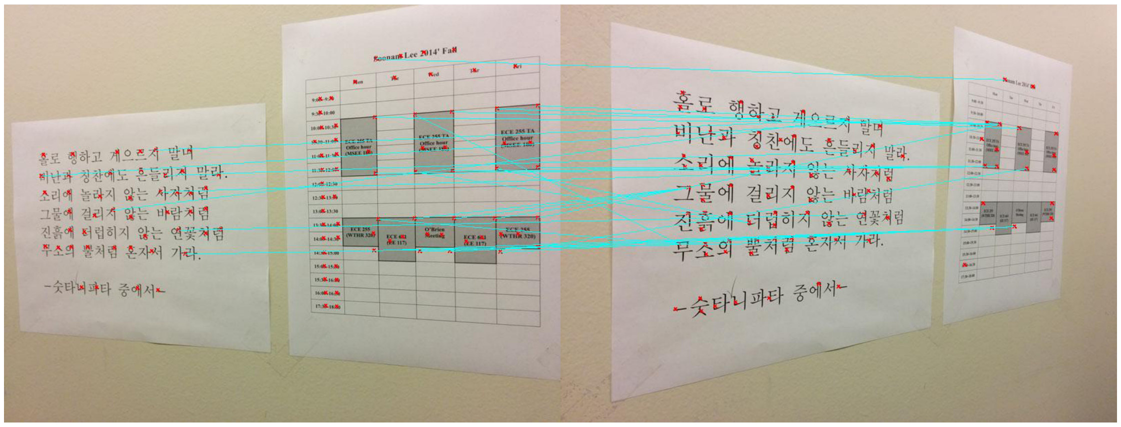
Figure 23: NCC correspondences found with scale 2 in set 3.

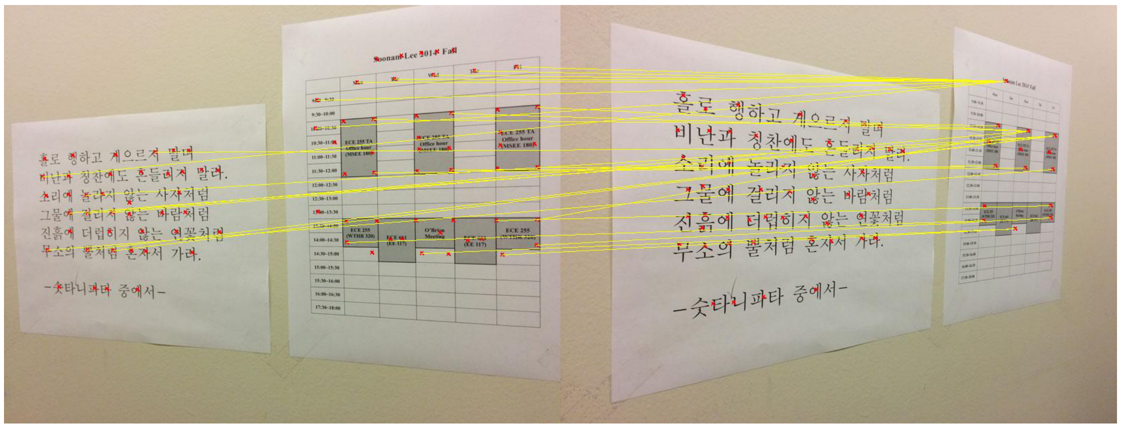

Figure 24: SSD correspondences found with scale 3 in set 3.

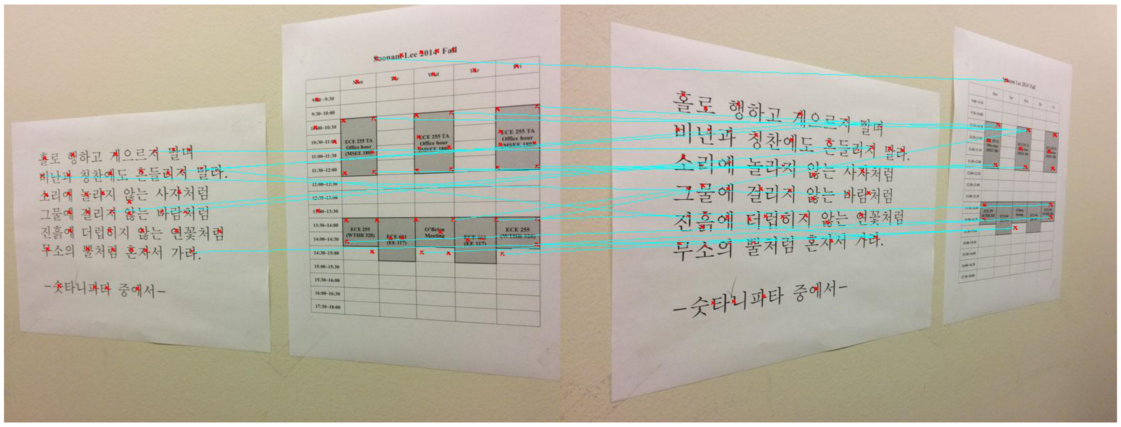

Figure 25: NCC correspondences found with scale 3 in set 3.

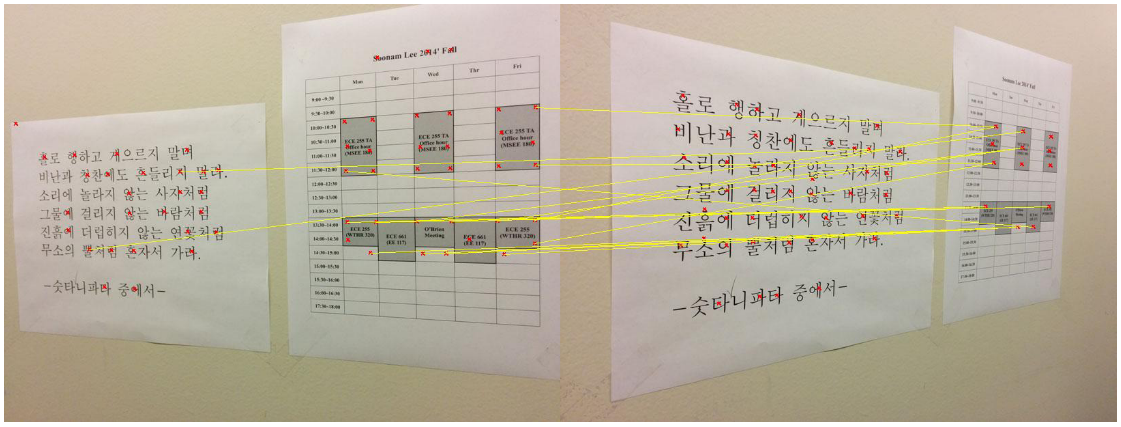
Figure 26: SSD correspondences found with scale 4 in set 3.

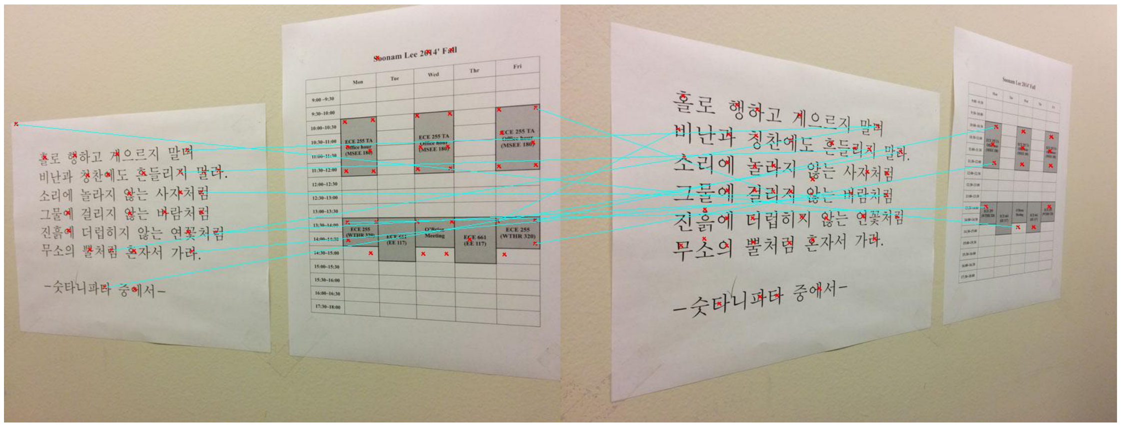
Figure 27: NCC correspondences found with scale 4 in set 3.

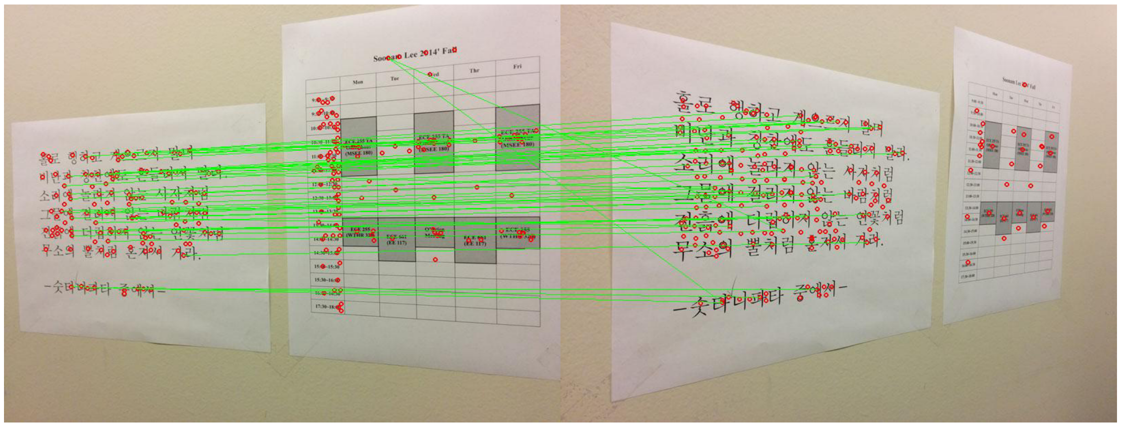
Figure 28: SIFT correspondences found with Euclidean distance metric in set 3.

5.4 Set 4

Set 4 is taken from different orientation with varying rotation and translation. This set turns out to be very difficult set. We decide scale for threshold p as 0.6 empirically for all metric in this set. As we mentioned, we apply four different scales such that $i = 1, 2, 3, 4$.

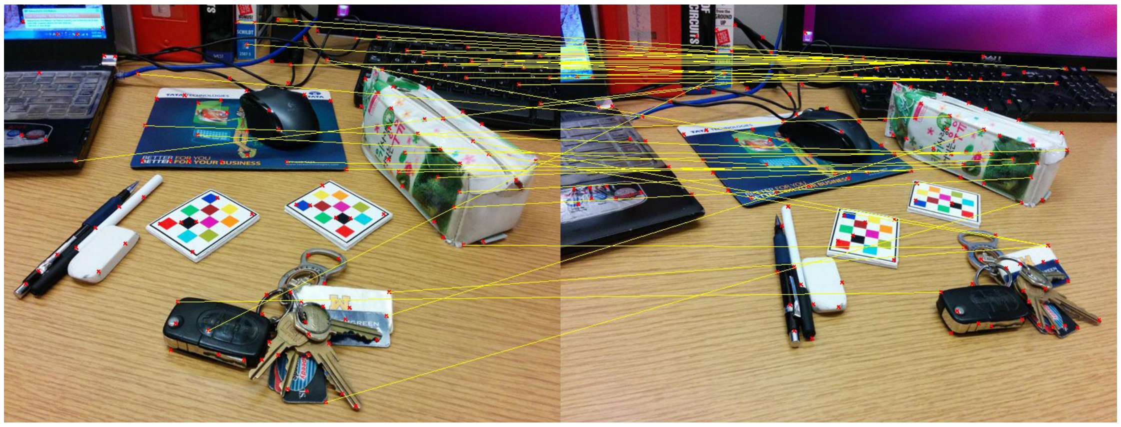

Figure 29: SSD correspondences found with scale 1 in set 4.

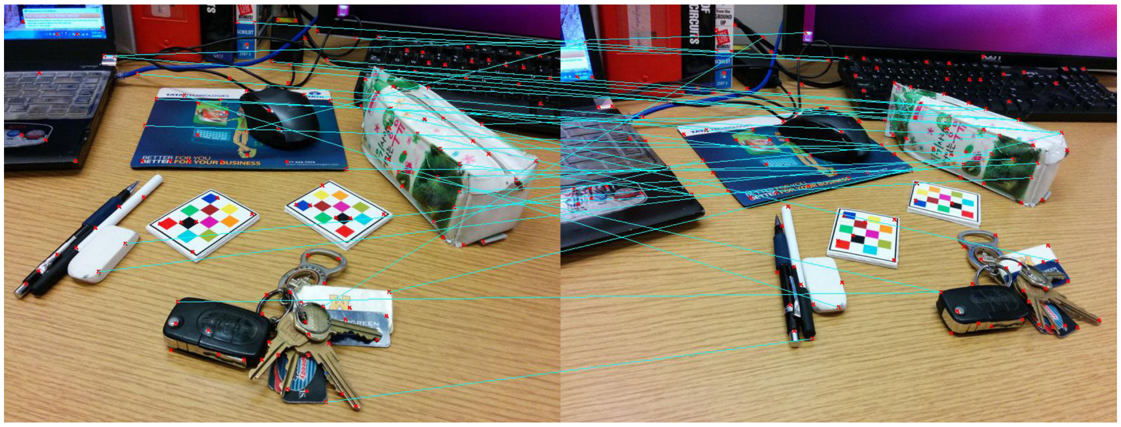

Figure 30: NCC correspondences found with scale 1 in set 4.

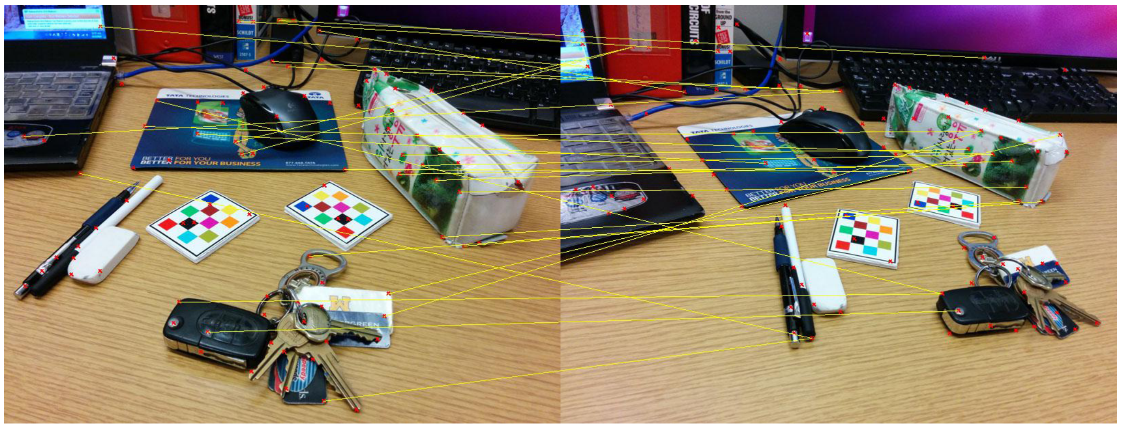
Figure 31: SSD correspondences found with scale 2 in set 4.

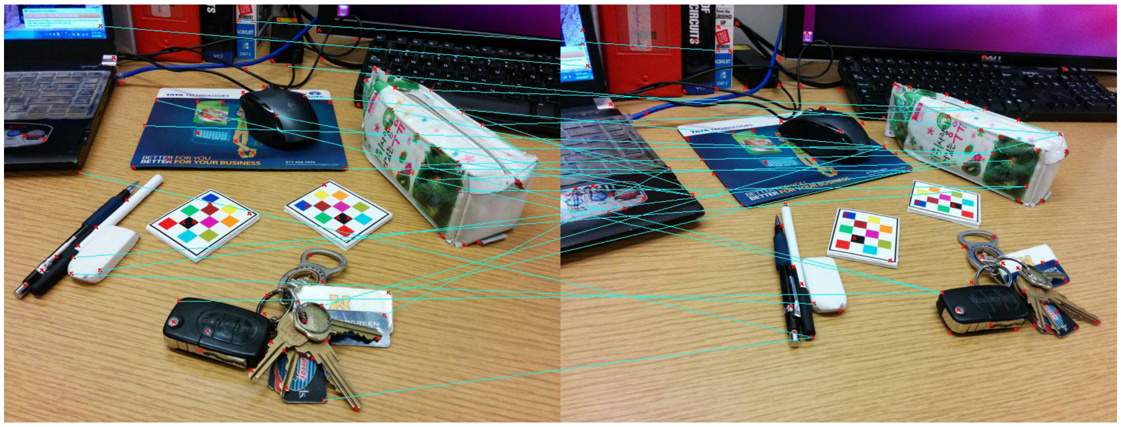
Figure 32: NCC correspondences found with scale 2 in set 4.

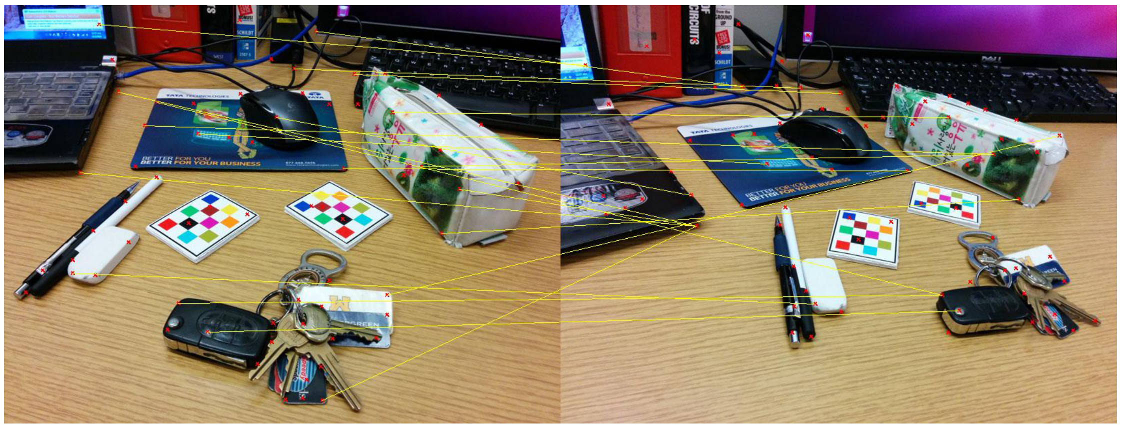
Figure 33: SSD correspondences found with scale 3 in set 4.

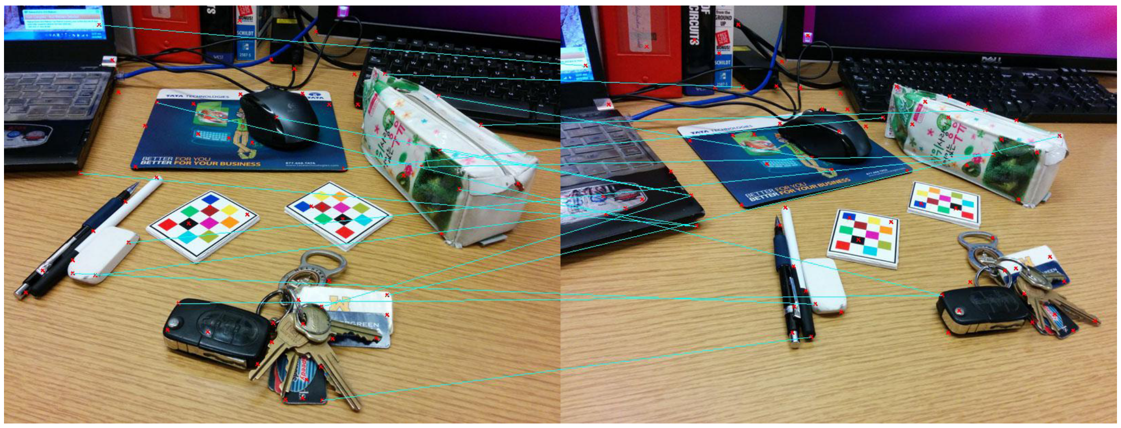
Figure 34: NCC correspondences found with scale 3 in set 4.

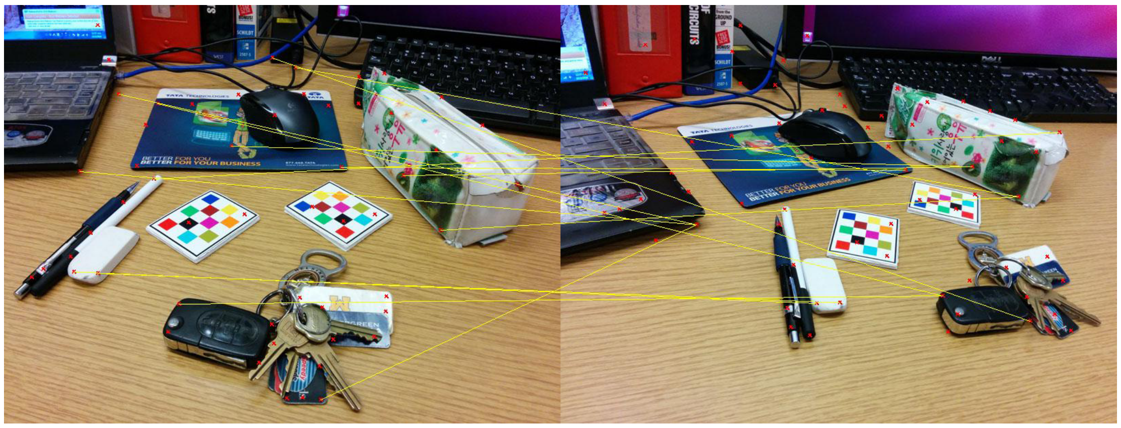

Figure 35: SSD correspondences found with scale 4 in set 4.

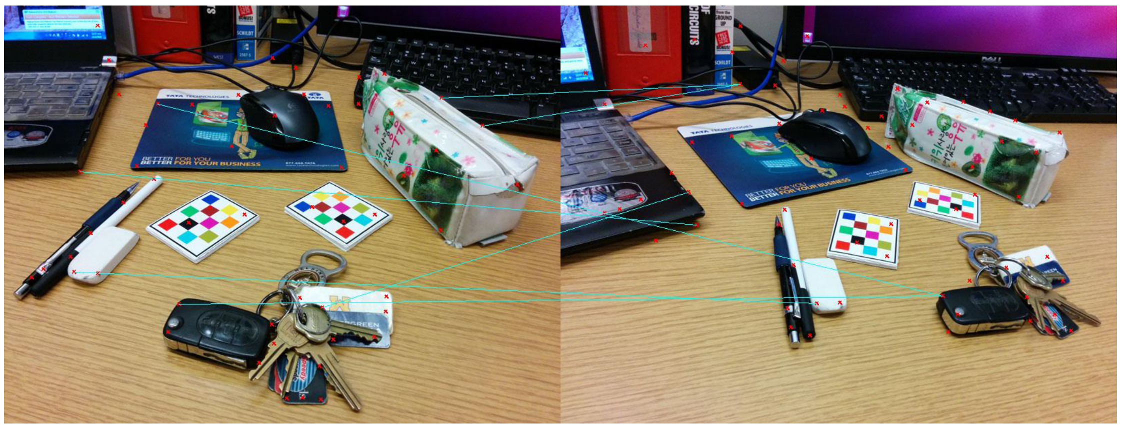

Figure 36: NCC correspondences found with scale 4 in set 4.

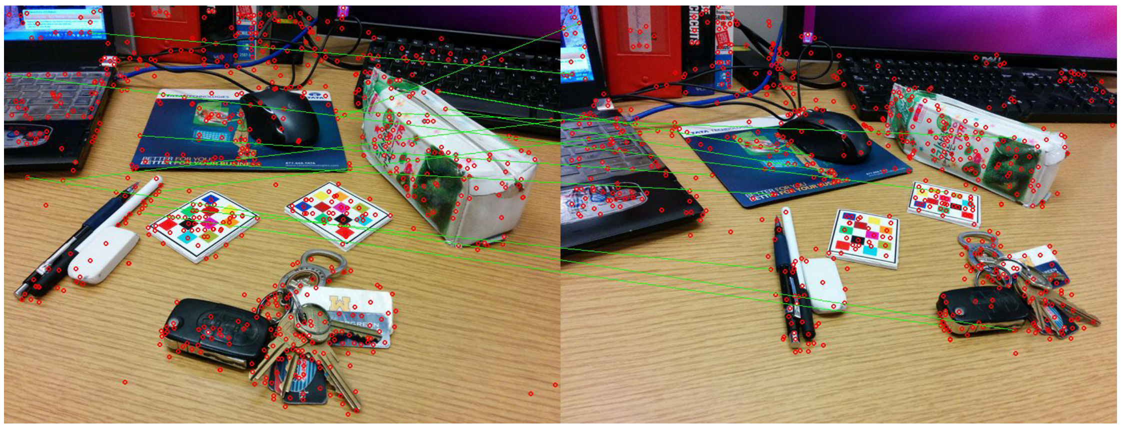

Figure 37: SIFT correspondences found with Euclidean distance metric in set 4.

Appendix: MATLAB Codes

Listing 1: MATLAB code - HW4main.m

```matlab
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% HW4main.m

clc; clear all; close all;

%% main
path = '/home/stargate/a/sig/lee714/coursework/ECE661/Homework/HW4/';
% path = 'D:\Study\Purdue_Study\2014_Fall\ECE661\Homework\HW4\';

bools.Plot = true;
bools.Save = false;

% Set index
idx = 4;

% Load input image
img1_orig = imread([path, 'Pics/Set', num2str(idx), '/pic1.jpg']);
img2_orig = imread([path, 'Pics/Set', num2str(idx), '/pic2.jpg']);
img1 = double(rgb2gray(img1_orig));
img2 = double(rgb2gray(img2_orig));

% Set the parameters
params.scale = 4;
params.sigma = 1.2;
params.k = 0.04;
params.W_nms = 29;
params.W_SSD = 20;
params.W_NCC = 20;
params.p = 0.6;

% Perform Harris Corner Detector
HarrisCorner(img1_orig, img2_orig, params, idx, path, bools);

% Perform SIFT descriptor
SIFTDescriptor(img1_orig, img2_orig, params, idx, path, bools);
```

Listing 2: MATLAB code - HarrisCorner.m

```matlab
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% HarrisCorner.m

function HarrisCorner(img1_orig, img2_orig, params, idx, path, bools)

img1 = double(rgb2gray(img1_orig));
img2 = double(rgb2gray(img2_orig));
% Corner responses map R1, R2
R1 = zeros(size(img1)); R2 = zeros(size(img2));
% Corner location map C1, C2
```

```
C1 = zeros(size(img1)); C2 = zeros(size(img2));

% % normalize image 1 and image 2
% img1 = img1/max(max(img1));
% img2 = img2/max(max(img2));

% Get the size of images
[height, width] = size(img1);
sigma = params.scale*params.sigma;

tic

%% Ix, Iy convlved with Haarx and Haary
% Construct Haar wavelet filter with respect to size of sigma
% Set the S1, the smallest even integer greater than 4*sigma.
W1 = round(ceil(4*sigma)/2)*2;
Haarx = [-ones(W1, W1/2), ones(W1,W1/2)];
Haary = [ones(W1/2, W1); -ones(W1/2, W1)];

Ix1 = imfilter(img1, Haarx, 'same');
Iy1 = imfilter(img1, Haary, 'same');
Ix2 = imfilter(img2, Haarx, 'same');
Iy2 = imfilter(img2, Haary, 'same');

% if bools.Plot
% figure; imshow(uint8(Ix1)); figure; imshow(uint8(Iy1));
% figure; imshow(uint8(Ix2)); figure; imshow(uint8(Iy2));
% end

%% Construct matrix M1 and M2
% window size should always be odd
W2 = round(ceil(5*sigma)/2)*2+1;
half_W2 = round(W2/2)-1;

for i = half_W2+1:width-half_W2-1
    for j = half_W2+1:height-half_W2-1
       %% Image 1
        % Get local window from Ix1, Iy1
        Ix1_sub = Ix1(j-half_W2:j+half_W2, i-half_W2:i+half_W2);
        Iy1_sub = Iy1(j-half_W2:j+half_W2, i-half_W2:i+half_W2);
        % Get M matrix
        M1(1,1) = sum(sum(Ix1_sub.^2));
        M1(1,2) = sum(sum(Ix1_sub.*Iy1_sub));
        M1(2,2) = sum(sum(Iy1_sub.^2));
        % Corner response
        R1(j,i) = det(M1) - params.k*trace(M1)^2;

       %% Image 2
        % Get local window from Ix2, Iy2
        Ix2_sub = Ix2(j-half_W2:j+half_W2, i-half_W2:i+half_W2);
        Iy2_sub = Iy2(j-half_W2:j+half_W2, i-half_W2:i+half_W2);
        % Get M matrix
        M2(1,1) = sum(sum(Ix2_sub.^2));
        M2(1,2) = sum(sum(Ix2_sub.*Iy2_sub));
        M2(2,2) = sum(sum(Iy2_sub.^2));
        % Corner response
        R2(j,i) = det(M2) - params.k*trace(M2)^2;
    end
end

% Window size for non-maximum suppresion of corner response
W_nms = params.W_nms;
half_W_nms = round(W_nms/2)-1;
for i = half_W_nms+1:width-half_W_nms-1
    for j = half_W_nms+1:height-half_W_nms-1
        % Get local window from corner response R1
        R1_sub = R1(j-half_W_nms:j+half_W_nms, i-half_W_nms:i+half_W_nms);
        % Non-maxima suppresion and thresholding
        if (R1(j,i) == max(max(R1_sub)) && R1(j,i) > mean(mean(abs(R1))))
            C1(j,i) = 1;
        end

        % Get local window from corner response R2
        R2_sub = R2(j-half_W_nms:j+half_W_nms, i-half_W_nms:i+half_W_nms);
        % Non-maxima suppresion and thresholding
        if (R2(j,i) == max(max(R2_sub)) && R2(j,i) > mean(mean(abs(R2))))
            C2(j,i) = 1;
        end
    end
end

t = toc;

[col1, row1] = find(C1 == 1);
[col2, row2] = find(C2 == 1);

% if bools.Plot   % plot image
% figure;
% imagesc([img1_orig, img2_orig]);
% hold on;
% plot(row1, col1,'rx', 'LineWidth',2);
% plot(row2+width, col2, 'rx', 'LineWidth',2);
% axis off; hold off;
% end

%% Match corresponding corners
% SSD
[pts1, pts2] = SSD(img1, img2, C1, C2, params);

if bools.Plot   % plot image
hfig1 = figure; imshow([img1_orig, img2_orig]);
set(gca,'Position',[0 0 1 1]);
set(gcf,'Position',[0 0 2*width+1 height+1]);
set(gcf, 'PaperPositionMode', 'auto');
truesize(hfig1,[height 2*width]);
hold on;
% Save the shown figure itself
F = getframe(hfig1);
plot(row1,col1,'rx','LineWidth',2);
plot(row2+width,col2,'rx','LineWidth',2);
for i=1:min(length(pts1), length(pts2))
plot([pts1(i,1), pts2(i,1)+width], [pts1(i,2), pts2(i,2)], '-y');
```

```
end
axis off; hold off;
end
if bools.Save   % save image
    imwrite(F.cdata,[path,'Pics/Set',num2str(idx),'/Harris_SSDMatched.jpg'], ...
        'jpeg','Quality',100);
end

% NCC
[pts3, pts4] = NCC(img1, img2, C1, C2, params);

if bools.Plot   % plot image
hfig2 = figure; imshow([img1_orig, img2_orig]);
set(gca,'Position',[0 0 1 1]);
set(gcf,'Position',[0 0 2*width+1 height+1]);
set(gcf, 'PaperPositionMode', 'auto');
truesize(hfig2,[height 2*width]);
hold on;
% Save the shown figure itself
F = getframe(hfig2);
plot(row1,col1,'rx','LineWidth',2);
plot(row2+width,col2,'rx','LineWidth',2);
for i=1:min(length(pts3), length(pts4))
plot([pts3(i,1), pts4(i,1)+width], [pts3(i,2), pts4(i,2)], '-c');
end
axis off; hold off;
end
if bools.Save   % save image
    imwrite(F.cdata,[path,'Pics/Set',num2str(idx),'/Harris_NCCMatched.jpg'],...
        'jpeg','Quality',100);
end
```

Listing 3: MATLAB code - SIFTDescriptor.m

```
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% SIFTDescriptor.m

function SIFTDescriptor(img1_orig, img2_orig, params, idx, path, bools)

img1 = double(rgb2gray(img1_orig));
img2 = double(rgb2gray(img2_orig));
% Get the size of images
[height, width] = size(img1);

% Use vl_sift
[f1,d1] = vl_sift(single(img1),'PeakThresh',3);
[f2,d2] = vl_sift(single(img2),'PeakThresh',3);

[pts1, pts2] = distEuclidean(f1, f2, d1, d2, params);

pts1 = round(pts1);
pts2 = round(pts2);

% if bools.Plot   % plot image
% figure;
% imagesc([img1_orig, img2_orig]);
% hold on;
% plot(round(f1(1,:)), round(f1(2,:)), 'ro','LineWidth',2);
% plot(round(f2(1,:))+width, round(f2(2,:)), 'ro','LineWidth',2);
% % plot(pts1(:,2), pts1(:,1), 'ro','LineWidth',2);
% % plot(pts2(:,2)+width, pts2(:,1), 'ro','LineWidth',2);
% axis off; hold off;
% end

if bools.Plot   % plot image
hfig1 = figure; imshow([img1_orig, img2_orig]);
set(gca,'Position',[0 0 1 1]);
set(gcf,'Position',[0 0 2*width+1 height+1]);
set(gcf, 'PaperPositionMode', 'auto');
truesize(hfig1,[height 2*width]);
hold on;
% Save the shown figure itself
F = getframe(hfig1);
plot(round(f1(1,:)), round(f1(2,:)), 'ro','LineWidth',2);
plot(round(f2(1,:))+width, round(f2(2,:)), 'ro','LineWidth',2);
% plot(pts1(:,2), pts1(:,1),'ro','LineWidth',2);
% plot(pts2(:,2)+width, pts2(:,1), 'ro','LineWidth',2);
for i=1:min(length(pts1), length(pts2))
plot([pts1(i,2), pts2(i,2)+width], [pts1(i,1), pts2(i,1)], '-g');
end
axis off; hold off;
end
if bools.Save   % save image
    imwrite(F.cdata,[path,'Pics/Set',num2str(idx),'/SIFT_EuclideanMatched.jpg'], ...
        'jpeg','Quality',100);
end
```

Listing 4: MATLAB code - SSD.m

```
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% SSD.m

function [pts1, pts2] = SSD(img1, img2, C1, C2, params)

[col1, row1] = find(C1 == 1);
[col2, row2] = find(C2 == 1);
pts1 = NaN(length(col1),2);
pts2 = NaN(length(col2),2);
W_SSD = params.W_SSD + 1;
half_W_SSD = round(W_SSD/2)-1;

for j = 1:length(col2)
    j
    for i = 1:length(col1)
        % Get local neighbors from corner and compare the gray level image
        % around an neighbors
        sub1 = img1(col1(i)-half_W_SSD:col1(i)+half_W_SSD, ...
    row1(i)-half_W_SSD:row1(i)+half_W_SSD);
```

```
        sub2 = img2(col2(j)-half_W_SSD:col2(j)+half_W_SSD, ...
    row2(j)-half_W_SSD:row2(j)+half_W_SSD);
        SSD_sub = (abs(sub1 - sub2)).^2;
        SSD(j,i) = sum(sum(SSD_sub));
    end
end

% Want to find the index of minimum differences
[value, idx] = min(SSD);
T_SSD = params.p*mean(value);
% Thresholding using T_SSD
newcol1 = col1; newrow1 = row1;
newcol1(find(value > T_SSD)) = NaN;
newrow1(find(value > T_SSD)) = NaN;
newcol1(isnan(newcol1),:) = [];
newrow1(isnan(newrow1),:) = [];

% find index that corresponding thresholded points
idx(find(value > T_SSD)) = NaN;

cnt = 1;
for i=1:length(idx)
    if ~(isnan((idx(i))))
        newrow2(cnt) = row2(idx(i));
        newcol2(cnt) = col2(idx(i));
        cnt = cnt+1;
    end
end

% Get pts1 and pts2
pts1 = [newrow1, newcol1];
pts2 = [newrow2', newcol2'];
```

Listing 5: MATLAB code - NCC.m

```
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% NCC.m

function [pts3, pts4] = NCC(img1, img2, C1, C2, params)

[col1, row1] = find(C1 == 1);
[col2, row2] = find(C2 == 1);
W_NCC = params.W_NCC + 1;
half_W_NCC = round(W_NCC/2)-1;

for j = 1:length(col2)
    j
    for i = 1:length(col1)
        % Get local neighbors from corner and compare the gray level image
        % around an neighbors
        sub1 = img1(col1(i)-half_W_NCC:col1(i)+half_W_NCC, ...
    row1(i)-half_W_NCC:row1(i)+half_W_NCC);
        sub2 = img2(col2(j)-half_W_NCC:col2(j)+half_W_NCC, ...
    row2(j)-half_W_NCC:row2(j)+half_W_NCC);
```

```
        m1 = mean(mean(sub1));  m2 = mean(mean(sub2));
        deviation1 = sub1 - m1; deviation2 = sub2 - m2;
        NCC_numer = sum(sum(deviation1.*deviation2));
        NCC_denom = sqrt(sum(sum(deviation1.^2)) * sum(sum(deviation2.^2)));
        NCC_sub = NCC_numer / NCC_denom;
        NCC(j,i) = NCC_sub;
    end
end

% Want to find the index that NCC is close to 1
[value, idx] = min(abs(NCC-1));
T_NCC = params.p*mean(value);
% Thresholding using T_NCC
newcol1 = col1; newrow1 = row1;
newcol1(find(value > T_NCC)) = NaN;
newrow1(find(value > T_NCC)) = NaN;
newcol1(isnan(newcol1),:) = [];
newrow1(isnan(newrow1),:) = [];

% find index that corresponding thresholded points
idx(find(value > T_NCC)) = NaN;

cnt = 1;
for i=1:length(idx)
    if ~(isnan((idx(i))))
        newrow2(cnt) = row2(idx(i));
        newcol2(cnt) = col2(idx(i));
        cnt = cnt+1;
    end
end

% Get pts1 and pts2
pts3 = [newrow1, newcol1];
pts4 = [newrow2', newcol2'];
```

Listing 6: MATLAB code - distEuclidean.m

```
%% ECE 661 HW #4
% PUID: 0023094873
% name: Soonam Lee
% distEuclidean.m

function [pts1, pts2] = distEuclidean(f1, f2, d1, d2, params)

pts1 = NaN(length(d1),2);
pts2 = NaN(length(d2),2);

% Euclidean distance
for j = 1:size(d2,2)
    j
    for i = 1:size(d1,2)
        distE(j,i) = sqrt(sum((d1(:,i)-d2(:,j)).^2));
    end
end

% Want to find the index of minimum differences
```

```
[value, idx] = min(distE);
T_distE = params.p*mean(value);
% Thresholding using T_distE
newcol1 = f1(1,:)'; newrow1 = f1(2,:)';
newcol1(find(value > T_distE)) = NaN;
newrow1(find(value > T_distE)) = NaN;
newcol1(isnan(newcol1),:) = [];
newrow1(isnan(newrow1),:) = [];

% find index that corresponding thresholded points
idx(find(value > T_distE)) = NaN;

cnt = 1;
for i=1:length(idx)
    if ~(isnan((idx(i))))
        newrow2(cnt) = f2(2,idx(i));
        newcol2(cnt) = f2(1,idx(i));
        cnt = cnt+1;
    end
end

% Get pts1 and pts2
pts1 = [newrow1, newcol1];
pts2 = [newrow2', newcol2'];
```