ECE 661 : HW4

Yellamraju Tarun (ytarun@purdue.edu)

September 29, 2016

1 Harris Corner Detector

The Harris Corner Detector uses derivatives in x and y directions in order to detect robust interest points. The derivatives need to be scaled for different values of σ and so we form the derivative filter kernels for x and y using the Haar filters.

The Haar filter kernels have a size of smallest even integer greater than 4σ and are an extrapolation of the basic forms of the Haar wavelets along the x and y directions. As an example, for $\sigma = 1.2$, we get

$$\frac{\partial}{\partial x} = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \tag{1}$$

$$\frac{\partial}{\partial y} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{2}$$

Once the derivatives are computed, we form the C matrix for each pixel using a neighborhood of size $5\sigma \times 5\sigma$ where we enforce 5σ to be odd to get a unique center pixel. The C matrix is defined as

$$C = \begin{bmatrix} \sum d_x^2 & \sum d_x d_y \\ \sum d_x d_y & \sum d_y^2 \end{bmatrix} \tag{3}$$

where d_x and d_y denote the output of the derivative filters over the chosen neighborhood.

An interest point is completely characterized by the ratio of the eigenvalues of C , $r = \frac{\lambda_2}{\lambda_1}$ with $\lambda_1 > \lambda_2$ which can be subject to a threshold to detect interest points. However, since it computationally expensive to calculate the eigenvalues

at every pixel location, we use a proxy for the ratio r. We instead compute and apply a threshold to

$$\frac{r}{(1+r)^2} = \frac{\lambda_1\lambda_2}{(\lambda_1+\lambda_2)^2} = \frac{det(C)}{trace(C)^2} \tag{4}$$

which is computationally cheaper since it is directly in terms of the elements of the matrix C.

Once we have established interest points using the above method, we tend to get clusters of interest points along edges and corners which need to be filtered out. This is done through non-maxima suppression so that only local maxima (in terms of the ratio) interest points are retained.

2 Establishing Point Correspondences

2.1 NCC : Normalized Cross Correlation

Once we have established interest points in 2 images of the same scene, we can establish correspondences through the NCC metric applied over an $(M+1) \times (M+1)$ window around the points being considered. NCC is defined as follows

$$NCC = \frac{\sum_i \sum_j (f_1(i,j) - m_1)(f_2(i,j) - m_2)}{\sqrt{(\sum_i \sum_j (f_1(i,j) - m_1)^2)(\sum_i \sum_j (f_2(i,j) - m_2)^2)}} \tag{5}$$

where f_i is the window around the point in the i^{th} image and m_i is the mean of the pixels in that window. For a point to be matched, the NCC must be high. Since the NCC lies between 0 and 1, we can choose a suitable threshold to determine point correspondences.

In my experiments, I thresholded the NCC at 0.95 which produced accurate correspondences for all image pairs.

2.2 SSD : Sum of Squared Differences

Another metric used for establishing correspondences is the SSD which is also applied over an $(M+1) \times (M+1)$ window around the points being considered. SSD is defined as follows

$$SSD = \sum_i \sum_j |f_1(i,j) - f_2(i,j)|^2 \tag{6}$$

where f_i is the window around the point in the i^{th} image. For a point to be matched, the SSD must be low. We can threshold the SSD to determine point correspondences.

In my experiments, I thresholded the SSD at 10.min(SSD) which produced accurate correspondences for all image pairs. In this way, the threshold is dynamic and does not need to tweaked for every image pair and produces accurate correspondences generally

3 SIFT : Scale Invariant Feature Transform

The Scale Invariant Feature Transform is one of the most popular interest point detectors used in computer vision. For our experiments, we used the readily available implementation of SIFT found in the VLFeat library.

SIFT is a rather complicated algorithm if one considers all the details involved. Since it has been covered in thorough detail in class, we will provide a brief overview of the highlights of the algorithm.

- SIFT is based entirely on the scale space analysis to identify interest points. Specifically, it utilizes the DoG pyramid discussed in class. The SIFT interest points are the points of local extrema in the scale space $D(x, y, \sigma)$ which are found by comparisons in a $3 \times 3 \times 3$ volumetric neighborhood in the scale space.

- As the σ increases in the scale space, the spatial resolution gets coarser and in order to find accurate locations of the extrema in the original resolution, the scale space is approximated near the point of extrema using the Taylor Series expansion as

$$D(\vec{X}) \approx D(\vec{X_0}) + J^T(\vec{X_0})\vec{X} + \frac{1}{2}\vec{X}^T H(\vec{X_0})\vec{X} \tag{7}$$

  where J is the gradient vector evaluated at the extrema $\vec{X_0} = (x_0, y_0, \sigma_0)^T$ and H is the Hessian evaluated at $\vec{X_0}$. The higher order terms are neglected in this approximation.

  Now, to find accurate locations, the condition $\frac{\partial D(\vec{X})}{\partial \vec{X}} = 0$ is imposed which yields the extrema as $\vec{X} = -H^{-1}(\vec{X_0})J(\vec{X_0})$

- Weak Extrema points are filtered out by thresholding $|D(\vec{X})| < 0.03$ which results in removal of points along edges.

- Once we get points of extrema, we assign a dominant local orientation of the gradient at the point of extrema as

$$m(x, y) = \sqrt{|ff(x+1, y, \sigma) - ff(x, y, \sigma)|^2 + |ff(x, y+1, \sigma) - ff(x, y, \sigma)|^2} \tag{8}$$

$$\theta(x, y) = \arctan \frac{ff(x, y+1, \sigma) - ff(x, y, \sigma)}{ff(x+1, y, \sigma) - ff(x, y, \sigma)} \tag{9}$$

- Without going into the specific details, once we get a dominant orientation, a histogram of orientations for pixels in a neighborhood around the point of extrema is created where the direction of reference is the dominant direction that we determined. Stringing together these histogram entries generates the SIFT feature vector at the location of the point of extrema.

- It is interesting to note that since the orientations are measured using the dominant orientation as a reference, the histograms generated are invariant to in plane rotations. The feature vectors are also normalized to have a unit magnitude which makes them invariant to illumination changes. Hence the feature vector formed is highly robust.

4 Establishing Point Correspondences for SIFT

Since SIFT gives highly robust feature vectors at every interest point, we can establish correspondences between points by simply calculating the euclidian distance between their respective feature vectors and applying a threshold to it. If d_i is the feature vector corresponding to an interest point in the i^{th} image, then the euclidian distance is defined as the L2 norm

$$EuclidianDist = ||d_1 - d_2||_2 \tag{10}$$

Note that using the Euclidian distance metric is the same as using the SSD metric since it is just the square root of the SSD.

$$EuclidianDist = \sqrt{SSD} \tag{11}$$

In my experiments, I thresholded the euclidian distance at 3.min(Euclidian Distance) which produced accurate correspondences for all image pairs. In this way, the threshold is dynamic and does not need to tweaked for every image pair and produces accurate correspondences generally

In addition, I also used the NCC metric for SIFT correspondences with a threshold of 0.99 which produced more accurate correspondences than the Euclidian distance.

5 Experiments

5.1 Observations and Comparisons of the Methods

For the Harris detector, we see that as the scale increases, the number of detected interest points located along edges or what appears to be fine texture reduce. In a sense, these points are not very robust. On the other hand, interest points located at corners and other prominent features tend to have a high response to the Harris Detector and are found to be detected at all the scales used for the experiments.

We observe that in general SIFT is more robust than Harris and detects and matches a lot more interest points than the Harris detector does. Further, Harris detector's performance is highly dependent on the parameters used for the scales and matching algorithms. In my experiments I have used a single set of parameters for all image pairs which consequently produces better results in some particular cases as compared to the rest. Harris detector also picks up a lot

of false positive interest points which need to be filtered out using non-maxima suppression.

We further notice that the NCC metric is more accurate in establishing point correspondences in comparison to the SSD metric. SSD is highly dependent on the parameters used and is not as robust as the NCC over a wide range or perspective differences between 2 images.

5.2 Pair 1

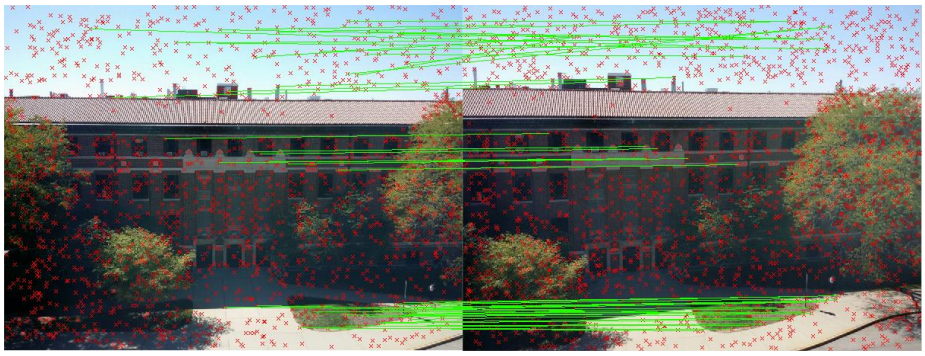

Figure 1: SIFT image with correspondences using Euclidian Distance

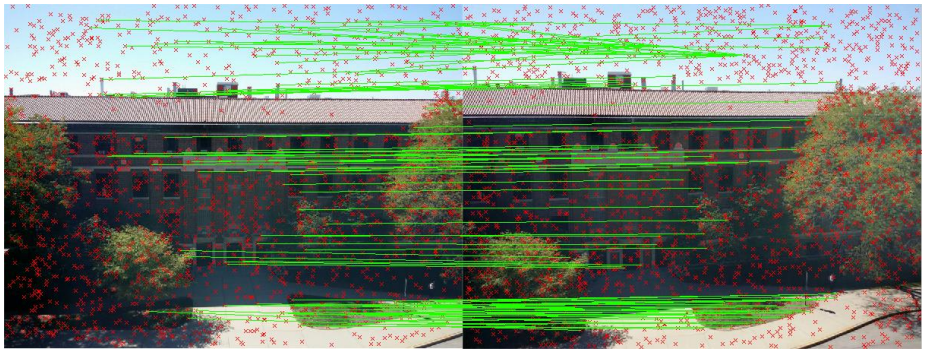

Figure 2: SIFT image with correspondences using NCC

Figure 3: Image correspondences using NCC on Harris with scale 1.2

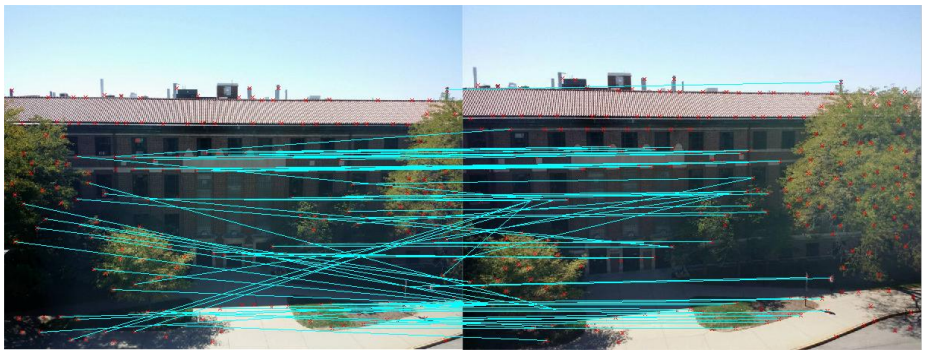

Figure 4: Image correspondences using SSD on Harris with scale 1.2

Figure 5: Image correspondences using NCC on Harris with scale 1.6

Figure 6: Image correspondences using SSD on Harris with scale 1.6

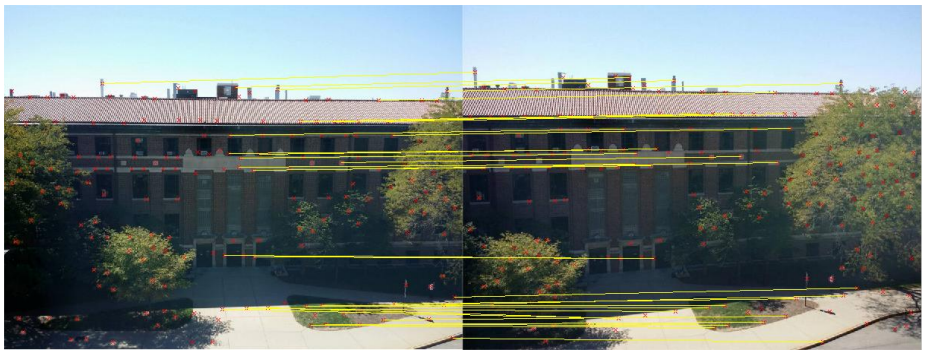

Figure 7: Image correspondences using NCC on Harris with scale 2.2

Figure 8: Image correspondences using SSD on Harris with scale 2.2

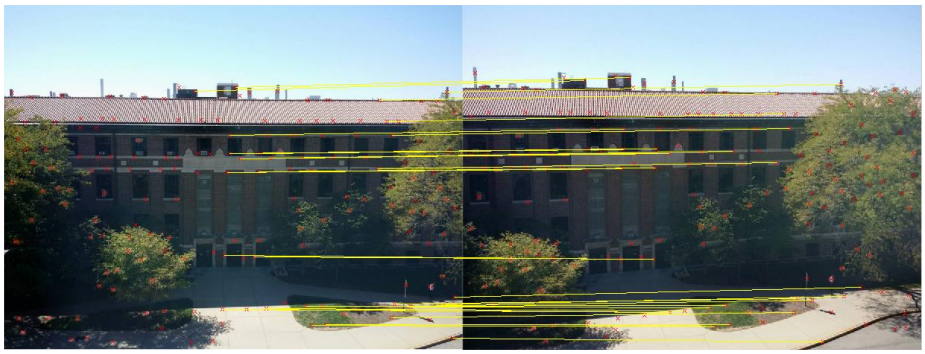

Figure 9: Image correspondences using NCC on Harris with scale 2.6

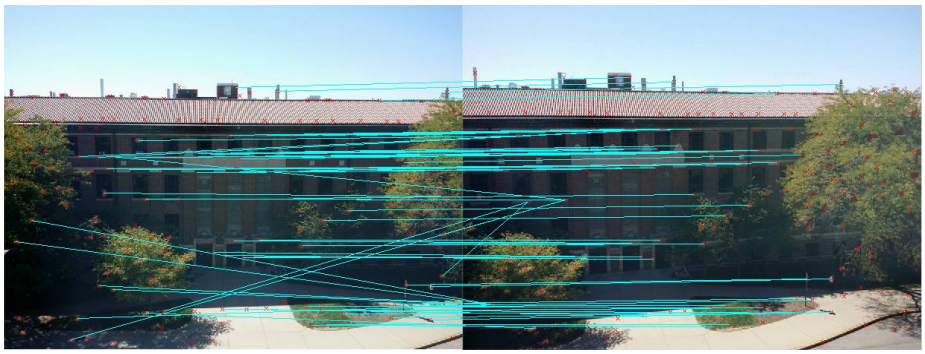

Figure 10: Image correspondences using SSD on Harris with scale 2.6

5.3 Pair 2

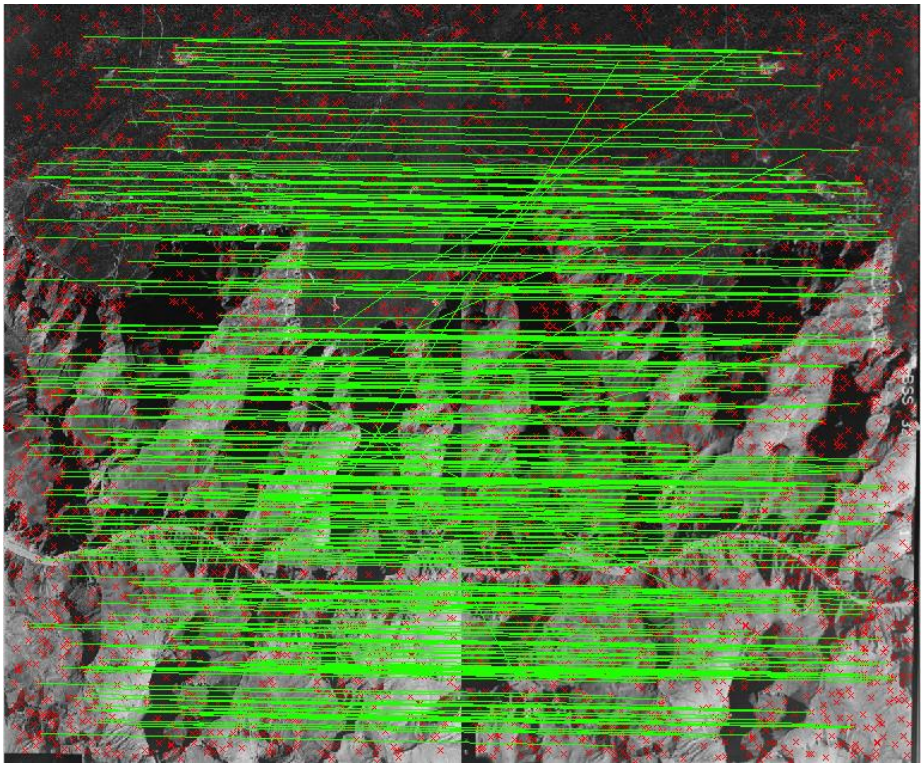

Figure 11: SIFT image with correspondences using Euclidian Distance

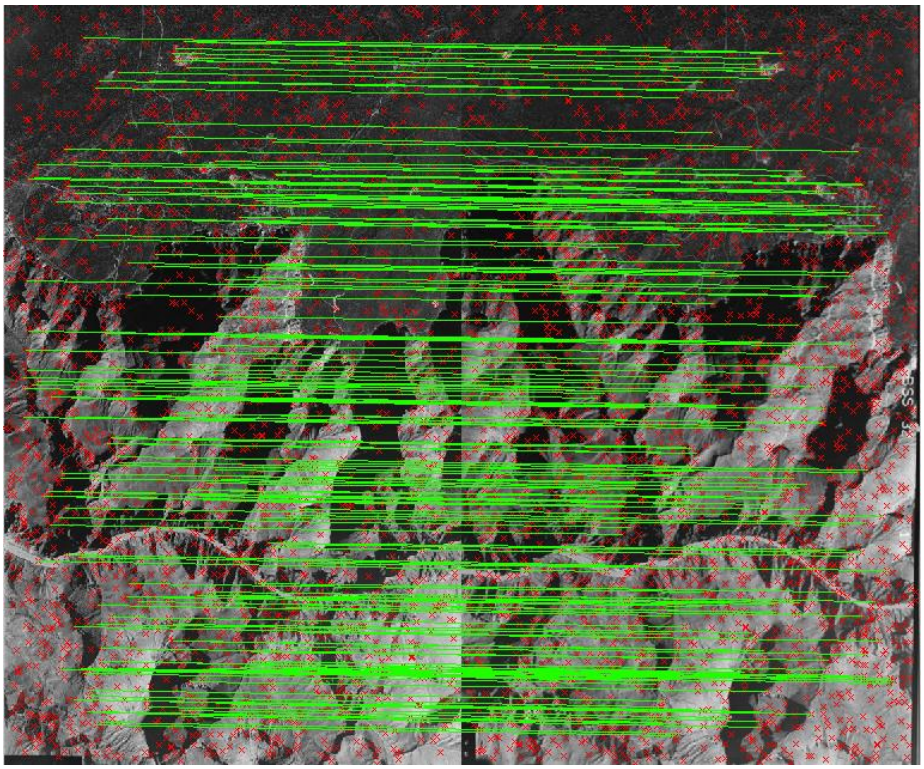

Figure 12: SIFT image with correspondences using NCC

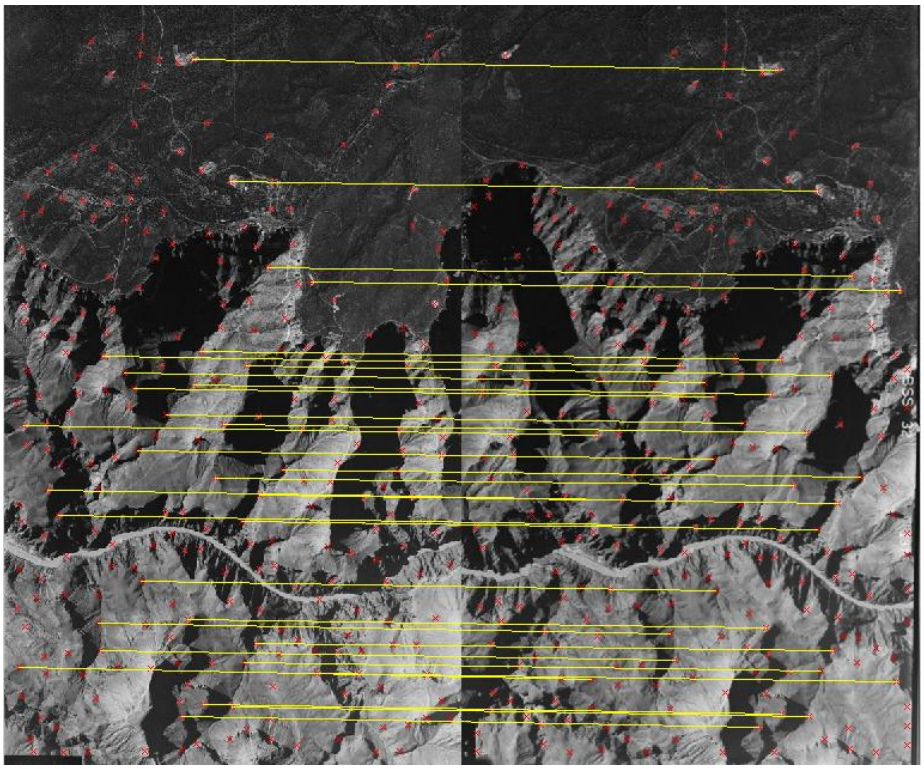

Figure 13: Image correspondences using NCC on Harris with scale 1.6

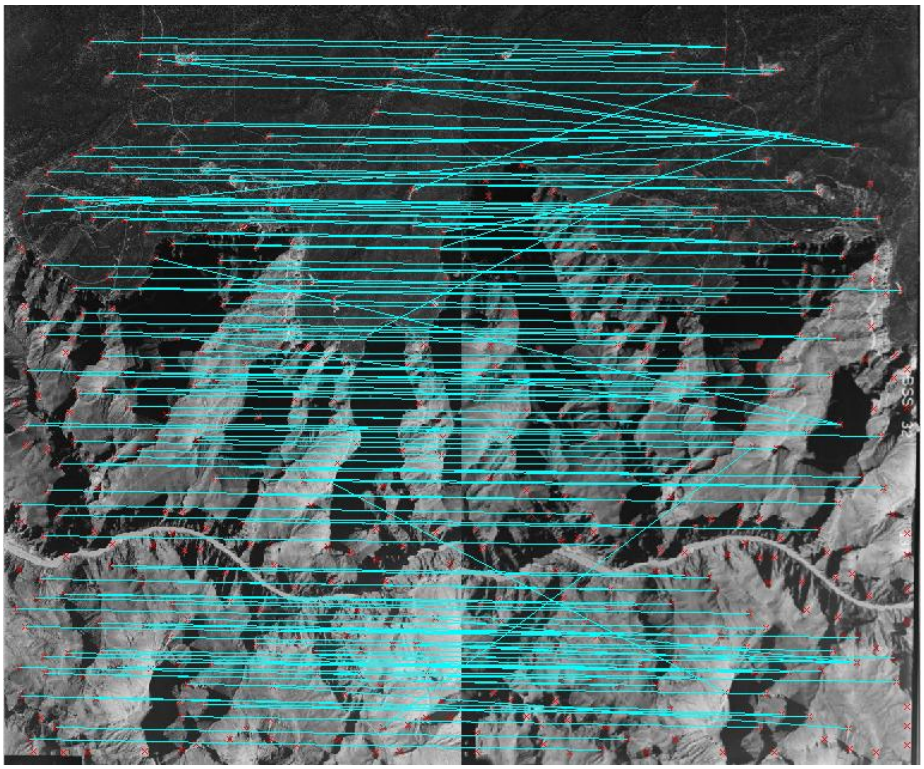

Figure 14: Image correspondences using SSD on Harris with scale 1.6

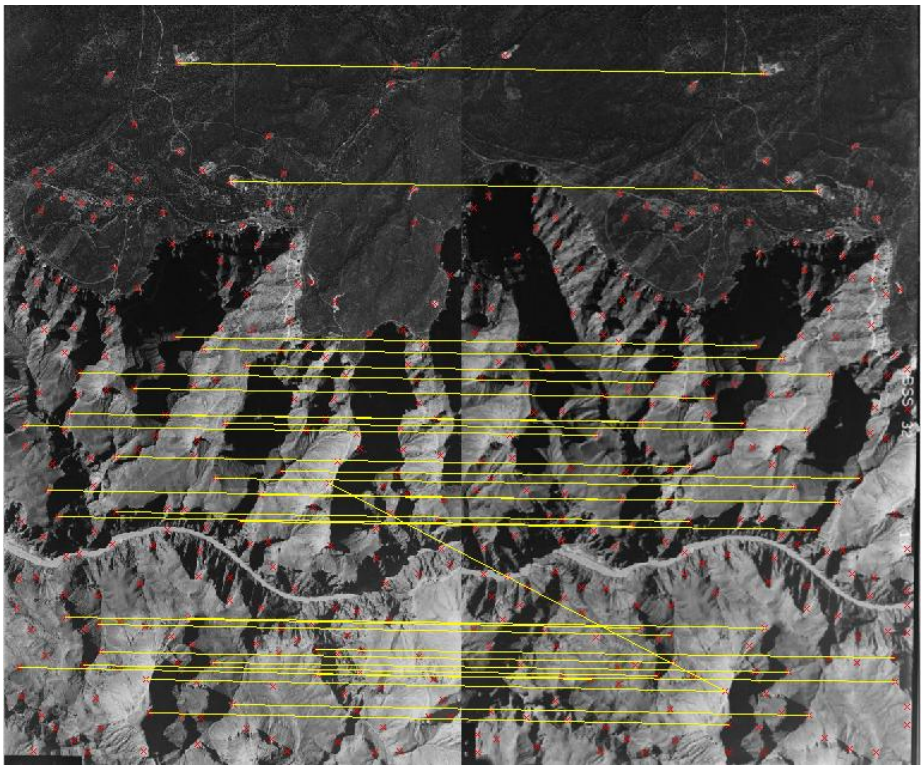

Figure 15: Image correspondences using NCC on Harris with scale 2.2

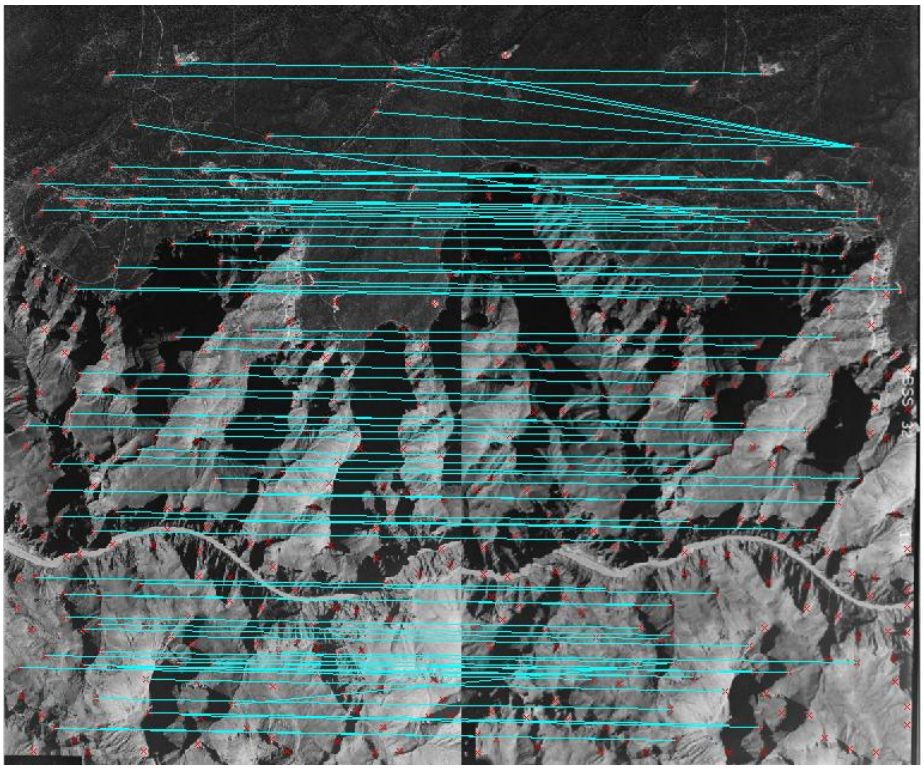

Figure 16: Image correspondences using SSD on Harris with scale 2.2

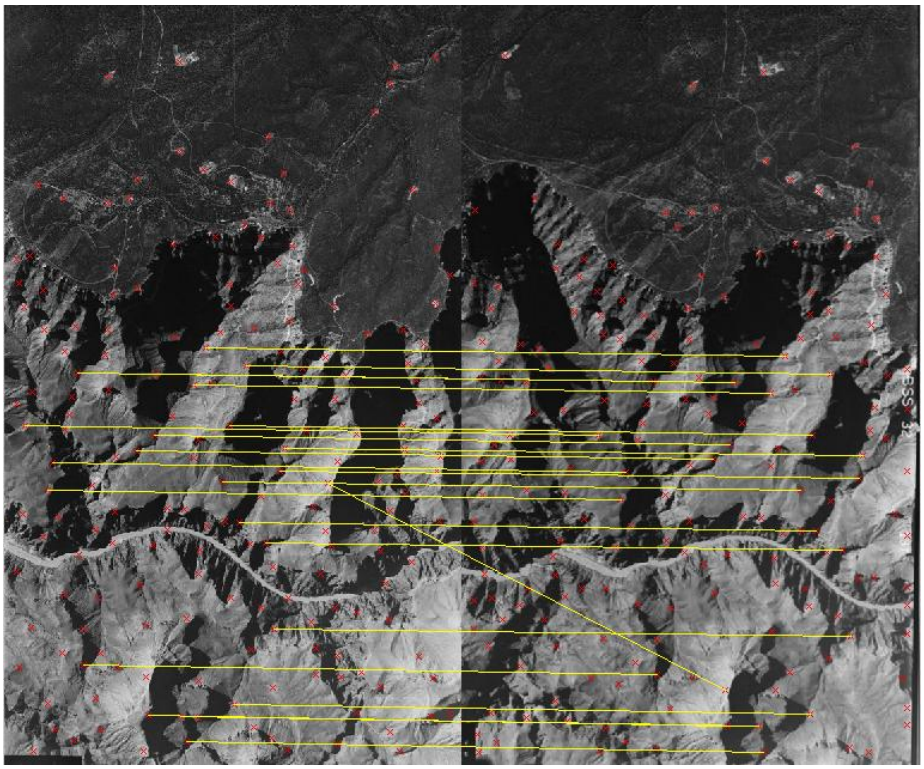

Figure 17: Image correspondences using NCC on Harris with scale 2.6

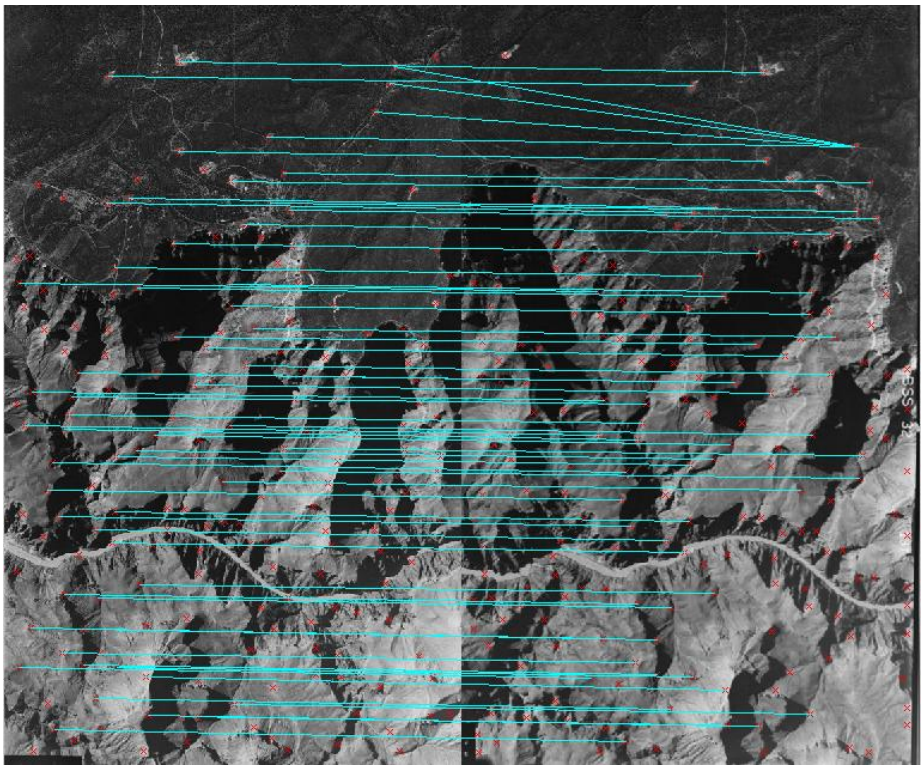

Figure 18: Image correspondences using SSD on Harris with scale 2.6

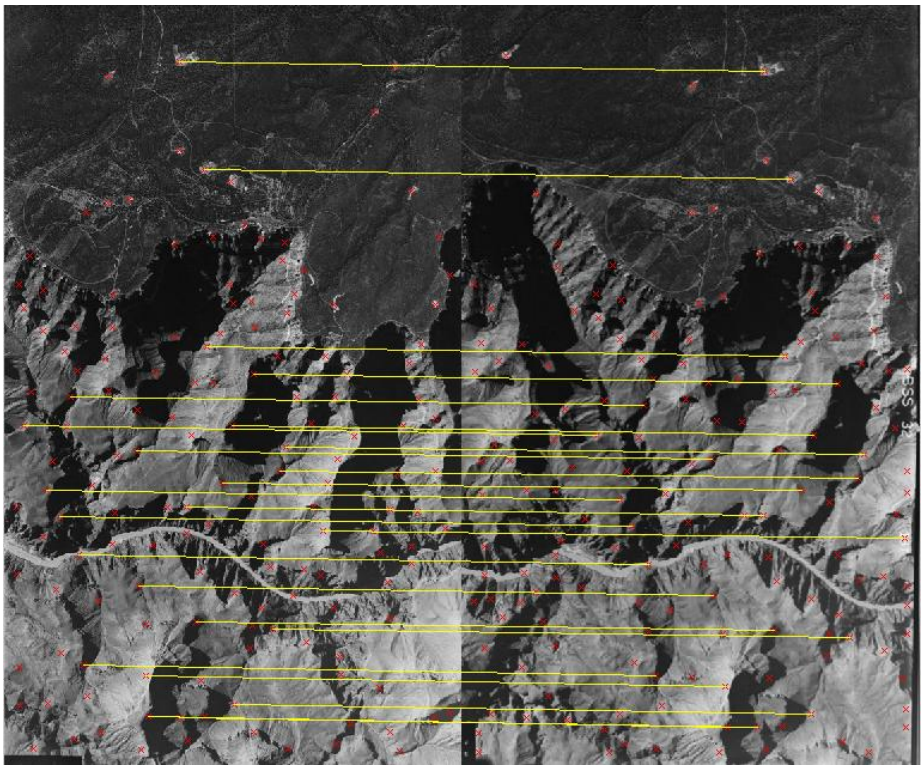

Figure 19: Image correspondences using NCC on Harris with scale 3.2

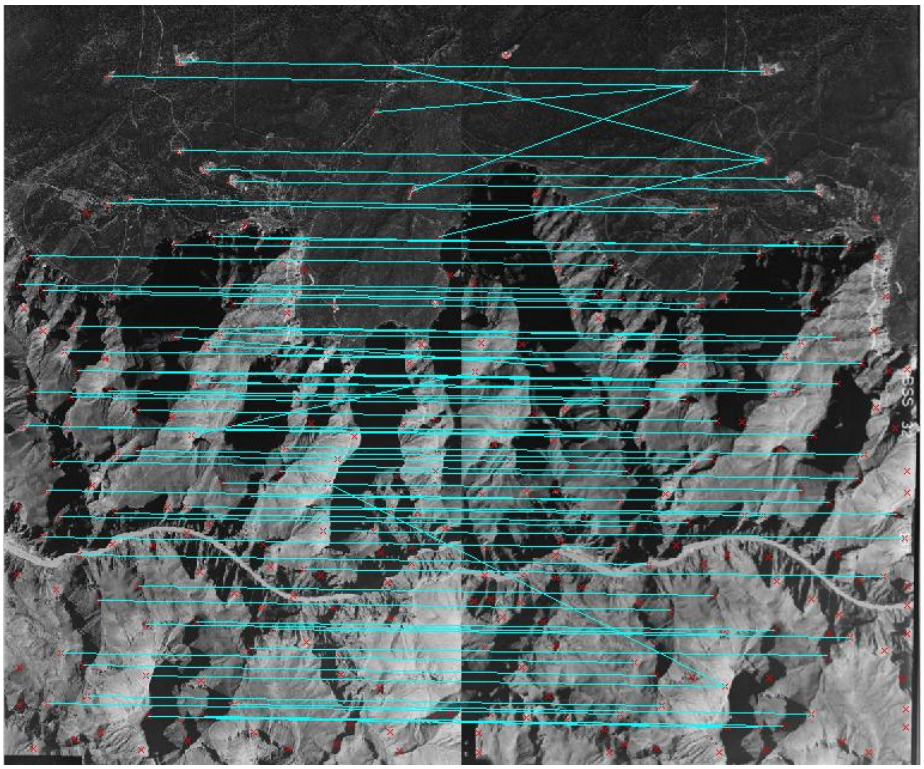

Figure 20: Image correspondences using SSD on Harris with scale 3.2

5.4 Pair 3

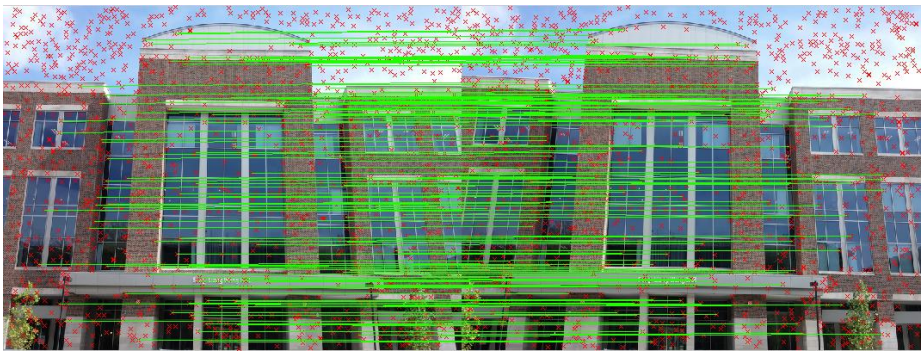

Figure 21: SIFT image with correspondences using Euclidian Distance

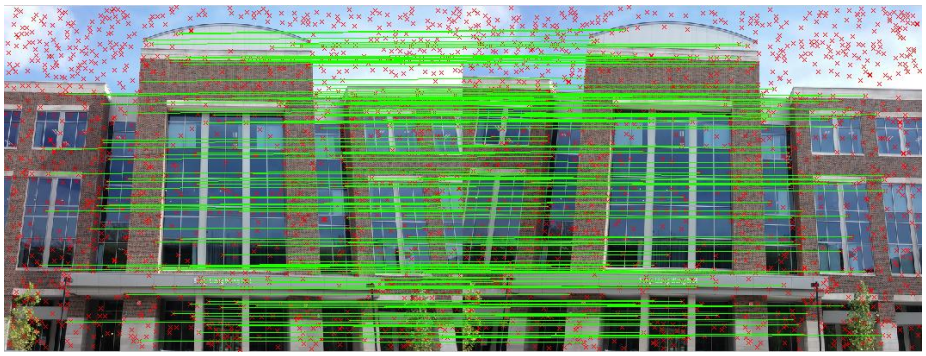

Figure 22: SIFT image with correspondences using NCC

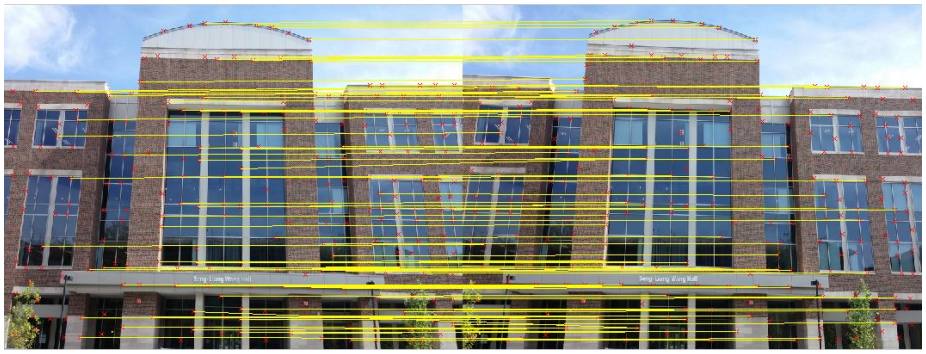

Figure 23: Image correspondences using NCC on Harris with scale 1.2

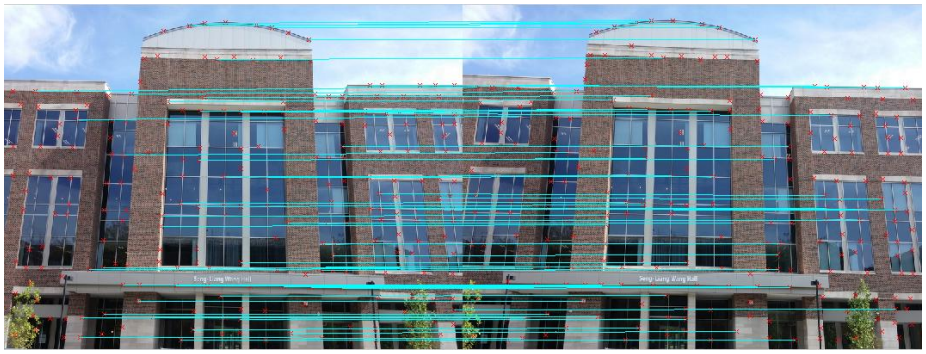

Figure 24: Image correspondences using SSD on Harris with scale 1.2

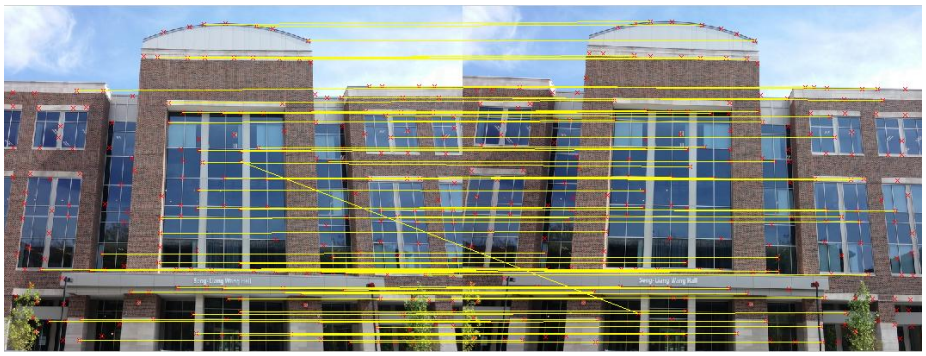

Figure 25: Image correspondences using NCC on Harris with scale 1.6

Figure 26: Image correspondences using SSD on Harris with scale 1.6

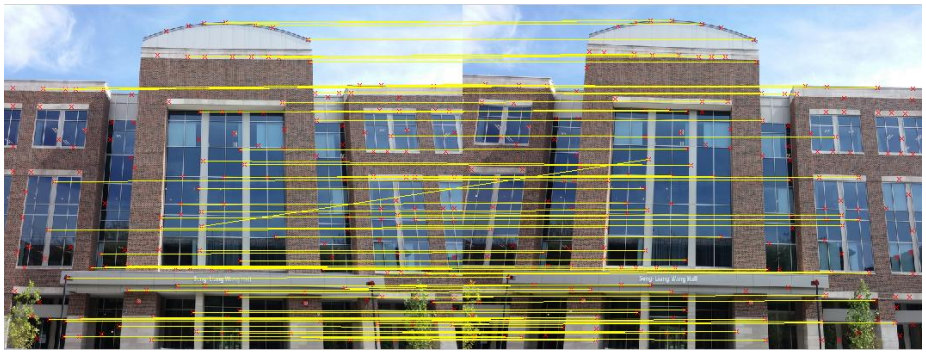

Figure 27: Image correspondences using NCC on Harris with scale 2.2

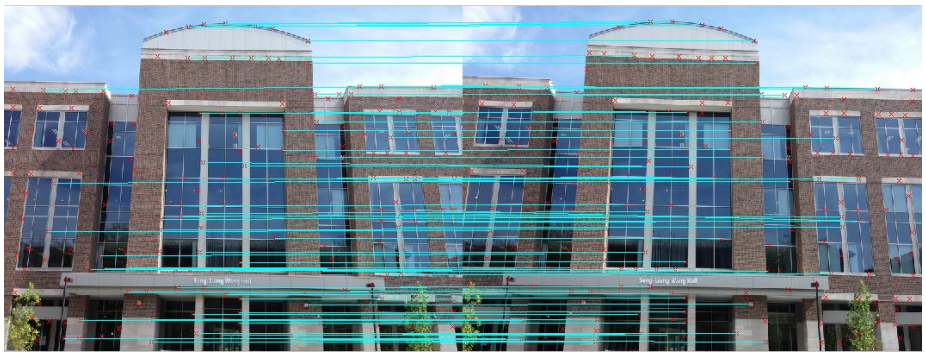

Figure 28: Image correspondences using SSD on Harris with scale 2.2

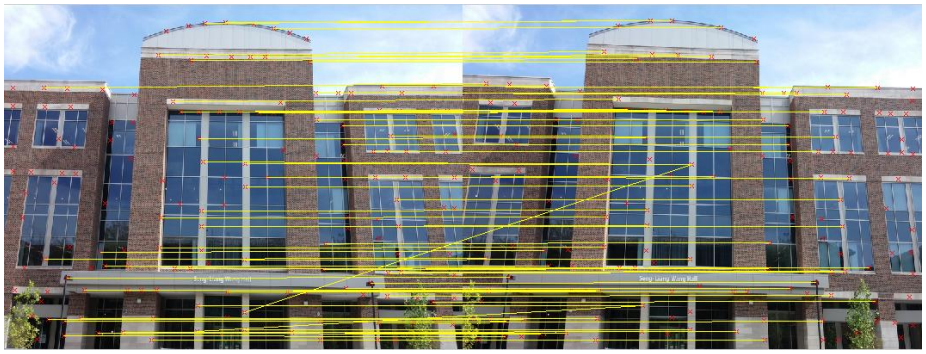

Figure 29: Image correspondences using NCC on Harris with scale 2.6

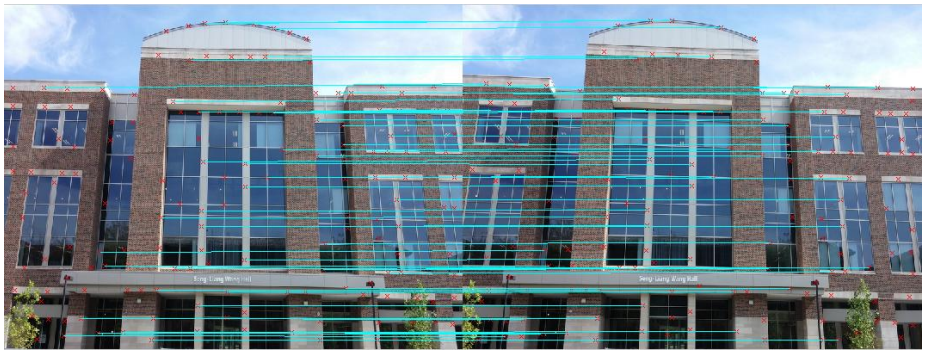

Figure 30: Image correspondences using SSD on Harris with scale 2.6

5.5 Pair 4

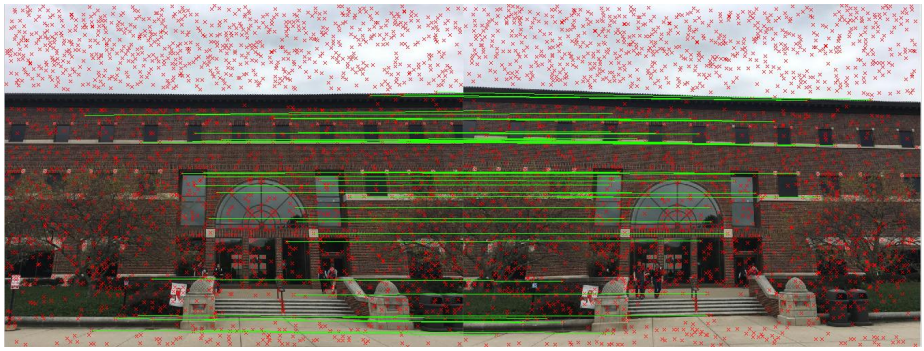

Figure 31: SIFT image with correspondences using Euclidian Distance

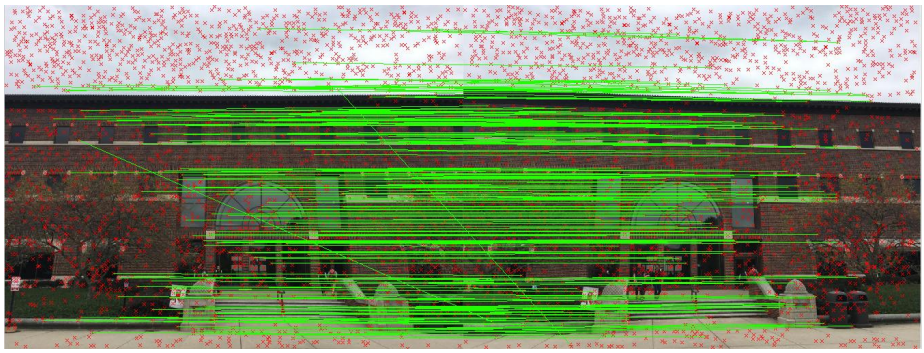

Figure 32: SIFT image with correspondences using NCC

Figure 33: Image correspondences using NCC on Harris with scale 1.2

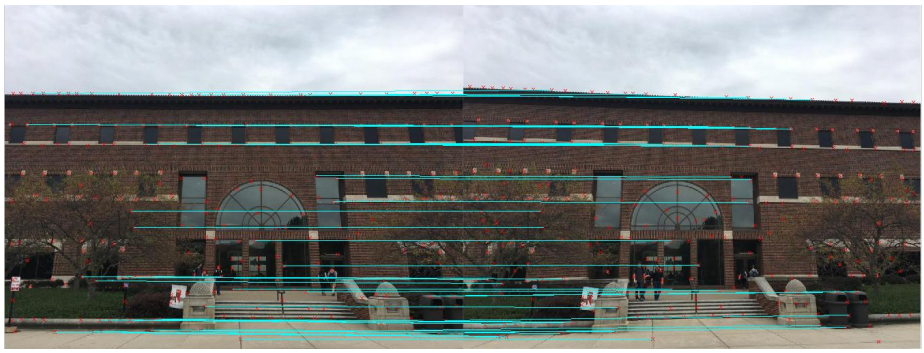

Figure 34: Image correspondences using SSD on Harris with scale 1.2

Figure 35: Image correspondences using NCC on Harris with scale 1.6

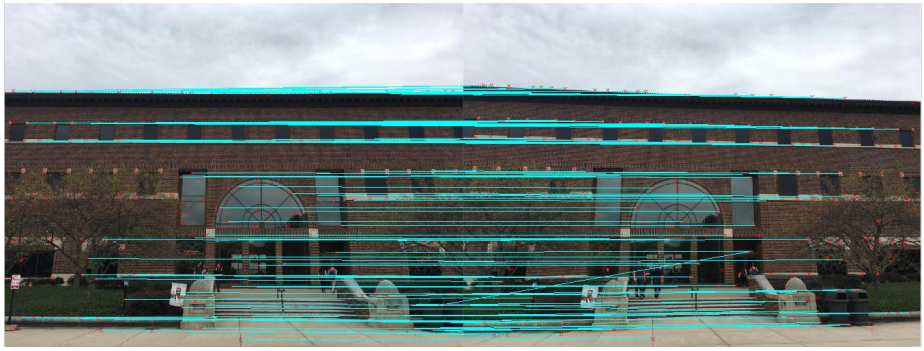

Figure 36: Image correspondences using SSD on Harris with scale 1.6

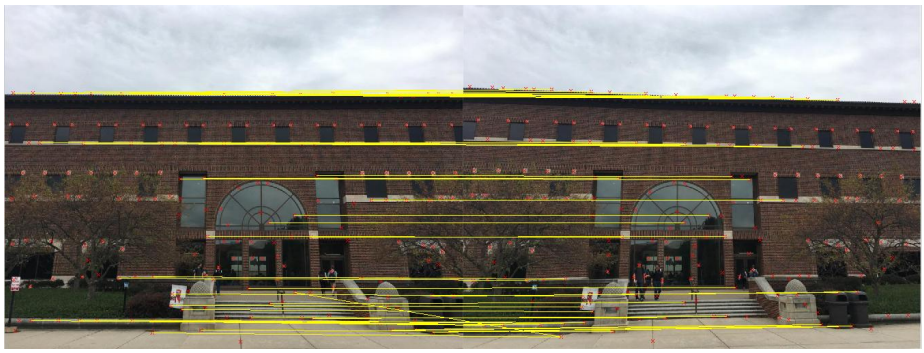

Figure 37: Image correspondences using NCC on Harris with scale 2.2

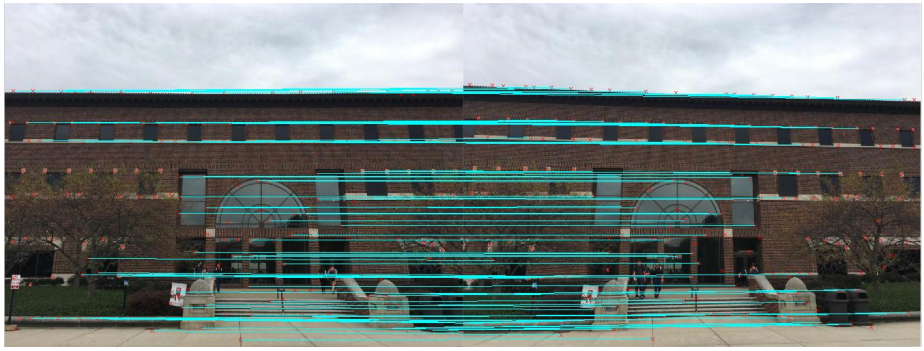

Figure 38: Image correspondences using SSD on Harris with scale 2.2

Figure 39: Image correspondences using NCC on Harris with scale 2.6

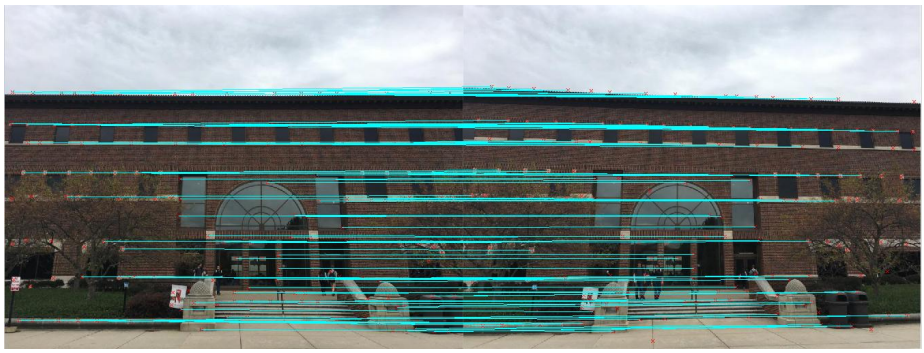

Figure 40: Image correspondences using SSD on Harris with scale 2.6

6 Source Code

6.1 Code for main function

```
clc
clear

cd HW4Pics
cd pair2          % Change according to which images to use for input
I1 = imread('1.jpg');
I2 = imread('2.jpg');
cd ..
cd ..

sigma = [0.8, 1.2, 1.6, 2.2, 2.6, 3.2];

SIFT(I1,I2);

for i = 1:length(sigma)
    Point_Correspondence(I1,I2,sigma(i));
end
```

6.2 Code for SIFT function

```
function [ ] = SIFT( I1,I2 )
%Function to calculate point correspondences between images I1 and I2 using
%SIFT

% Setup SIFT function from VLFeat library
run vlfeat-0.9.20/toolbox/vl_setup

I1_color = double(I1);
```

```
I2_color = double(I2);
I1 = double(rgb2gray(I1));
I2 = double(rgb2gray(I2));

% Find Point Correspondences through SIFT
[f1, d1] = vl_sift(single(I1));
[f2, d2] = vl_sift(single(I2));

P_sift = Euclid(f1,d1,f2,d2);

% Dispaly Point Correspondences for Euclid Metric
I_sift = [I1_color I2_color];
[~,n] = size(I1);
P_sift(:,4) = P_sift(:,4) + n;
figure, imshow(I_sift/255);
title('SIFT_Euclid');
hold on, scatter(f1(1,:),f1(2,:),'rx');
hold on, scatter(f2(1,:)+n,f2(2,:),'rx');
[k,~] = size(P_sift);
for i = 1:k
    hold on, plot([P_sift(i,2) P_sift(i,4)],[P_sift(i,1) P_sift(i,3)],'g');
end

pause(1);

frame = getframe(gca);
Im = frame2im(frame);
imwrite(Im,'SIFT.jpg');

% Dispaly Point Correspondences for NCC Metric
P_sift2 = NCC_SIFT(f1,d1,f2,d2);

I_sift = [I1_color I2_color];
[~,n] = size(I1);
P_sift2(:,4) = P_sift2(:,4) + n;
figure, imshow(I_sift/255);
title('SIFT');
hold on, scatter(f1(1,:),f1(2,:),'rx');
hold on, scatter(f2(1,:)+n,f2(2,:),'rx');
[k,~] = size(P_sift2);
for i = 1:k
    hold on, plot([P_sift2(i,2) P_sift2(i,4)],[P_sift2(i,1) P_sift2(i,3)],'g');
end

pause(1);
```

```
frame = getframe(gca);
Im = frame2im(frame);
imwrite(Im,'SIFT_NCC.jpg');

end
```

6.3 Code for Euclid function

```
function [ P ] = Euclid( f1,d1,f2,d2 )
%function to calculate point correspondences between SIFT features through
%euclidian distances

[~,m] = size(f1);
[~,n] = size(f2);

E = zeros(m,n);

% Euclid Distance between all combinations of interest points
for i = 1:m
    for j = 1:n
        E(i,j) = sqrt(sum((d1(:,i) - d2(:,j)).^2));
    end
end

% Thresholding distances to find point correspondences
[min_E, cor_loc] = min(E,[],2);
E_thresh = 3*min(min_E);
Valid = min_E < E_thresh;

P = zeros(sum(Valid),4);

count = 1;
for i = 1:m
    if (Valid(i) == 1)
        P(count,:) = [f1(2,i) f1(1,i) f2(2,cor_loc(i)) f2(1,cor_loc(i))];
        count = count + 1;
    end
end

end
```

6.4 Code for NCC SIFT function

```
function [ P ] = NCC_SIFT( f1,d1,f2,d2 )
%function to calculate point correspondences between SIFT features through
%NCC
```

```
[p,m] = size(d1);
[~,n] = size(d2);

NCC = zeros(m,n);

M1 = mean(d1);
M2 = mean(d2);
N1 = double(d1) - repmat(M1,[p 1]);
N2 = double(d2) - repmat(M2,[p 1]);

count = 0;

% NCC between all combinations of interest points
for i = 1:m
    for j = 1:n
        count = count + 1;
        T1 = N1(:,i);
        T2 = N2(:,j);
        NCC(i,j) = sum(T1.*T2)/sqrt(sum(T1.^2)*sum(T2.^2));
        if(mod(count,50000)==0)
            disp(count);
            pause(0.1);
        end
    end
end

% Thresholding distances to find point correspondences
[max_NCC, cor_loc] = max(NCC,[],2);
NCC_thresh = 0.99;
Valid = max_NCC > NCC_thresh;

P = zeros(sum(Valid),4);

count = 1;
for i = 1:m
    if (Valid(i) == 1)
        P(count,:) = [f1(2,i) f1(1,i) f2(2,cor_loc(i)) f2(1,cor_loc(i))];
        count = count + 1;
    end
end

end
```

6.5 Code for point correspondence function

```
function [ ] = Point_Correspondence( I1,I2,sigma )
%Function to calculate point Correspondences between images I1 and I2 using
%the scale parameter sigma for the Haar Filters in Harris Corner Detector

% Convert images to grayscale and doubles for computation
I1_color = double(I1);
I2_color = double(I2);
I1 = double(rgb2gray(I1));
I2 = double(rgb2gray(I2));

% Form Haar Filters
[X,Y] = Haar_Filters(sigma);

% Form the x and y derivative images
[Dx1, Dy1] = Derivatives(I1,X,Y);
[Dx2, Dy2] = Derivatives(I2,X,Y);

% Form a binary image with 1 at pixel signifying corner point
B1 = Corners(Dx1,Dy1,sigma);
[cx1,cy1] = find(B1 == 1);

B2 = Corners(Dx2,Dy2,sigma);
[cx2,cy2] = find(B2 == 1);

% Find Point Correspondences through SSD
P = SSD(cx1,cy1,cx2,cy2,I1,I2);
I = [I1_color I2_color];
[~,n] = size(I1);
P(:,4) = P(:,4) + n;
figure, imshow(I/255);
title(['SSD with Sigma = ' num2str(sigma)]);
hold on, scatter(cy1,cx1,'rx');
hold on, scatter(cy2+n,cx2,'rx');
[k,~] = size(P);
for i = 1:k
    hold on, plot([P(i,2) P(i,4)],[P(i,1) P(i,3)],'c');
end

pause(1);

frame = getframe(gca);
Im = frame2im(frame);
imwrite(Im,['SSD_sig_' num2str(sigma) '.jpg']);

% Find Point Correspondences through NCC
P_ncc = NCC(cx1,cy1,cx2,cy2,I1,I2);
```

```
I_ncc = [I1_color I2_color];
[~,n] = size(I1);
P_ncc(:,4) = P_ncc(:,4) + n;
figure, imshow(I_ncc/255);
title(['NCC with Sigma = ' num2str(sigma)]);
hold on, scatter(cy1,cx1,'rx');
hold on, scatter(cy2+n,cx2,'rx');
[k,~] = size(P_ncc);
for i = 1:k
    hold on, plot([P_ncc(i,2) P_ncc(i,4)],[P_ncc(i,1) P_ncc(i,3)],'y');
end

pause(1);

frame = getframe(gca);
Im = frame2im(frame);
imwrite(Im,['NCC_sig_' num2str(sigma) '.jpg']);

end
```

6.6 Code for Haar Filters function

```
function [ X,Y ] = Haar_Filters( sigma )
%Function to generate the Haar Filter kernels for the specified scale

% Filter size is smallest even integer greater than 4*sigma
N = ceil(4*sigma);
if (mod(N,2) ~= 0)
    N = N + 1;
end

% Forming Haar Filers
X = ones(N,N);
Y = -1.*X;

X(:,1:(N/2)) = -1.*X(:,1:(N/2));
Y(1:(N/2),:) = -1.*Y(1:(N/2),:);

end
```

6.7 Code for Derivatives function

```
function [ Dx,Dy ] = Derivatives( I,X,Y )
%Function to calculate the image derivatives in x and y directions given
%the derivative filters X and Y for the image I
```

```
Dx1 = conv2(I,X,'same');
Dy1 = conv2(I,Y,'same');

Dx = (Dx1 - min(Dx1(:)))./(max(Dx1(:)) - min(Dx1(:)));
Dy = (Dy1 - min(Dy1(:)))./(max(Dy1(:)) - min(Dy1(:)));

end
```

6.8 Code for Corners function

```
function [ I ] = Corners( X, Y, sigma )
%Function to output a 1 in pixel coordinates where corner is detected given
%a sigma and the x and y derivates.

[m,n] = size(X);
R = zeros(m,n);
I = R;

% Finding neighborhood size for forming matrix M
% Want an odd size neighborhood for to have unique central pixel
N = ceil(5*sigma);
if (mod(N,2) == 0)
    N = N + 1;
end

pad = (N-1)/2;
X = padarray(X,[pad pad]);
Y = padarray(Y,[pad pad]);

C = zeros(2,2);

% Forming matrix C at each pixel and calculating ratio r
count = 0;
for i = 1:m
    for j = 1:n
        count = count + 1;
        if(mod(count,50000) == 0)
            disp(count);
        end

        patchX = X((i+pad)-pad:(i+pad)+pad, (j+pad)-pad:(j+pad)+pad);
        patchY = Y((i+pad)-pad:(i+pad)+pad, (j+pad)-pad:(j+pad)+pad);

        C(1,1) = sum(patchX(:).^2);
        C(2,2) = sum(patchY(:).^2);
        C(1,2) = sum(patchX(:).*patchY(:));
```

```
        C(2,1) = C(1,2);

        r = rank(C);
        if (r == 2)
            R(i,j) = det(C)/(trace(C)^2);
        end
    end
end

% Removing detected points at the corners that can occur due to boundary
% conditions being 0 outside the boundary
R(1:15,:) = 0;
R(m-14:m,:) = 0;
R(:,1:15) = 0;
R(:,n-14:n) = 0;

% Non maximum suppression
SN = 31;      % Window size to consider for non maximum suppression
pad_SN = (SN-1)/2;
R_SN = padarray(R,[pad_SN pad_SN]);
Thresh = mean(R(:));

count = 0;
for i = 1:m
    for j = 1:n
        count = count + 1;
        if(mod(count,50000) == 0)
            disp(count);
        end

        if (R(i,j) > 0)
            patch = R_SN((i+pad_SN)-pad_SN:(i+pad_SN)+pad_SN, ...
                (j+pad_SN)-pad_SN:(j+pad_SN)+pad_SN);

            % Finding position of significant corner point
            if (R(i,j) == max(patch(:)) && R(i,j) > Thresh)
                I(i,j) = 1;
            end
        end
    end
end

end
```

6.9 Code for NCC function

```
function [ P ] = NCC( x1,y1,x2,y2,I1,I2 )
%Function to calculate point correspondences between sets of interest
%points for I1 and I2

N = 31;      %Window size for calculating SSD
pad = (N-1)/2;

% Zero padding boundaries
I1 = padarray(I1,[pad pad]);
I2 = padarray(I2,[pad pad]);

A = length(x1);
B = length(x2);

NCC = zeros(A,B);

% Finding NCC between all combinations of interest points
for i = 1:A
    for j = 1:B
        patch1 = I1((x1(i)+pad)-pad:(x1(i)+pad)+pad, ...
            (y1(i)+pad)-pad:(y1(i)+pad)+pad);
        patch2 = I2((x2(j)+pad)-pad:(x2(j)+pad)+pad, ...
            (y2(j)+pad)-pad:(y2(j)+pad)+pad);

        m1 = mean(patch1(:));
        m2 = mean(patch2(:));

        T1 = patch1 - m1;
        T2 = patch2 - m2;

        NCC(i,j) = sum(sum(T1.*T2))/sqrt(sum(T1(:).^2)*sum(T2(:).^2));
    end
end

% Thresholding NCC to find point correspondences
[max_NCC, cor_loc] = max(NCC,[],2);
NCC_thresh = 0.95;
Valid = max_NCC > NCC_thresh;

P = zeros(sum(Valid),4);

count = 1;
for i = 1:A
    if (Valid(i) == 1)
        P(count,:) = [x1(i) y1(i) x2(cor_loc(i)) y2(cor_loc(i))];
        count = count + 1;
```

```
    end
end

end
```

6.10 Code for SSD function

```
function [ P ] = SSD( x1,y1,x2,y2,I1,I2 )
%Function to calculate point correspondences between sets of interest
%points for I1 and I2

N = 31;     %Window size for calculating SSD
pad = (N-1)/2;

% Zero padding boundaries
I1 = padarray(I1,[pad pad]);
I2 = padarray(I2,[pad pad]);

A = length(x1);
B = length(x2);

SSD = zeros(A,B);

% Find SSD between all combinations of interest points
for i = 1:A
    for j = 1:B
        patch1 = I1((x1(i)+pad)-pad:(x1(i)+pad)+pad, ...
            (y1(i)+pad)-pad:(y1(i)+pad)+pad);
        patch2 = I2((x2(j)+pad)-pad:(x2(j)+pad)+pad, ...
            (y2(j)+pad)-pad:(y2(j)+pad)+pad);

        SSD(i,j) = sum(sum((patch1 - patch2).^2)) ;
    end
end

% Thresholding SSD to find correspondences
[min_SSD, cor_loc] = min(SSD,[],2);
SSD_thresh = 10*min(min_SSD);
Valid = min_SSD < SSD_thresh;

P = zeros(sum(Valid),4);

count = 1;
for i = 1:A
    if (Valid(i) == 1)
        P(count,:) = [x1(i) y1(i) x2(cor_loc(i)) y2(cor_loc(i))];
```

```
            count = count + 1;
        end
    end

    end
```