Anthony Hearst
ahearst@purdue.edu
October 4th, 2016
ECE 661

Homework 4

Introduction

A common task in computer vision is establishing correspondences between interest points in two images of the same scene. The first step in this task is identifying distinct interest points in both images that may be invariant to changes in scale and rotation. To achieve this task, we explore the use of the Harris corner detector and the Scale Invariant Feature Transform (SIFT). The next step is matching the interest points in the two images. To achieve this task, we explore the use of the Sum of Squared Differences (SSD) and the Normalized Cross Correlation (NCC) of image gray levels in the vicinity of the interest points as metrics of similarity. In the case of SIFT, we use the Euclidean distance between formal descriptors of the interest points as the metric of similarity. Correspondences between the images may be established by identifying interest points with high similarity. First, we provide theoretical background on Harris Corner and SIFT key point detection and matching. Then we compare the performance of these algorithms using four pairs of test images.

Theoretical Background

Harris Corner Detection

Harris corners are points in the image at which strong variations in gray levels are occurring in two orthogonal directions. The two orthogonal directions of maximum variation define the orientation of the corner. This makes Harris Corners invariant to image rotation. Harris corners may also be detected at multiple scales using properly-sized Haar filters. In this study, Harris Corner detection was accomplished using the following algorithm:

1. Define an image scale σ (e.g. $\sigma = \sqrt{2}, 2\sqrt{2}, 3\sqrt{2},$ or $4\sqrt{2}$). The corresponding Haar filters are of size $N \times N$ such that N is the smallest even integer greater than 4σ. For example, when $\sigma = \sqrt{2}$ the corresponding Haar filters in the x and y direction are given below.

$$h_x = \begin{bmatrix} -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & 1 \end{bmatrix} \quad h_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

2. Convert the image I to grayscale and compute the x and y gradients of all of the pixels in the image at the desired scale σ by convolving the image with the appropriate Haar filters h_x and h_y.

$$G_x = I * h_x \qquad G_y = I * h_y$$

3. Compute the squares and products of the gradients at every pixel.

$$G_{x^2} = G_x^2 \qquad G_{y^2} = G_y^2 \qquad G_{xy} = G_x G_y$$

4. Define a window that you will use to compute Harris corner detector responses at each image pixel. The window may be sized as $5\sigma \times 5\sigma$ rounded to the nearest odd integer.

5. Center the search window on every image pixel (x, y). Within each search window, compute the sums of the corresponding G_{x^2}, G_{y^2}, and G_{xy} values and define the matrix $H(x, y)$.

$$S_{x^2} = \sum G_{x^2} \qquad S_{y^2} = \sum G_{y^2} \qquad S_{xy} = \sum G_{xy}$$

$$H(x, y) = \begin{bmatrix} S_{x^2} & S_{xy} \\ S_{xy} & S_{y^2} \end{bmatrix}$$

6. Compute the Harris corner detector response (R) for each image pixel based on the eigenvalues of the matrix $H(x, y)$ (these eigenvalues correspond to the two orthogonal directions of maximum variation about the pixel) using experimentally-determined parameter $k = 0.04$.

$$R = Det(H) - k\big(Trace(H)\big)^2$$

$$Det(H) = \lambda_1 \lambda_2 \qquad Trace(H) = \lambda_1 + \lambda_2$$

7. Find Harris Corners by thresholding out pixels with small or negative R values and identifying the local maxima among the remaining pixels. In this study, local maxima were found using a search window of size 29x29 pixels. These maxima are likely to be strong 'corners' in the sense that they exhibit a relatively large amount of variation in gray levels about two orthogonal axes.

Matching Harris Corners

Harris Corners were matched using the Sum of Squared Differences (SSD) and the Normalized Cross Correlation (NCC) as metrics of similarity. These metrics were calculated over $N \times N$ windows of pixels centered on each Harris Corner where N was set equal to 21. For a pair of Harris corners in images I_1 and I_2, the SSD and NCC are defined as follows:

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

$$NCC = \frac{\sum_i^N \sum_j^N (I_1(i,j) - \mu_1)(I_2(i,j) - \mu_2)}{\sum_i^N \sum_j^N (I_1(i,j) - \mu_1)^2 \sum_i^N \sum_j^N (I_2(i,j) - \mu_2)^2}$$

Where μ_1 and μ_2 are the means of the gray levels of the pixels in the $N \times N$ windows in image I_1 and I_2, respectively. In theory, the NCC should allow for slightly better matching than the SSD because it can account for small differences in illumination between the images by normalizing with respect to their mean gray levels μ_1 and μ_2.

We note that although Harris Corners are invariant to rotation and detected at multiple scales, the SSD and NCC are computed in a manner such that they do not capture any changes in the scale or rotation of the interest points that may be present. Therefore, poor matching is to be expected for both NCC and SSD in images that exhibit large differences in scale and rotation. But, for image pairs that exhibit minimal differences in scale and rotation, it may still be reasonable to postulate that the lower the SSD or the higher the NCC, the more likely the interest points in the two images are a match.

Following this reasoning, it might make sense to compute the SSD and NCC for all possible combinations of Harris Corners in the two images and then apply a threshold to these values to identify matching combinations. However, if we plot these SSD and NCC values for a pair of images, it becomes clear that there is no obvious way of identifying such a threshold (or even a ratio of successive SSD or NCC values) that indicates a transition from matches to non-matches (Figure 1). This problem persists even after removing obvious non-matches such as points that were already matched with other points or point combinations with very high SSD or low NCC. It might be possible to improve the situation by considering the image coordinates of the Harris Corners, but this would require a great deal of meaningless, ad-hoc programming that would not

provide any valuable insight into the quality of correspondences that may be obtained by finding Harris Corners and relying on the SSD and NCC to match them.

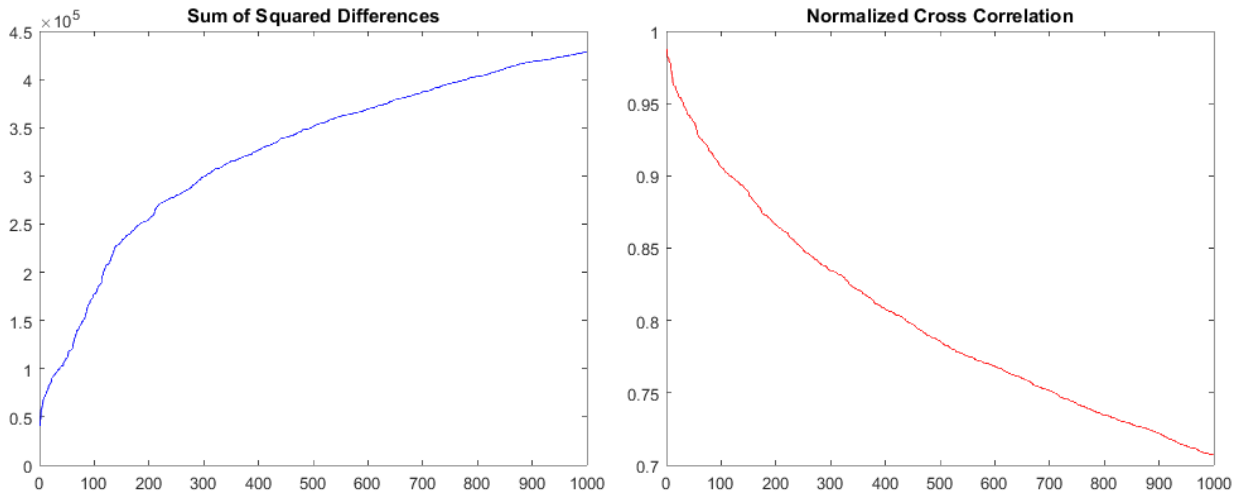


Figure 1. The best SSD and NCC values for Harris Corners detected at scale 1 in the first pair of test images. Matches are among these point combinations, but it is impossible to identify them based on the SSD and NCC alone.

Therefore, in this implementation, the 200 combinations with the highest SSD and NCC values were taken to be 'matches' and the quality of the matches was evaluated and compared to the top 200 'matches' obtained from SIFT.

SIFT Key Point Detection

The SIFT algorithm was described in detail in lecture, so it will only be described briefly here. SIFT key points are defined as the local extrema of the Difference of Gaussian (DoG) pyramid of an image. The DoG of an image (D) is defined as follows:

$$D(x, y, \sigma) = ff(x, y, \sigma_1) - ff(x, y, \sigma_2)$$

Where $ff(x, y, \sigma)$ is the image convolved with a Gaussian filter of standard deviation σ and σ_1 and σ_2 are two closely spaced values of σ (the DoG serves as an approximation for the Laplacian of the image computed at scale σ). The DoG pyramid extends through multiple octaves of scale space ($i\sigma$ where $i = 1, 2,$ and 4) and multiple DoG's are calculated across the scale space within each octave. Key points are identified as the local extrema of D in the space defined by $\mathrm{X} = (x, y, \sigma)$. Since D is initially computed at discrete sampling intervals, additional work is needed to localize the extrema with sub-pixel accuracy. This is accomplished by using a Taylor series

expansion to form a continuous approximation for D about the approximate location of the extrema $\mathrm{X}_0 = (x_0, y_0, \sigma_0)$ based on its second-order derivative:

$$D(x, y, \sigma) = D(\mathrm{X}_0) + J^T(\mathrm{X}_0)\mathrm{X} + \frac{1}{2}x^T H(\mathrm{X}_0)\mathrm{X}$$

Where X is an incremental deviation from X_0, J is the Jacobian, and H is the Hessian:

$$J(\mathrm{X}_0) = \left(\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial \sigma}\right)^T_{\mathrm{X}_0}$$

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

Following this local approximation for D to a refined location for the extrema allows SIFT key points to be localized with sub-pixel precision. Low-contrast key points are ignored by requiring that $|D(\mathrm{X})| > 0.03$ and key points near the edge of the image are also excluded. In this study, SIFT key point detection was accomplished using the implementation in the VLFeat library with default parameter settings (http://www.vlfeat.org/).

Matching SIFT Key Points

Once key points are localized in both images, the next step is to match them across the images. In the SIFT algorithm this is accomplished by describing each key point uniquely and in a manner that is invariant to scale and rotation using a 128-dimensional descriptor. Key points are matched based on the Euclidean distance between their descriptors. A small Euclidean distance between two descriptors indicates a match.

The descriptor for a given key point is computed based on a window of pixels in the vicinity of the key point in the image smoothed at the scale in which the key point was detected $ff(x, y, \sigma)$ (this makes the descriptor invariant to scale). First, the orientation of the key point is determined by analyzing the magnitudes and orientations of gray levels within the vicinity of the key point:

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

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

The orientations θ are weighted by their respective magnitudes m and sorted into a histogram of 36 bins spanning 360°. The bins with the greatest weight determine the dominant orientation of the key point. Finally, the descriptor is computed using a 16×16 window of pixels surrounding the key point. The 16×16 window of pixels is sub-divided into sixteen 4×4 blocks of pixels. Each 4×4 block contains 16 pixels. For each 4×4 block the magnitudes and orientations of the 16 pixels are calculated relative to the dominant orientation of the key point and binned into an 8-bin histogram (this makes the descriptor invariant to rotation). Thus, we get one 8-bin histogram for each 4×4 block of pixels, resulting in a total of sixteen 8-bin histograms. These histograms are stringed together to form a 128-dimensional descriptor for each key point. The descriptor is normalized to unit length to make it invariant to small changes in illumination. The Euclidean distance between key point descriptors (labeled a and b) is calculated as follows:

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

The smaller the Euclidean distance between two key points, the more likely they are a match. These descriptors offer a highly detailed and unique description of each key point that is invariant to scale and rotation. This makes it possible to identify matches between the images by computing distances between all possible combinations of key points and applying a threshold.

Validation of Matches

A crude, but fast and sufficiently accurate method was devised for validating matches detected in this image set. This involved comparing the image coordinates of the detected matches to the image coordinates of manually-measured matches. Specifically, the images were appended side-by-side and lines were drawn between the detected matches. The angles and lengths of the lines were compared to the angle and length of a line between the manually measured matches. If the angle differed by more than 5 degrees or the length differed by more than 5% from the reference line, then the detected match was considered erroneous. This method worked well for this image set because these images exhibited minimal differences in rotation and scale and also minimal projective distortion.

Experimental Results

Image Set

Four pairs of images were used for this study (Figures 2, 3, 4, and 5).

Figure 2. Image pair 1.

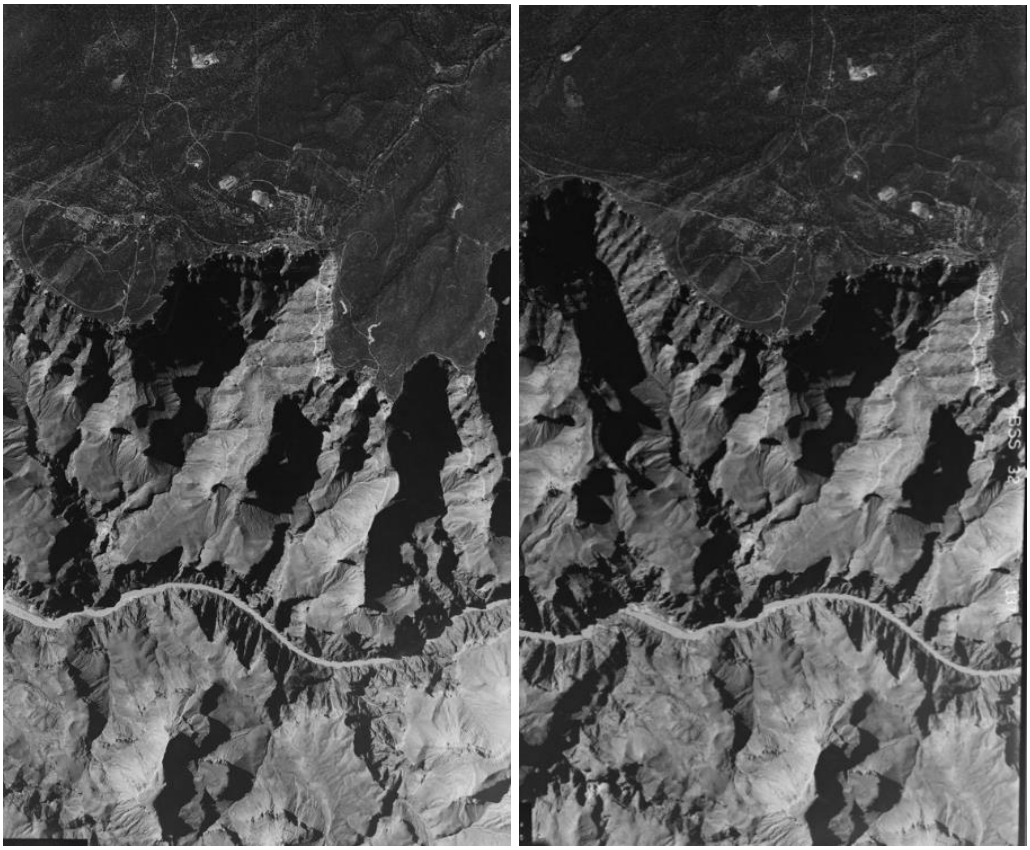


Figure 3. Image pair 2.

Figure 4. Image pair 3.

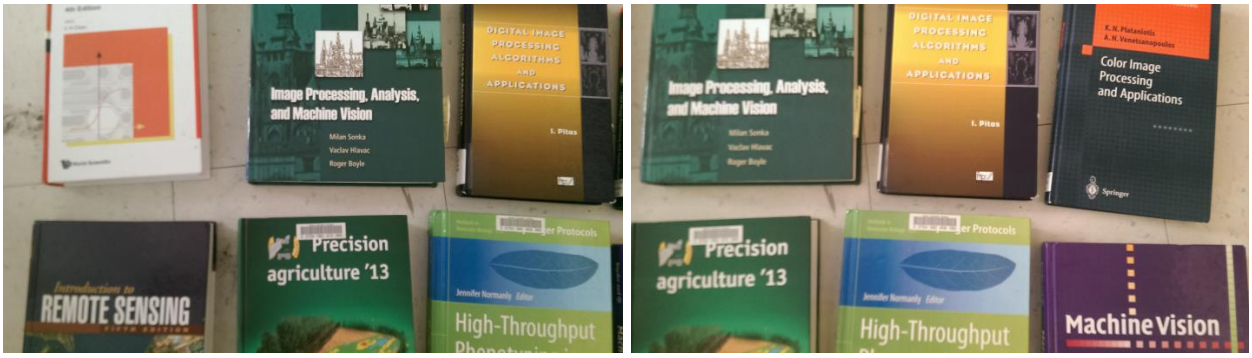


Figure 5. Image pair 4.

Tables 1 and 2 summarize the parameters used for interest point detection and matching.

Table 1. Parameters for Harris Corner detection and matching.

Parameter	Value
Scale (σ) (unitless)	$\sqrt{2}, 2\sqrt{2}, 3\sqrt{2}$, and $4\sqrt{2}$
Haar filter size (pixels)	Smallest even integer greater than 4σ
Size of window for computing Harris corner detector responses (pixels)	$5\sigma \times 5\sigma$ rounded to nearest odd integer
Experimentally-determined parameter for computing detector responses (k)	0.04
Size of window for finding local maxima of detector responses in the image (pixels)	29×29
Size of window for computing SSD and NCC (pixels)	21×21
Threshold for matches	Best 200 SSD and NCC values are 'matches'
Evaluating matches	'Correct' if within 5 degrees and 5% of the angle and length of a line drawn between true matches

Table 2. Parameters for SIFT key point detection and matching. *

Threshold for matches	Smallest 200 Euclidean distances between key point descriptors are 'matches'
Evaluating matches	'Correct' if within 5 degrees and 5% of the angle and length of a line drawn between true matches

*****Default parameters were used for SIFT according to the implementation at vlfeat.org.

Matching Harris Corners and SIFT Key Points

The number of correct matches of Harris Corners based on the SSD and NCC and the number of correct matches of SIFT key points based on Euclidean distances are tabulated below. Also included are the number of Harris Corners detected at each scale and the number of SIFT key points.

Table 3. Number of Harris corners detected.

Image Pair	**1**		**2**		**3**		**4**	
Image	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
Scale 1	493	504	229	227	205	209	540	589
Scale 2	283	265	144	154	118	121	403	484
Scale 3	181	178	114	123	91	89	303	342
Scale 4	144	147	109	111	89	93	268	289

Table 4. Number of correct matches of Harris corners using SSD and NCC.

Image Pair	**1**		**2**		**3**		**4**	
Scale	**SSD**	**NCC**	**SSD**	**NCC**	**SSD**	**NCC**	**SSD**	**NCC**
1	79	130	111	104	161	136	2	11
2	74	66	80	84	109	104	5	19
3	76	70	69	70	82	78	8	37
4	61	59	50	55	75	68	12	35

Table 5. Number of SIFT key points detected and correct matches using Euclidean distances.

Image Pair	**1**		**2**		**3**		**4**	
Image	**1**	**2**	**1**	**2**	**1**	**2**	**1**	**2**
Key Points	8260	10570	2312	2750	1393	1380	11531	12559
Correct Matches	148		200		200		92	

***Note:** Total number of detected 'matches' was 200 in all cases.

In the appendix, Figures 6-25 illustrate the correct, incorrect, and validation matches in each case as blue, red, and yellow lines, respectively. Basically, more blue lines indicate better performance.

Discussion and Conclusions

As the scale is increased, the number of Harris corners that are detected decreases proportionately. For example, for image 1 in pair 1 at scale 1, 493 corners were detected, but at scale 4, only 144 corners were detected (Table 3). This has obvious and important ramifications for image matching. Basically, the larger the scale, the smaller the number of corners, and therefore smaller the number of possible matches.

It is also interesting to observe what types of corners are detected at multiple scales. Basically, for a corner to remain detectable at multiple scales, it must be a large, yet sharply-defined corner without too much noise in its vicinity that could cause it to become obscured at larger spatial scales. Such corners might be found on the outline of a large, sharply-defined object in the image, such as the edge of a building or tree where it meets the sky. For example, in image pair 1 (Figure 6-9) we can see that corners within the trees at small spatial scales do not remain at larger spatial scales since they get smoothed out. But, corners detected at the edge of the tree are still detected at larger spatial scales since the edge of the tree does not get smoothed out.

The experimental results are consistent with these ideas. In all image pairs, the number of correct matches decreases as the scale increases (Table 4). This is largely because there are fewer corners to work with. Also, depending on the type of scene that is imaged, corners that are defined at larger spatial scales (such as corners of buildings, which typically look the same) may be harder to distinguish and match correctly than corners defined at smaller spatial scales where more intricate details may be present.

Contrary to expectations, for this image set, the NCC and SSD exhibited similar performance in terms of their ability to discriminate matches from non-matches. This is reflected by the fact that the best 200 NCC and SSD values contained very similar numbers of correct matches (Table 4). This is likely because in this image set, there are only minimal differences in illumination from image to image, so the normalization applied by the NCC does not really help make the corners easier to match. Only in image pair 4 did the NCC clearly out-perform the SSD. Nevertheless, it is expected that the NCC would yield more correct matches than the SSD for a larger set of images that exhibits greater changes in illumination.

With regards to SIFT key points, we note that on the order of 10 times as many SIFT key points were detected than Harris Corners in these image pairs (Table 5). Generally, with more key points it is reasonable to expect a greater number of possible matches. So this is one advantage that SIFT key points have over Harris Corners.

Also, when we compare the results of matching Harris Corners based on SSD and NCC to matching SIFT key points based on Euclidean distances between descriptors, we find that SIFT key points are much easier to match correctly. In the case of SIFT, key points were detected at multiple spatial scales and the point combinations with the smallest 200 Euclidean distances between their descriptors were taken to be 'matches'. Remarkably, the descriptors described the key points so uniquely that many of these point combinations (often the majority) were correct matches. For example, in two of the image pairs, all 200 point combinations were correct matches (Table 5). Still, SIFT key points were not always matched correctly. Some incorrect matches may be unavoidable in portions of scenes where there is little contrast or various points or objects look the same, as in image pairs 1 and 4 (Figures 22 and 25). Nevertheless, this is a strong demonstration of why it is better to match interest points based on descriptors rather than crude metrics like the SSD or NCC. Descriptors offer a much more detailed and therefore unique description of each key point. Also, they may be invariant to both scale and rotation and they can be normalized for changes in illumination, as is the case for SIFT descriptors.

Appendix.

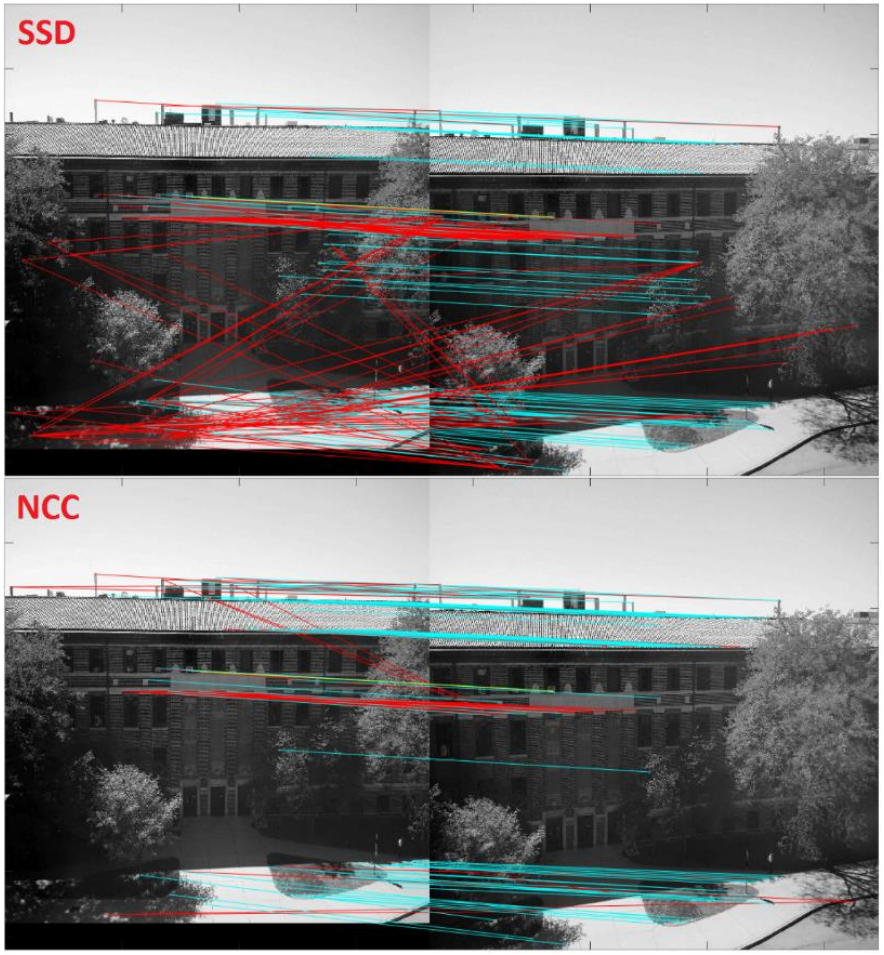


Figure 6. Pair 1 scale 1 Harris matches.

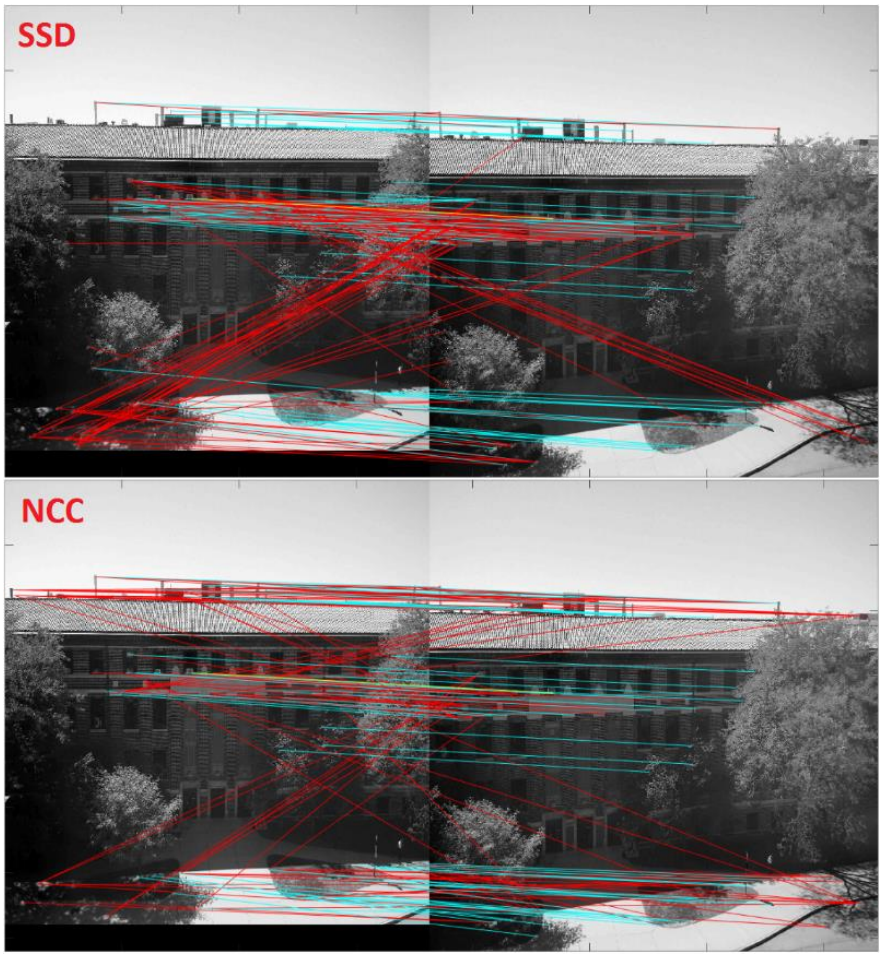


Figure 7. Pair 1 scale 2 Harris matches.

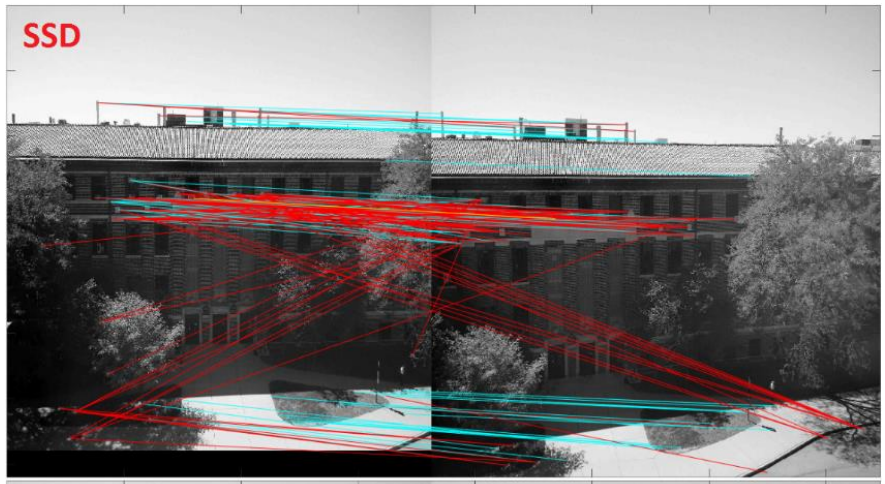


Figure 8. Pair 1 scale 3 Harris matches.

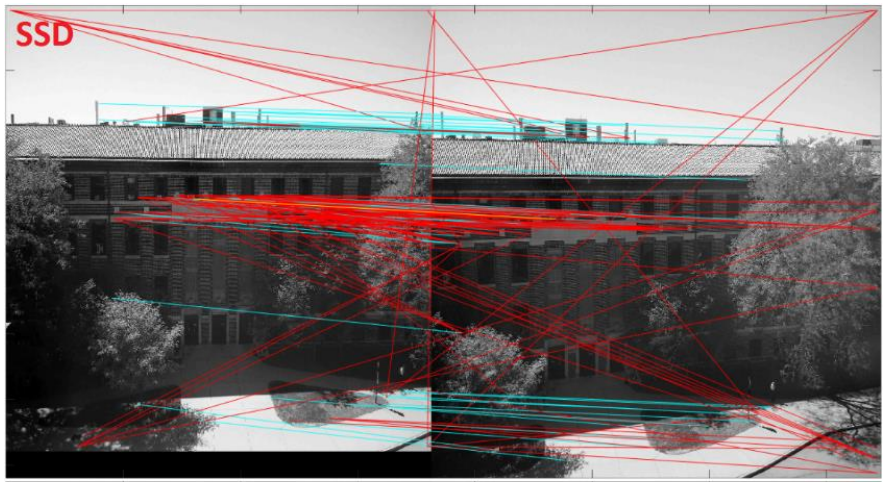


Figure 9. Pair 1 scale 4 Harris matches.

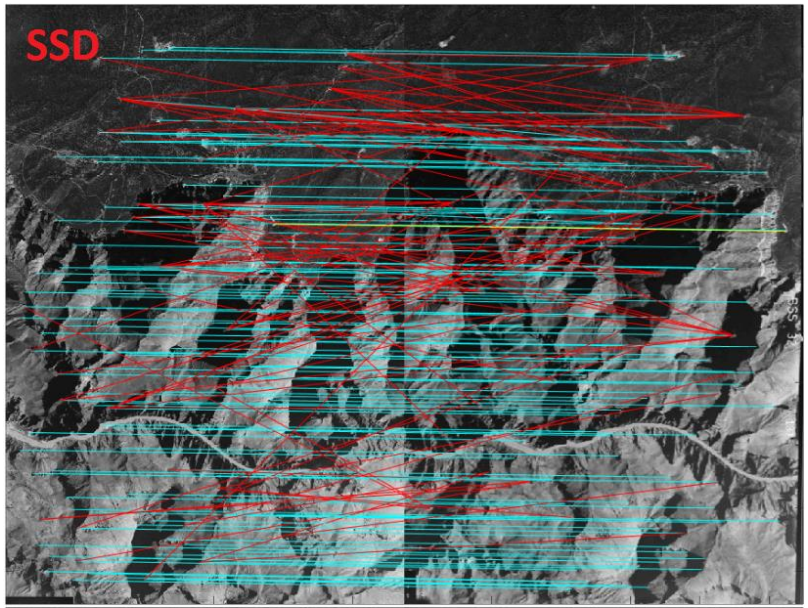


Figure 10. Pair 2 scale 1 Harris matches.

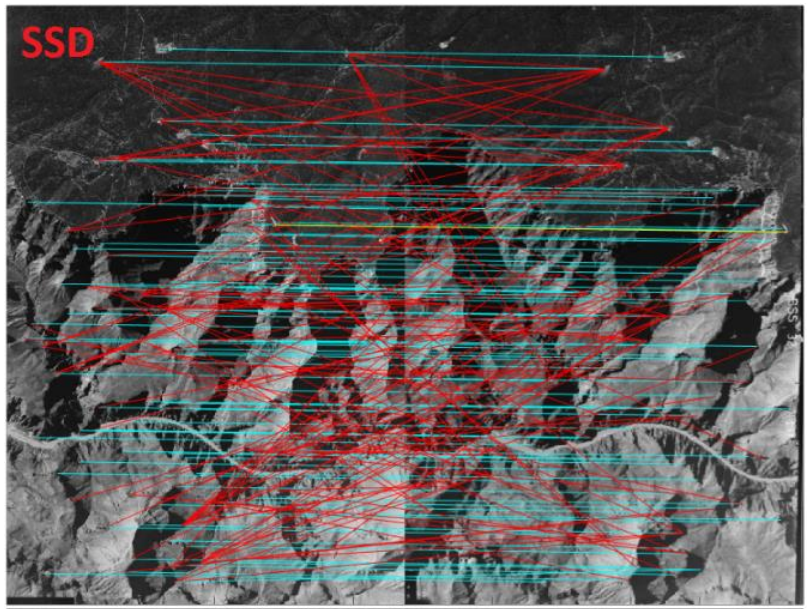
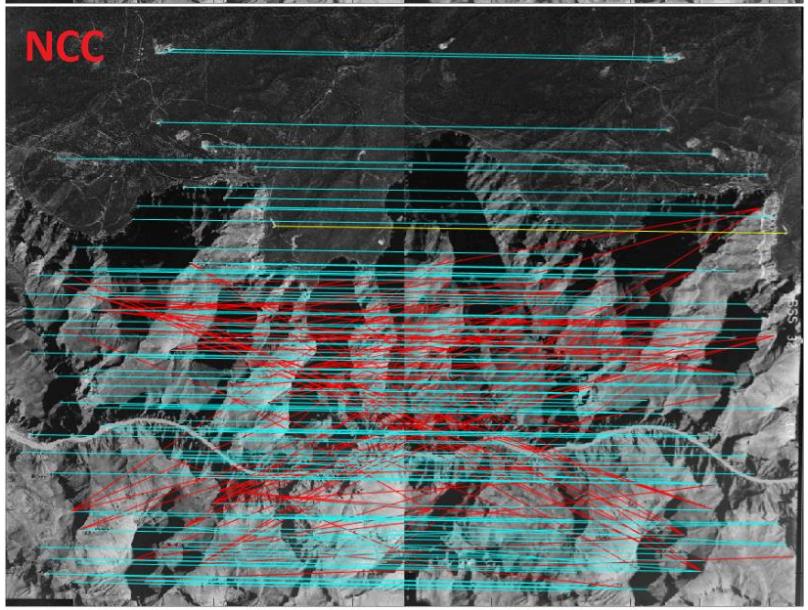
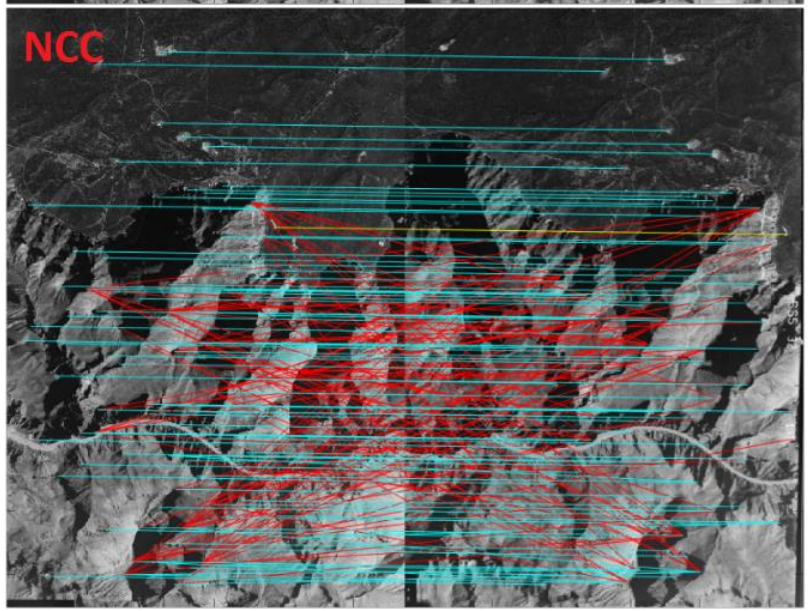


Figure 11. Pair 2 scale 2 Harris matches.





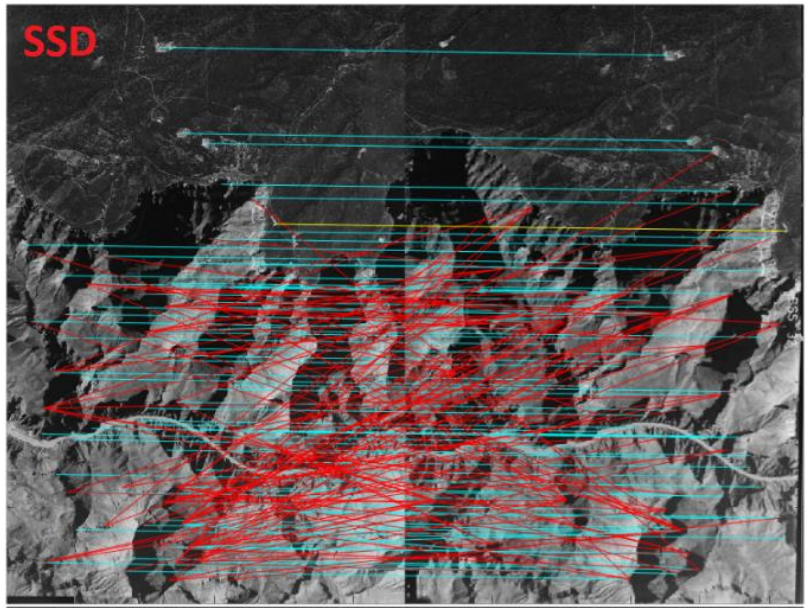


Figure 12. Pair 2 scale 3 Harris matches.

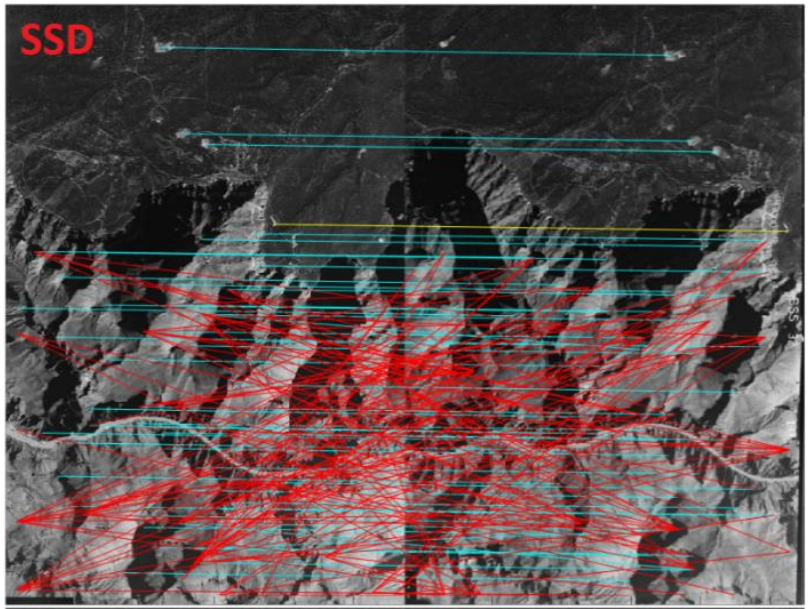


Figure 13. Pair 2 scale 4 Harris matches.

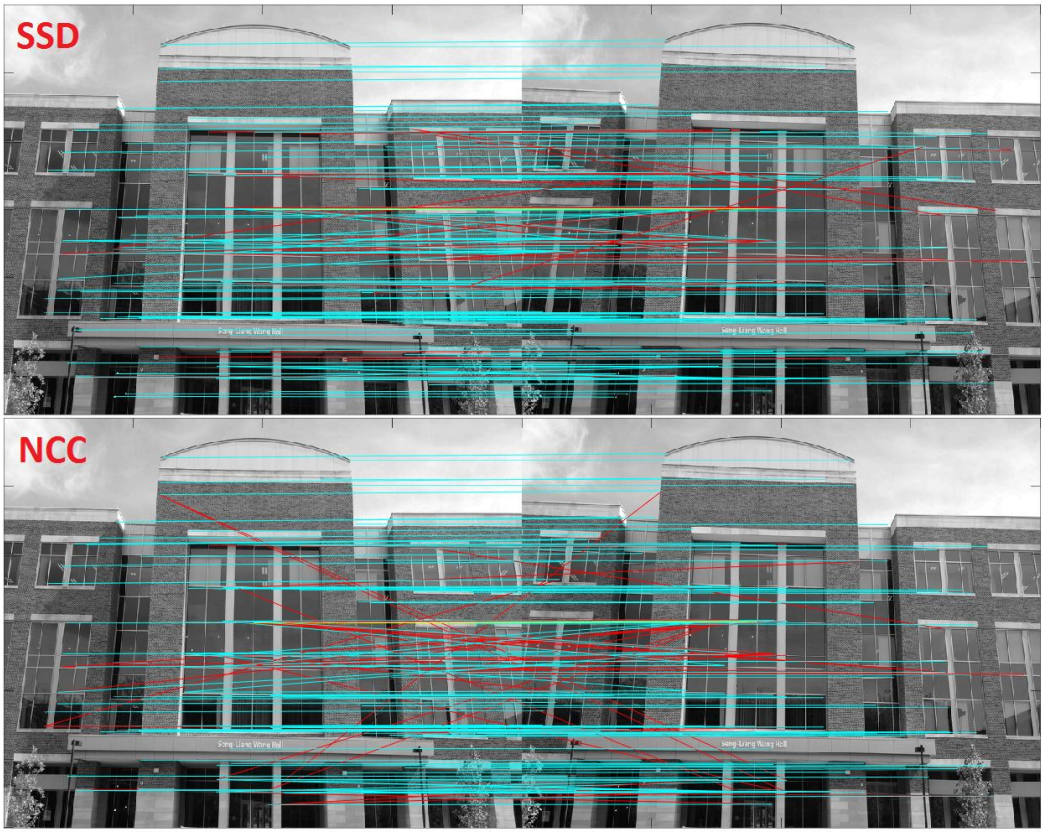


Figure 14. Pair 3 scale 1 Harris matches.

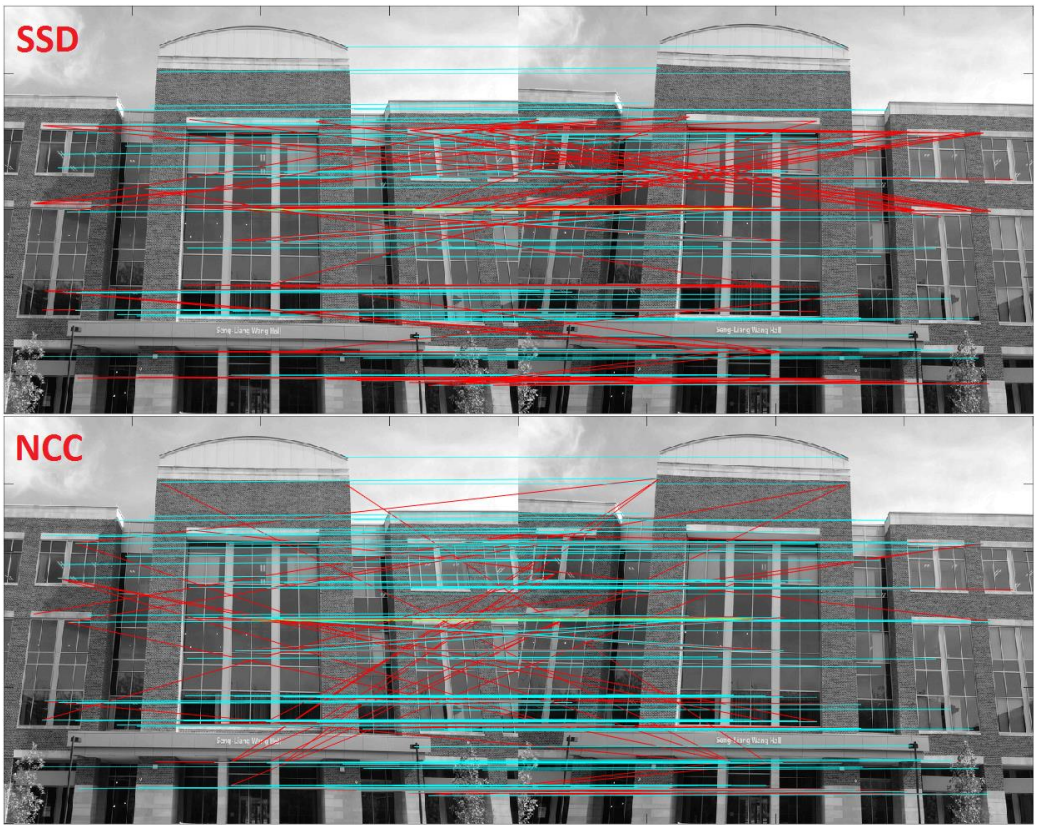


Figure 15. Pair 3 scale 2 Harris matches.

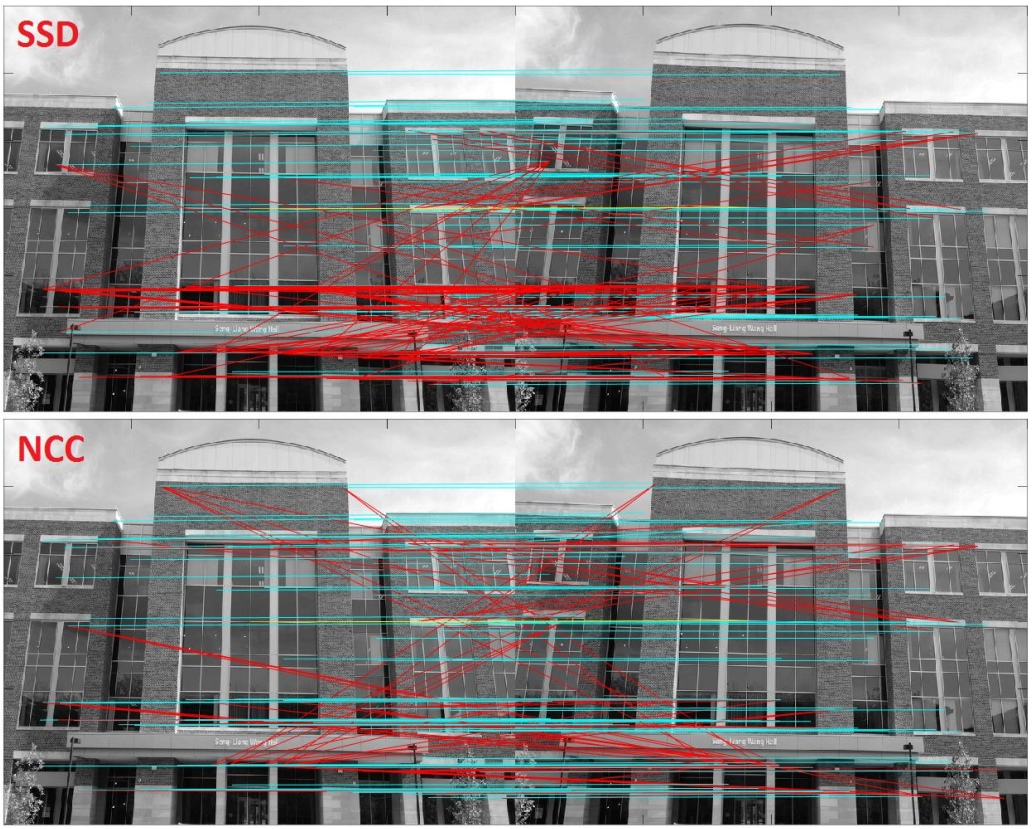


Figure 16. Pair 3 scale 3 Harris matches.

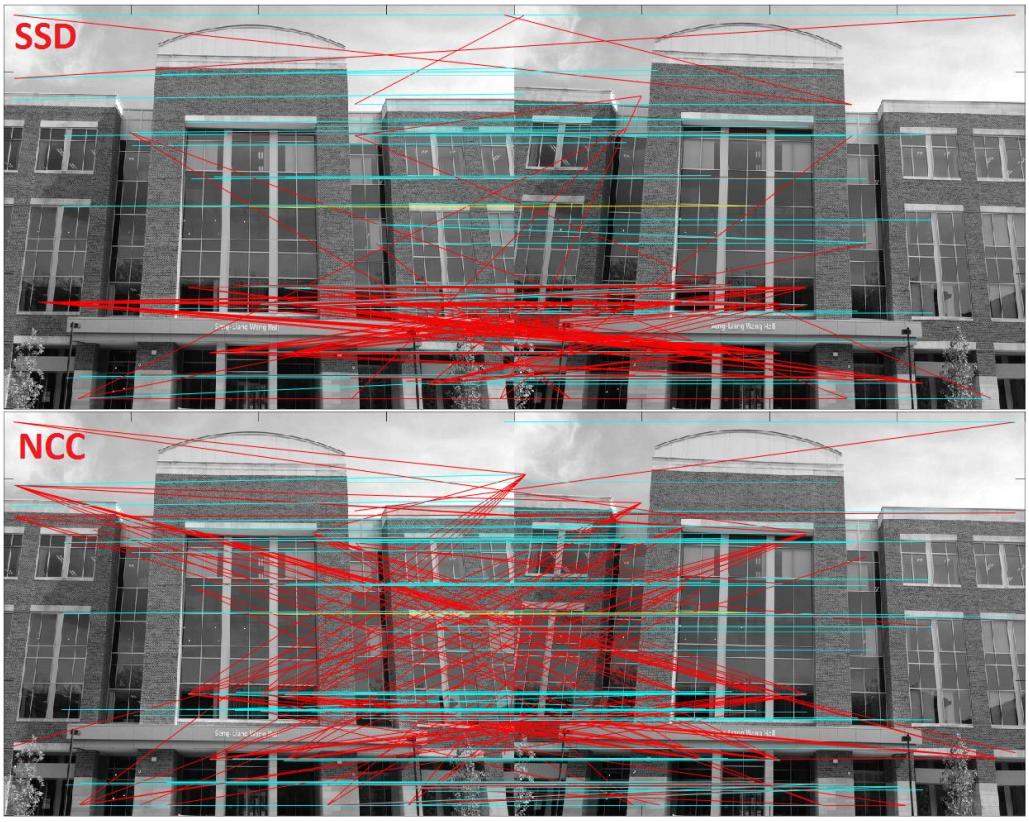


Figure 17. Pair 3 scale 4 Harris matches.

Figure 18. Pair 4 scale 1 Harris matches.

Figure 19. Pair 4 scale 2 Harris matches.

Figure 20. Pair 4 scale 3 Harris matches.

Figure 21. Pair 4 scale 4 Harris matches.

Matching SIFT Key Points

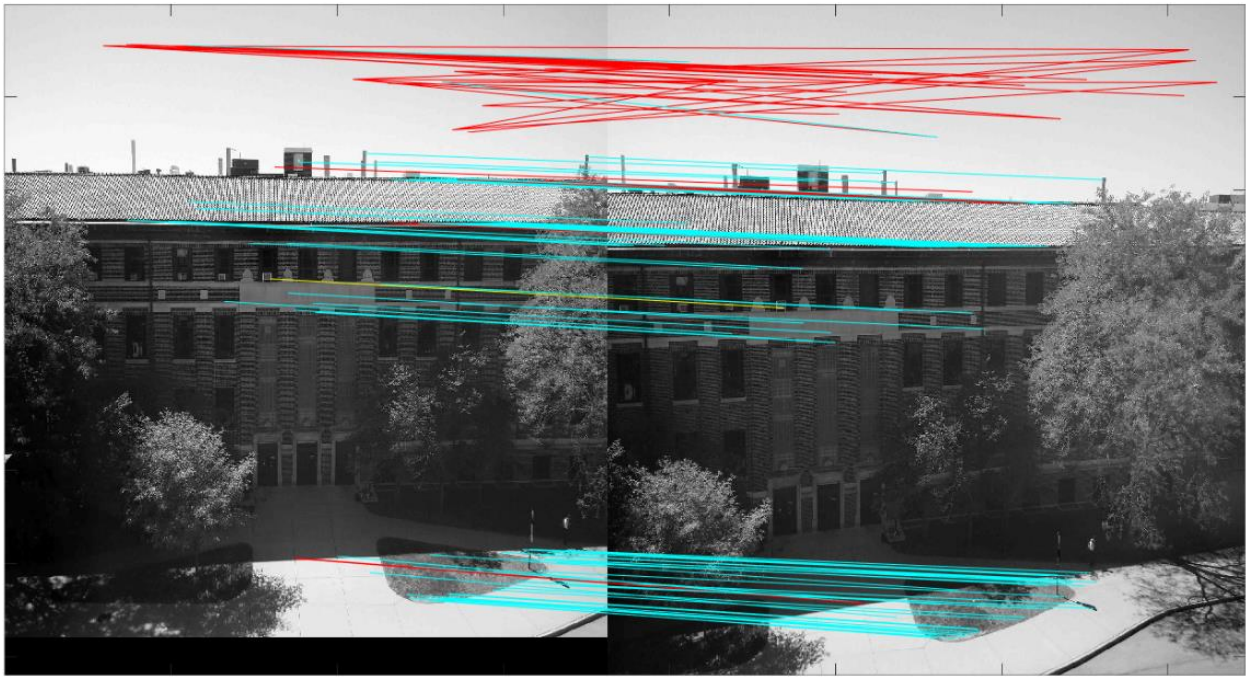


Figure 22. Pair 1 SIFT matches.

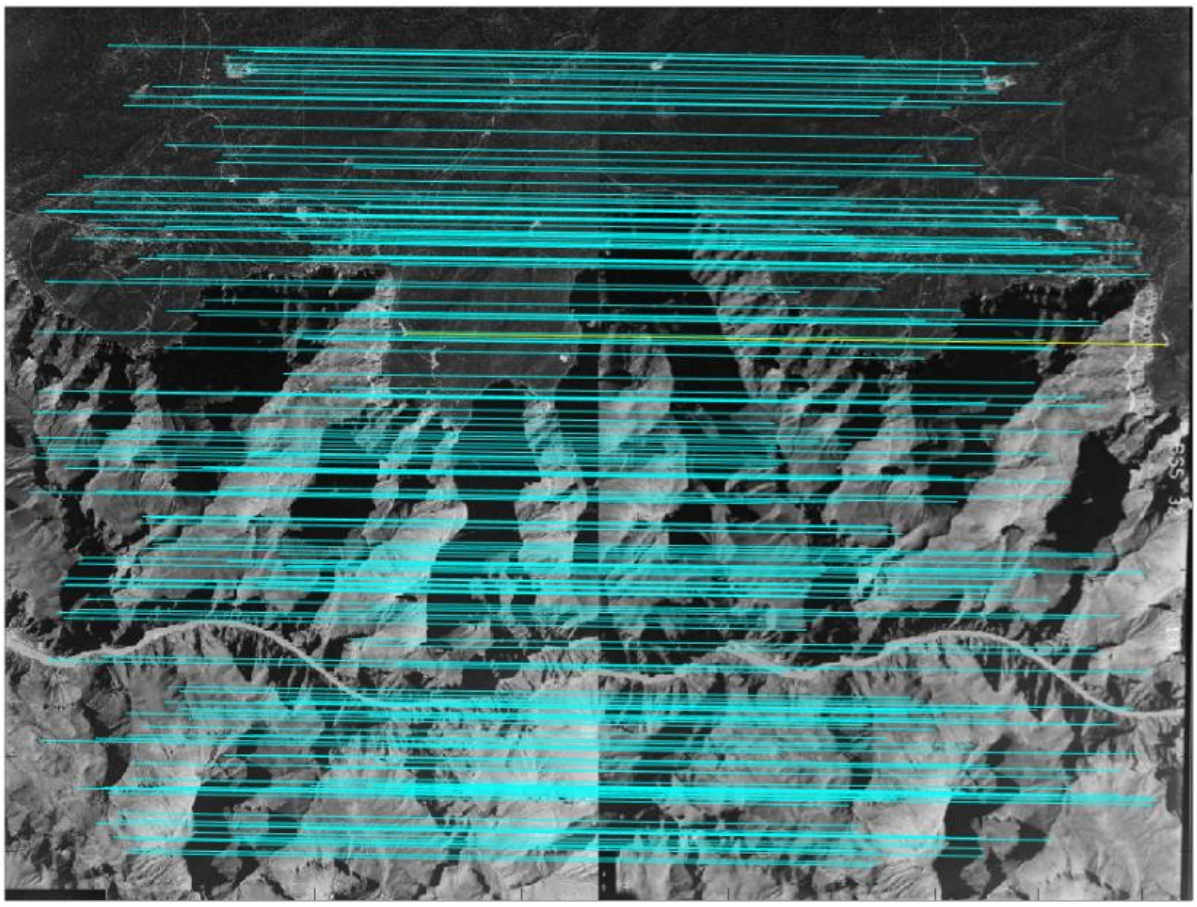


Figure 23. Pair 2 SIFT matches.

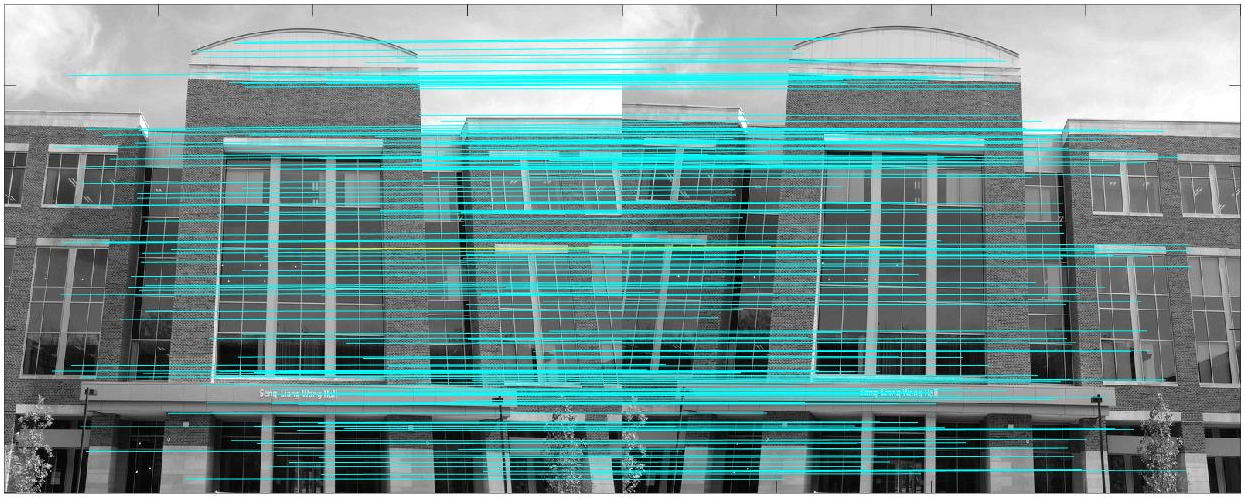


Figure 24. Pair 3 SIFT matches.

Figure 25. Pair 4 SIFT matches.

0.1 Code

0.1.1 Harris.m

```
%harris corner detection and matching

%assuming noise in the image happens at spatial scale of
    sigma = sqrt(2)
sigma = sqrt(2)*scale;

%calculate size of square haar filter
hs = round(ceil(4*sigma)/2)*2;%(haar size) (smallest even
     interger greater than 4*sigma)

%calculate size of square pixel windows that you will use
     to calculate corner scores
cs = round(ceil(5*sigma)/2)*2+1;%(corner size) (will be
    odd so there's a center pixel)

%save halfs of some window sizes
half_cs = (cs-1)/2;
half_ms = (ms-1)/2;
half_ps = (ps-1)/2;

%generate haar filters in x and y direction
hx = [-ones(hs,hs/2), ones(hs,hs/2)];
hy = [ones(hs/2,hs); -ones(hs/2,hs)];

%----------------implement algorithm on image pair
    ----------------

%
    ------------------------------------------------------------------------

%
    ------------------------------------------------------------------------

%----------------------PART 1 - FIND HARRIS CORNERS
    ----------------------------

%
    ------------------------------------------------------------------------

%
    ------------------------------------------------------------------------
```

```
%loop over the pair of images (named 1.jpg and 2.jpg)
for n = 1:2

    %load the image
    load(strcat('pair_',pair,'_img_',num2str(n),'.mat'));
    I = double(I);

    %compute x and y gradients of image at proper scale
        using haar filters
    Gx = imfilter(I,hx);
    Gy = imfilter(I,hy);
    clear I

    %compute squares and products of gradients at every
        pixel in the image (xx, yy, xy)
    Gx2 = Gx.^2;
    Gy2 = Gy.^2;
    Gxy = Gx.*Gy;
    clear Gx Gy

    %at every pixel (except ones too close to image
        boundaries) compute the Harris Detector Score R
    [nrows,ncols] = size(Gx2);
    R = zeros(nrows,ncols);%store corner scores for
        pixels
    for i = half_cs+1:nrows-half_cs
        for j = half_cs+1:ncols-half_cs
            %grab gradient values within the search
                window around this pixel
            Gx2_sub = Gx2(i-half_cs:i+half_cs, j-half_cs:
                j+half_cs);
            Gy2_sub = Gy2(i-half_cs:i+half_cs, j-half_cs:
                j+half_cs);
            Gxy_sub = Gxy(i-half_cs:i+half_cs, j-half_cs:
                j+half_cs);

            %compute sums of gradients within the search
                window
            Sx2 = sum(sum(Gx2_sub));
            Sy2 = sum(sum(Gy2_sub));
            Sxy = sum(sum(Gxy_sub));

            %define matrix H at each pixel
            H = [ Sx2 Sxy ;...
                  Sxy Sy2 ];
```

```
            %compute response of harris corner detector
                for the pixel (i,j)
            R(i,j) = det(H) - k*(trace(H))^2;
        end
    end
    clear Gx2 Gy2 Gxy Gx2_sub Gy2_sub Gxy_sub

    %find local maxima of the corner responses
    C = uint8(zeros(nrows,ncols));%make 1 for corner,
        leave as 0 for non corner,
    for i = half_ms+1:nrows-half_ms
        for j = half_ms+1:ncols-half_ms
            %grab corner response values within the
                search window around this pixel
            R_sub = R(i-half_ms:i+half_ms, j-half_ms:j+
                half_ms);

            %only assign a 1 to this pixel if it has the
                maximum corner
            %response within the search window and its
                non-negative and it
            %has a corner response that is relatively
                large compared to the
            %overall image (its not in a smooth part of
                the image)
            if ( R(i,j) == max(max(R_sub)) ) && ( R(i,j)
                > 0 ) && ( abs(R(i,j)) > mean(mean(abs(R))
                ) )
                C(i,j) = uint8(1);
            end
        end
    end
    clear R R_sub

    %get the row,column indices of the corners (and save
        images of corners)
    if n == 1
        [rows1, cols1] = find(C == uint8(1));
    else
        [rows2, cols2] = find(C == uint8(1));
    end
    clear C

end
```

```
%
    ____________________________________________________________________

%
    ____________________________________________________________________

%----------PART 2 - FIND CORRESPONDENCES BETWEEN IMAGES
    ____________________
%
    ____________________________________________________________________

%
    ____________________________________________________________________


%load the grayscale images (in units of double)
load(strcat('pair_',pair,'_img_1.mat'))
I1 = double(I);
load(strcat('pair_',pair,'_img_2.mat'))
I2 = double(I);
clear I

%okay...so we need to grab local regions of pixels around
     all of the corners in each
%of the images...then we need to compare all possible
    combinations of corners in the
%two images and keep only the best matching corners...
    where best match is determined
%by lowest SSD or highest NCC
n_pts_1 = length(cols1);
n_pts_2 = length(cols2);
n_combinations = n_pts_1*n_pts_2;
pt_data = zeros(n_combinations,4);%4 columns correspond
    to (image 1 pt id),(image 2 pt id), (ssd), (ncc), and
    there are as many rows as there are possible
    combinations of interest pts in the two images
idx = 0;%index for combination number
for n = 1:n_pts_1

    %grab image coordinates of the point in image 1
    i1 = rows1(n);
    j1 = cols1(n);

    %grab pixels around the point in image 1
    sub1 = I1(i1-half_ps:i1+half_ps, j1-half_ps:j1+
        half_ps);
```

```
        %compute statistics for sub1 (need for ncc
            calculations)
        mu1 = mean(mean(sub1));
        dev1 = sub1-mu1;

        for m = 1:n_pts_2

            %grab image coordinates of points
            i2 = rows2(m);
            j2 = cols2(m);

            %grab pixels around the point in image 2
            sub2 = I2(i2-half_ps:i2+half_ps, j2-half_ps:j2+
                half_ps);

            %compute statistics for sub2 (need for ncc
                calculations)
            mu2 = mean(mean(sub2));
            dev2 = sub2-mu2;

            %compute NCC
            ncc = sum(sum(dev1.*dev2)) / sqrt( sum(sum(dev1
                .^2))*sum(sum(dev2.^2)) );

            %compute SSD
            ssd = sum(sum((sub1 - sub2).^2));

            %compute the point combination index
            idx = idx+1;

            %store the point data
            pt_data(idx,1) = n;
            pt_data(idx,2) = m;
            pt_data(idx,3) = ssd;
            pt_data(idx,4) = ncc;

        end
    end

    % %sort the ssd and ncc data and plot them to see how
        they change... try to
    % %identify the cutoff where it goes from matches to non-
        matches
    % ssd = sort(pt_data(:,3),'ascend');
    % ncc = sort(pt_data(:,4),'descend');
```

```
% figure(1)
% plot(1:1000,ssd(1:1000),'b')
% title('Sum of Squared Differences')
% figure(2)
% plot(1:1000,ncc(1:1000),'r')
% title('Normalized Cross Correlation')

%NOTE - the above plots clearly show that there IS no
    obvious cutoff of sdd
%or ncc values that separates good matches from bad
    matches. therefore,
%there really isn't any sensible way of identifying
    matches from these data
%alone, other than just taking the top n_matches and
    later evaluating their
%accuracy by separate means.

%get ssd matches
pt_data = sortrows(pt_data,3);
ssd_pt_data = pt_data(1:n_matches,:);
ssd_matches = zeros(n_matches,4);
for i = 1:n_matches
    n = ssd_pt_data(i,1);
    m = ssd_pt_data(i,2);
    match = [ rows1(n) cols1(n) rows2(m) cols2(m) ];%
        store match as [i1 j1 i2 j2] row vector
    ssd_matches(i,:) = match;%stack match row vectors
end

%get ncc matches
pt_data = sortrows(pt_data,4);
ncc_pt_data = pt_data(n_combinations-n_matches+1:
    n_combinations,:);%note, sorted in ascending order, so
     take last n_matches entries
ncc_matches = zeros(n_matches,4);
for i = 1:n_matches
    n = ncc_pt_data(i,1);
    m = ncc_pt_data(i,2);
    match = [ rows1(n) cols1(n) rows2(m) cols2(m) ];%
        store match as [i1 j1 i2 j2] row vector
    ncc_matches(i,:) = match;%stack match row vectors
end

%open the images in figures, draw the matches on the
    figures, and then save the figures
%append the images
```

```
if (nrows1 < nrows2)
     I1(nrows2,1) = 0;
else
     I2(nrows1,1) = 0;
end
I3 = [I1 I2];

%draw ssd matches and store line angles and lengths
ssd_angles = zeros(length(ssd_matches(:,1)),1);
ssd_lengths = zeros(length(ssd_matches(:,1)),1);
figure('Position', [100 100 size(I3,2) size(I3,1)]);
colormap('gray');
imagesc(I3);
hold on;
for i = 1:length(ssd_matches(:,1))
    %get line endpoints
    x1 = ssd_matches(i,2);
    x2 = ssd_matches(i,4)+ncols1;
    y1 = ssd_matches(i,1);
    y2 = ssd_matches(i,3);
    %get the angle of the line
    %angle_in_deg=atan2(y2-y1,x2-x1)*180/pi
    ssd_angles(i) = (180/pi)*atan2(y2-y1, x2-x1);
    %get the length of the line
    ssd_lengths(i) = sqrt((x2-x1)^2+(y2-y1)^2);

    %if the angle and distance of the line are reasonable
        , draw in red
    if ( (ssd_angles(i) > angl-a) && (ssd_angles(i) <
        angl+a) && (ssd_lengths(i) > dist-d) && (
        ssd_lengths(i) < dist+d) )
         line( [ x1 x2 ],[ y1 y2 ], 'Color', 'c');%draw
             lines between matches using line(X,Y) (shift x
             's for image 2 by ncols1)
    else%if the angle and distance are not reasonable,
        draw in cyan (blue)
         line( [ x1 x2 ],[ y1 y2 ], 'Color', 'r');%draw
             lines between matches using line(X,Y) (shift x
             's for image 2 by ncols1)
    end

end
%draw the validation line on the figure in yellow
line( [ X1 X2 ],[ Y1 Y2 ], 'Color', 'y');
hold off;
```

```
%draw ncc matches and store line angles and lengths
ncc_angles = zeros(length(ncc_matches(:,1)),1);
ncc_lengths = zeros(length(ssd_matches(:,1)),1);
figure('Position', [100 100 size(I3,2) size(I3,1)]);
colormap('gray');
imagesc(I3);
hold on;
for i = 1:length(ncc_matches(:,1))
    %get line endpoints
    x1 = ncc_matches(i,2);
    x2 = ncc_matches(i,4)+ncols1;
    y1 = ncc_matches(i,1);
    y2 = ncc_matches(i,3);
    %get the angle of the line
    %angle_in_deg=atan2(y2-y1,x2-x1)*180/pi
    ncc_angles(i) = (180/pi)*atan2(y2-y1, x2-x1);
    %get the length of the line
    ncc_lengths(i) = sqrt((x2-x1)^2+(y2-y1)^2);

    %if the angle and distance of the line are reasonable
        , draw in red
    if ( (ncc_angles(i) > angl-a) && (ncc_angles(i) <
        angl+a) && (ncc_lengths(i) > dist-d) && (
        ncc_lengths(i) < dist+d) )
        line( [ x1 x2 ],[ y1 y2 ], 'Color', 'c');%draw
            lines between matches using line(X,Y) (shift x
            's for image 2 by ncols1)
    else%if the angle and distance are not reasonable,
        draw in cyan (blue)
        line( [ x1 x2 ],[ y1 y2 ], 'Color', 'r');%draw
            lines between matches using line(X,Y) (shift x
            's for image 2 by ncols1)
    end
end
%draw the validation line on the figure in yellow
line( [ X1 X2 ],[ Y1 Y2 ], 'Color', 'y');
hold off;

%figure out how many false-positives there were for each
    method based on the angles of the lines formed in the
    figure
ssd_correct_matches = sum( (ssd_angles > angl-a).*(
    ssd_angles < angl+a).*(ssd_lengths > dist-d).*(
    ssd_lengths < dist+d) )
ssd_false_positives = n_matches - ssd_correct_matches
```

```
ncc_correct_matches = sum( (ncc_angles > angl-a).*(
    ncc_angles < angl+a).*(ncc_lengths > dist-d).*(
    ncc_lengths < dist+d) )
ncc_false_positives = n_matches - ncc_correct_matches

toc
```

0.1.2 Sift.m

```
%sift feature detection and matching

%--------------implement algorithm on image pair
    ------------------

%
    %-------------------------------------------------------------------------

%
    %-------------------------------------------------------------------------

% %-----------------------PART 1 - FIND SIFT FEATURES
    ---------------------------
%
    %-------------------------------------------------------------------------

%
    %-------------------------------------------------------------------------


run(VLfeat_setup_path)

%load the grayscale images (in units of single)
load(strcat('pair_',pair,'_img_1.mat'))
I1 = single(I);
load(strcat('pair_',pair,'_img_2.mat'))
I2 = single(I);
clear I

[f1,d1] = vl_sift(I1);
[f2,d2] = vl_sift(I2);

cols1 = f1(1,:);%x coordinates of keypoints
rows1 = f1(2,:);%y coordinates of keypoints

cols2 = f2(1,:);%x coordinates of keypoints
```

```
rows2 = f2(2,:);%y coordinates of keypoints


%
    %-------------------------------------------------------------------------

%
    %-------------------------------------------------------------------------

% %-------------------------PART 2 - FIND SIFT MATCHES
    -------------------------
%
    %-------------------------------------------------------------------------

%
    %-------------------------------------------------------------------------


%d has descriptors for sift keypoints as columns, so you
    just need to
%directly compare the descriptors using euclidean
    distance... so now
%euclidean distance is your "match" metric
n_pts_1 = length(cols1);
n_pts_2 = length(cols2);
n_combinations = n_pts_1*n_pts_2;
pt_data = zeros(n_combinations,3);%4 columns correspond
    to (image 1 pt id),(image 2 pt id), (ssd), (ncc), and
    there are as many rows as there are possible
    combinations of interest pts in the two images
idx = 0;%index for combination number
for n = 1:n_pts_1

    %grab descriptor for point in image 1
    D1 = double(d1(:,n));

    for m = 1:n_pts_2

        %grab descriptor for point in image 1
        D2 = double(d2(:,m));

        %compute euclidean distance
        euc = sqrt( sum( (D1-D2).^2) );

        %compute the point combination index
        idx = idx+1;
```

```
        %store the point data
        pt_data(idx,1) = n;
        pt_data(idx,2) = m;
        pt_data(idx,3) = euc;

    end
end

%get sift matches
pt_data = sortrows(pt_data,3);
pt_data = pt_data(1:n_matches,:);
sift_matches = zeros(n_matches,4);
for i = 1:n_matches
    n = pt_data(i,1);
    m = pt_data(i,2);
    match = [ rows1(n) cols1(n) rows2(m) cols2(m) ];%
        store match as [i1 j1 i2 j2] row vector
    sift_matches(i,:) = match;%stack match row vectors
end

%open the images in figures, draw the matches on the
    figures, and then save the figures
%append the images
if (nrows1 < nrows2)
     I1(nrows2,1) = 0;
else
     I2(nrows1,1) = 0;
end
I3 = [I1 I2];

%draw sift matches and store line angles and lengths
sift_angles = zeros(length(sift_matches(:,1)),1);
sift_lengths = zeros(length(sift_matches(:,1)),1);
figure('Position', [100 100 size(I3,2) size(I3,1)]);
colormap('gray');
imagesc(I3);
hold on;
for i = 1:length(sift_matches(:,1))
    %get line endpoints
    x1 = sift_matches(i,2);
    x2 = sift_matches(i,4)+ncols1;
    y1 = sift_matches(i,1);
    y2 = sift_matches(i,3);
    %get the angle of the line
    %angle_in_deg=atan2(y2-y1,x2-x1)*180/pi
```

```
    sift_angles(i) = (180/pi)*atan2(y2-y1, x2-x1);
    %get the length of the line
    sift_lengths(i) = sqrt((x2-x1)^2+(y2-y1)^2);

    %if the angle and distance of the line are reasonable
        , draw in red
    if ( (sift_angles(i) > angl-a) && (sift_angles(i) <
        angl+a) && (sift_lengths(i) > dist-d) && (
        sift_lengths(i) < dist+d) )
        line( [ x1 x2 ],[ y1 y2 ], 'Color', 'c');%draw
            lines between matches using line(X,Y) (shift x
            's for image 2 by ncols1)
    else%if the angle and distance are not reasonable,
        draw in cyan (blue)
        line( [ x1 x2 ],[ y1 y2 ], 'Color', 'r');%draw
            lines between matches using line(X,Y) (shift x
            's for image 2 by ncols1)
    end

end
%draw the validation line on the figure in yellow
line( [ X1 X2 ],[ Y1 Y2 ], 'Color', 'y');
hold off;

%figure out how many false-positives there were for each
    method based on the angles of the lines formed in the
    figure
sift_correct_matches = sum( (sift_angles > angl-a).*(
    sift_angles < angl+a).*(sift_lengths > dist-d).*(
    sift_lengths < dist+d) )
sift_false_positives = n_matches - sift_correct_matches

toc
```

0.1.3 HW4 Inputs.m

```
%format inputs for harris and sift algorithms

tic;
format compact
%--------------------specify inputs
    ------------------------------


%select algorithm
method = 2;%1 for harris corners, 2 for sift features
```

```
%select image scale at which you search for features and
    matches (1, 2, 3, or 4)
scale = 1;

%image directory (uncomment correct directory)
% img_directory = 'U:\Personal\ece-661\hw4\HW4Pics\pair1
    \';
% img_directory = 'U:\Personal\ece-661\hw4\HW4Pics\pair2
    \';
% img_directory = 'U:\Personal\ece-661\hw4\HW4Pics\pair3
    \';
img_directory = 'U:\Personal\ece-661\hw4\HW4Pics\pair4\';

%VLfeat SIFT folder directory
VLfeat_setup_path = 'U:\Personal\ece-661\hw4\
    final_submission\vlfeat-0.9.20\toolbox\vl_setup';

%manually measured matches in each pair of images
%pair 1 in row 1, pair 2 in row 2, pair 3 in row 3 as [y1
     x1 y2 x2] row vectors
manual_matches = [ 590  799  652  534  ;...%pair 1
                   349  388  360  551  ;...%pair 2
                   301  386  299  364  ;...%pair 3
                   428  1798 396  851  ];  %pair 4


%number of matches you want to obtain (recommended not to
     exceed 200)
n_matches = 200;

%constant k for computing harris detector responses
k = 0.04;%0.04 to 0.06 recommended in literature

%size of search window for finding local maxima of corner
     detector responses
ms = 29;%pixels (maxima search) (must be odd)

%size of window for computing sum of squared differences
    or normalized cross corelation to match corners in the
     image pairs
ps = 21;%pixels (pair search) (must be odd)

%maximum allowable change in the angle of the line drawn
    between matches in
```

```
%the images (used to identify false-positives after
    grabbing the specified
%number of "matches")
a = 5;%degrees

%maximum allowable fractional change in the length of the
     line drawn
%between matches in the images (used to identify false-
    positives after
%grabbing the specified number of "matches")
f = 0.05;%fraction

%------------------calculated inputs
    ---------------------------


%get the pair number
pair = img_directory(length(img_directory)-1);

for n = 1:2

    %get the image filename
    img_name = strcat(num2str(n),'.jpg');
    img_filename = strcat(img_directory,img_name);

    %read the image
    I = imread(img_filename);

    %convert image to grayscale
    I = rgb2gray(I);

    %get image dimensions
    if n == 1
        nrows1 = size(I,1);
        ncols1 = size(I,2);
    else
        nrows2 = size(I,1);
        ncols2 = size(I,2);
    end

    %save the image as a .mat file for fast reloading
        later
    save(strcat('pair_',pair,'_img_',num2str(n),'.mat'),'
        I');

end
```

```
%calculate reference angles and lengths for lines drawn
    between
%correspondences when the images are placed side-by-side
    (used
%to help identify incorrect matches)
X1 = manual_matches(str2num(pair),2);%stored as [y1 x1 y2
     x2]
X2 = manual_matches(str2num(pair),4)+ncols1;
Y1 = manual_matches(str2num(pair),1);
Y2 = manual_matches(str2num(pair),3);
angl = (180/pi)*atan2(Y2-Y1, X2-X1);
dist = sqrt((X2-X1)^2+(Y2-Y1)^2);
d = dist*f;

%run the selected alrogithm
if method == 1
    disp('harris')
    m160923__ece661_hw4_harris__ah
else
    disp('sift')
    m160923__ece661_hw4_sift__ah
end
```