CSE 252B - Homework 4

Lingting Ge A53212800

3/7/2017

1 Feature detection

Download input data from the course website. The file *price_center*20.*JPG* contains image1 and the file *price_center*21.*JPG* contains image 2. In your report, include a figure containing the pair of input images.

For each input image, calculate an image where each pixel value is the minor eigenvalue of the gradient matrix

$$\mathrm{N} = \begin{bmatrix} \Sigma_w I_x^2 & \Sigma_w I_x I_y \\ \Sigma_w I_x I_y & \Sigma_w I_y^2 \end{bmatrix} \tag{1}$$

where w is the window about the pixel, and I_x and I_y are the gradient images in the x and y direction, respectively. Calculate the gradient images using the five point central difference operator. Set resulting values that are below a specified threshold value to zero (hint: calculating the mean instead of the sum in N allows for adjusting the size of the window without changing the threshold value). Apply an operation that suppresses (sets to 0) local (i.e., about a window) nonmaximum pixel values in the minor eigenvalue image. Vary these parameters such that around 600-650 features are detected in each image. For resulting nonzero pixel values, determine the subpixel feature coordinate using the Forstner corner point operator.

In your report, state the size of the feature detection window (i.e., the size of the window used to calculate the elements in the gradient matrix N), the minor eigenvalue threshold value, the size of the local nonmaximum suppression window, and the resulting number of features detected in each image. Additionally, include a figure containing the pair of images, where the detected features (after local nonmaximum suppression) in each of the images are indicated by a square about the feature, where the size of the square is the size of the detection window.

Solution:

The parameters I used are as follows:

Size of the feature detection window is 9 * 9;
The minor eigenvalue threshold value is 380;
The size of the local non maximum suppression window is 9 * 9.

The resulting numbers of features detected are 646 and 631 for the first and second image respectively.

Here is the procedure that I solve this problem:

- Read the image and transform the image into gray scale image.
- Calculate gradient images I_x and I_y respectively.
- Select a window size and calculate gradient matrix.
- Calculate the minor eigenvalue image.
- Compute the non maxima suppression.

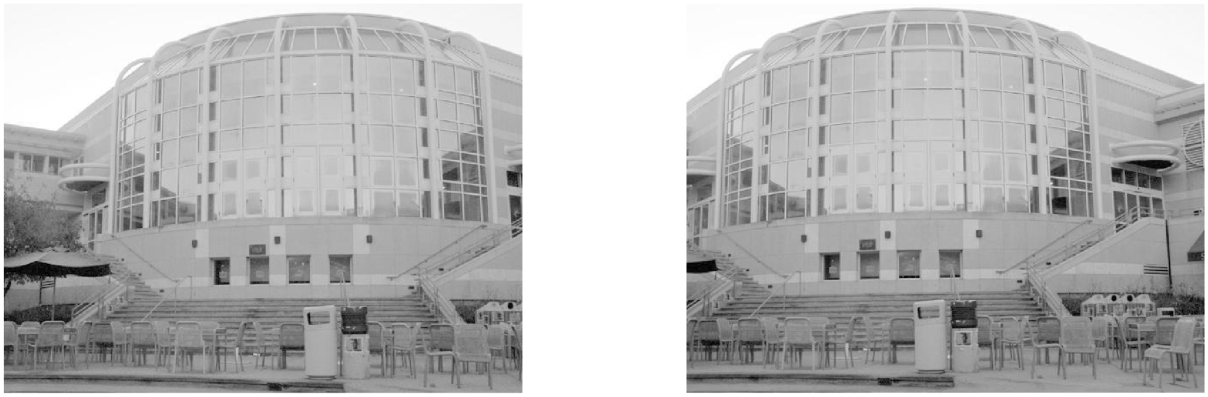

Figure 1: Original figure

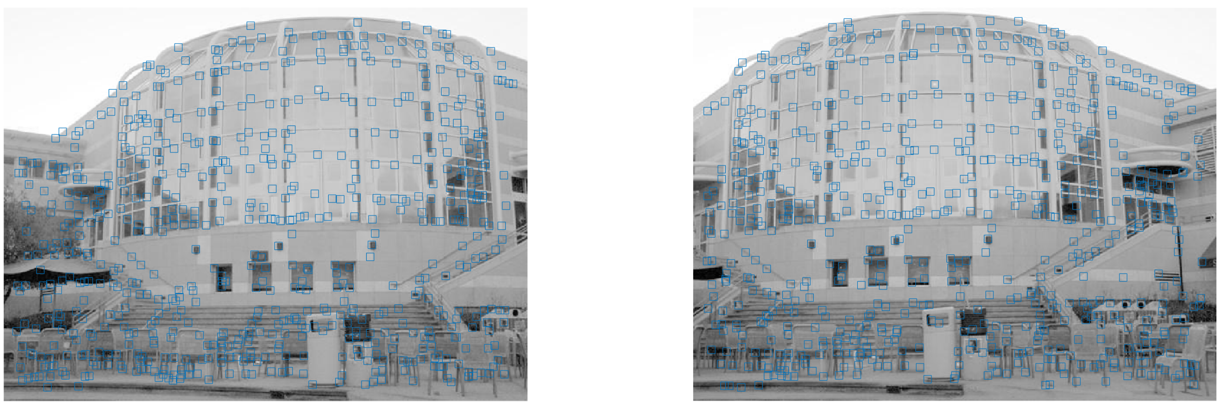

Figure 2: Corner detection

- Using Forstner corner point method to find the coordinates of the corners.

The original figures are shown in Figure 1.

The detected features are shown in Figure 2.

2 Feature matching

Determine the set of one-to-one putative feature correspondences by performing a brute-force search for the greatest correlation coefficient value (in the range [-1, 1]) between the detected features in image 1 and the detected features in image 2. Only allow matches that are above a specified correlation coefficient threshold value (note that calculating the correlation coefficient allows for adjusting the size of the matching window without changing the threshold value). Further, only allow matches that are above a specified distance ratio threshold value, where distance is measured to the next best match for a given feature. Vary these parameters such that around 200 putative feature correspondences are established. Optional: constrain the search to coordinates in image 2 that are within a proximity of the detected feature coordinates in image 1.

In your report, state the size of the proximity window (if used), the correlation coefficient threshold value, the distance ratio threshold value, and the resulting number of putative feature correspondences (i.e., matched features). Additionally, include a figure containing the pair of images, where the matched features in each of the images are indicated by a square about the feature, where the size of the square is the size of the matching window, and a line segment is drawn from the feature to the coordinates of the corresponding feature in the other image (see Fig. 4.9(e) in the Hartley & Zisserman book as an example).

Solution:

The parameters I used are as follows:

The correlation coefficient threshold value is 0.6;
The distance ratio threshold value is 0.7;
The window size is 21 * 21.
I didn't use the proximity window.

The resulting number of putative feature correspondences is 201.

Here is the process that I used to solve this problem:

- Fetch the corners of two features from the result of question (a).
- Take out all windows corresponding to their corners.
- Calculate correlation coefficients of window pair between all window in two figures.
- Do the one-to-one matching as follows:
  - Find the indices of element with maximum value.
  - If the maximum value is larger than the similarity threshold, do the follows:
    - Store the best match value.
    - Set this element value to -1.
    - Find the next best match value as the following equation:

$$nextbestmatch = max(nextbestmatchvalueinrow, nextbestmatchvalueincolumn) \tag{2}$$

    - If the $(1 - bestmatchvalue) < (1 - nextbestmatchvalue) * distanceratiothreshold$, then store the feature match. Otherwise, this match is not unique enough.
    - Set the row and column to be -1.
- Otherwise, stop the iteration.

The final feature matching images are shown in Figure 3:

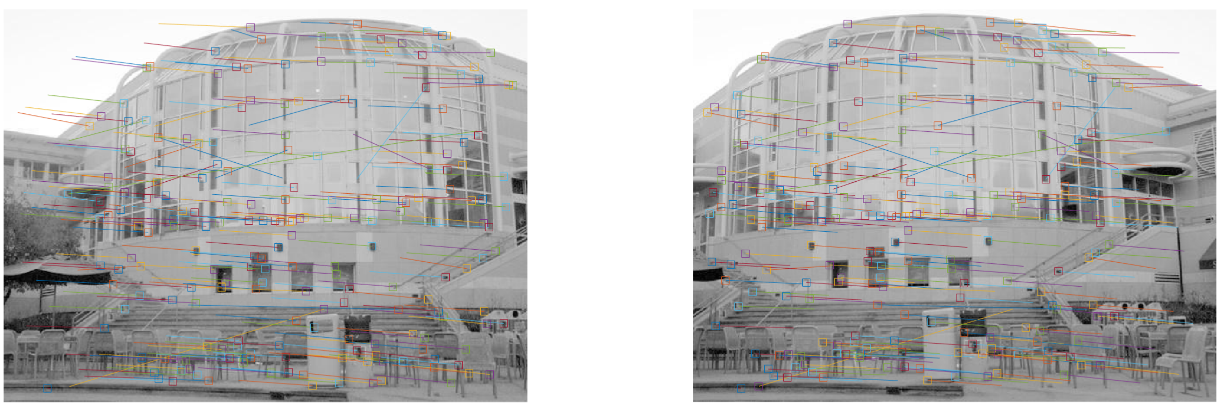

Figure 3: Feature mapping

3 Outlier rejection

The resulting set of putative point correspondences should contain both inlier and outlier correspondences (i.e., false matches). Determine the set of inlier point correspondences using the M-estimator Sample Consensus (MSAC) algorithm, where the maximum number of attempts to find a consensus set is determined adaptively. For each trial, you must use the 4-point algorithm (as described in lecture) to estimate the planar projective transformation from the 2D points in image 1 to the 2D points in image 2. Calculate the (squared) Sampson error as a first order approximation to the geometric error.

In your report, describe any assumptions, including the probability p that at least one of the random samples does not contain any outliers (used to determine the number of attempts to find a consensus set), and the probability α that a given data point is an inlier and the variance σ^2 of the measurement error (both used to determine the distance threshold; hint: this problem has codimension 2). State the resulting number of inliers and the number of attempts to find the consensus set.

Additionally, include a figure containing the pair of images, where the inlier features in each of the images are indicated by a square about the feature, where the size of the square is the size of the matching window, and a line segment is drawn from the feature to the coordinates of the corresponding feature in the other image (see Fig. 4.9(g) in the Hartley & Zisserman book as an example).

Solution:

The parameters I used are as follows:

Probability p = 0.99;
Probability $\alpha = 0.95$;
Variance σ^2 assumed to be 1;

The number of inliers I find is 163;
The number of trials is 11.

The random seed I used is rand('seed', 0);

Here are the general steps that I used to solve this question is as follows:

- For initialization, set max trials and minimum cost to be infinity. And number of trials to be 0;
- Begin the iteration of MASC method.
- Select four random points for two figures respectively.
- Compute the matrix H_1^{-1} for figure 1, using four-point method.
- Compute the matrix H_2^{-1} for figure 2, using four-point method.
- Calculate the matrix H using following equation:

$$H = H_2^{-1} * (H_1^{-1})^{-1} \tag{3}$$

- Calculate the corresponding cost and model, the error is computed as the squared Sampson error.
- If the cost is less then current minimum cost, do the follows:
  - Update the minimum cost and model.
  - Calculate the number inliers and update the max trials using parameter w, where $w = \frac{number of inliers}{total number of data points}$, and $max\_trials = \frac{log(1-P)}{log(1-w^s)}$, where $s = 4$ since we have three random points for each iteration.
- If current number of trials is bigger than the current max trials, then stop the iteration. Otherwise, continue.

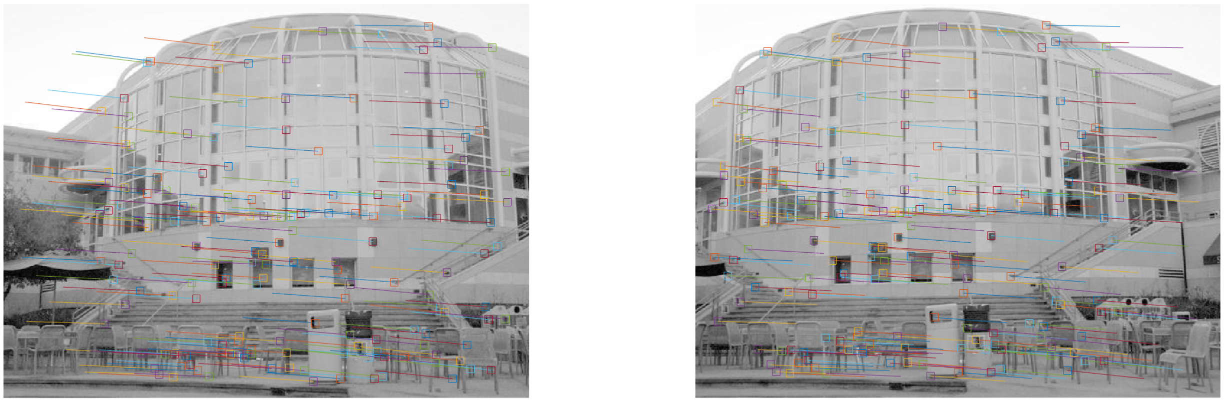

Figure 4: Feature mapping using inliers

- After the iteration, we can get the inliers by compare the error of each point with the tolerance, using the model we get from the iteration.

The matching figures are shown in Figure 4.

4 Linear estimation

Estimate the planar projective transformation H_{DLT} from the resulting set of inlier correspondences using the direct linear transformation (DLT) algorithm (with data normalization). You must express $\mathbf{x\prime}_i = \mathbf{H}\mathbf{x}_i$ as $[\mathbf{x\prime}]_i^{\perp}\mathbf{H}\mathbf{x}_i = 0$ (not $\mathbf{x\prime}_i \times \mathbf{H}\mathbf{x}_i = 0$), where $[\mathbf{x\prime}]_i^{\perp}\mathbf{x\prime}_i = 0$, when forming the solution. Include the numerical values of the resulting P_{DLT}, scaled such that $\| H_{DLT} \|_{Fro} = 1$, in your report with sufficient precision such that it can be evaluated (hint: use format longg in MATLAB prior to displaying your results).

Solution:

The planar projective transformation H_{DLT} is as follows:

$$H_{DLT} = \begin{bmatrix} -0.0106540525548768 & -0.000322565823387382 & 0.175433007787968 \\ 1.50813373020026e-05 & -0.0109447323490494 & 0.984320115185873 \\ -1.18906447804917e-07 & -1.23974438128738e-06 & -0.0101867727565862 \end{bmatrix} \tag{4}$$

Here is the general process that I used to solve this problem:

- Do the data normalization for both figures.
- Then we can get the normalized data and matrix T_1 and T_2 for both figures respectively.
- Using Householder matrix to compute the left null space of each point in figure 2.
- Form the big matrix A using kron operations for each pair of points in both figures.
- Compute the vector form of matrix H, using SVD operation of matrix A.
- Reshape the vector form of matrix H, then we can get the matrix H_norm.
- Calculate the matrix H_{DLT}, using the following equation:

$$H_{DLT} = inv(T_2) * P_{norm} * T_1. \tag{5}$$

- Scale the matrix H_{DLT} so that its fro norm is equal to 1.

5 Nonlinear estimation

Use H_{DLT} and the Sampson corrected points (in image 1) as an initial estimate to an iterative estimation method, specifically the sparse Levenberg-Marquardt algorithm, to determine the Maximum Likelihood estimate of the planar projective transformation that minimizes the reprojection error. You must parameterize the planar projective transformation matrix and the homogeneous 2D scene points that are being adjusted using the parameterization of homogeneous vectors (see section A6.9.2 (page 624) of the textbook, and the corrections and errata).

In your report, show the initial cost (i.e., the cost at iteration 0) and the cost at the end of each successive iteration. Show the numerical values for the final estimate of the planar projective transformation matrix HLM, scaled such that $\parallel H_{LM} \parallel_{Fro}= 1$, in your report with sufficient precision such that it can be evaluated.

Solution:

The initial cost is 79.7774830417331.

And the cost of each successive iteration are as follows:

79.7774830417331
79.7579930299769
79.6976116725415
79.6966414533895
79.696641450062

The number of iterations that used to be convergence is 4.

The final estimate of the matrix is as follows:

$$H_{LM} = \begin{bmatrix} 0.0106660499793628 & 0.000320913162721748 & -0.174173928321309 \\ -1.92852986465492e-05 & 0.0109602143976192 & -0.984543171425123 \\ 1.05082564703225e-07 & 1.23568286685395e-06 & 0.0102074488860498 \end{bmatrix} \quad (6)$$

For this problem, the process of Levenberg algorithm (nonlinear estimation) is very clear according to the lecture note. And here is the general process:

- For initialization, we have initial matrix H_{DLT}, which is from the result of question (d), and the inlier points for both images, which are from the result of question (c).
- Set the initial value $\lambda = 0.001$.
- Compute the scene point using the first image and Sampson correction.
- Calculate initial measurement vector $[\hat{h}^T, \hat{x}_1^T, ..., \hat{x}_n^T]$, where $\hat{h}$ is the parameterization of the transformation matrix H, and $\hat{x}_i$ are the parameterization of scene points.
- Calculate the projections using following equations, where $x - scene$ is the scene points, and H is the initial transformation matrix H_{DLT}.

$$proj1 = I * x_{scene} \quad (7)$$

$$proj2 = H * x_{scene} \quad (8)$$

- Compute the measurement vector $[x\prime_1^T, x\prime_2^t, ..., x\prime_n^T, x\prime\prime_1^T, x\prime\prime_2^T, ..., x\prime\prime_n^T]$.
- Compute the initial error vector and further compute the initial cost.
- Begin the iteration of Levenberg method, do the following until convergence.

- First, compute the Jacobian matrix using parameter vector and inlier points for both images. And the Jacobian matrix is a sparse matrix, which contains three parts: A, $B1$ and $B2$.
- Second, using the three parts of Jacobian matrix to compute the normal equations matrix, which also contains three parts: U, V, and W.
- Third, compute the normal equations vector, which is in the form of $(\epsilon_a^T, \epsilon_{b_1}^T, \epsilon_{b_2}^T, ..., \epsilon_{b_n}^T)^T$.
- Forth, compute the augmented normal equations, solve the equations and we can get the increment δ.
- Then, add δ to the parameter vector, we can get a new vector.
- Deparameterize the two parts of the new parameter vector, then we can get the new transformation matrix and the new scene points.
- Recompute the new error and cost, using the new transformation matrix and new scene points.
- If the current cost is less than the current minimum cost, then update the error, cost and model, and go to the first step, with setting $\lambda = 0.1\lambda$. Otherwise, go to the forth step by setting $\lambda = 10\lambda$.

- Finally, after convergence, we can get the result.

Appendix: Source code

```
% main.m

clc, clear, close all;
format longg;

% input image
image1 = 'price_center20.JPG';
image2 = 'price_center21.JPG';

figure(1)
subplot(1, 2, 1);
imshow(rgb2gray(imread(image1)));
subplot(1, 2, 2)
imshow(rgb2gray(imread(image2)));


%--------- Question (a) -----------%
% feature detection %

% set parameter
w_size1 = 9;
threshold = 380;
w_size2 = 9;
w_sizeb = 21;
simThresh = 0.6;
ratioThresh = 0.7;

% corner detection
[row1, col1] = featureDetection(image1, w_size1, threshold, w_size2);
[row2, col2] = featureDetection(image2, w_size1, threshold, w_size2);

% count # features
x1 = row1(:);
x2 = row2(:);
y1 = col1(:);
y2 = col2(:);
disp('Question (a):');
disp('number of features in figure 1:');
disp(size(x1, 1));
disp('number of features in figure 2:');
disp(size(x2, 1));

% show the feature image
figure(2)
subplot(1, 2, 1);
imshow(rgb2gray(imread(image1)));
hold on;
scatter(y1, x1, w_size1 * w_size1, 's');
subplot(1, 2, 2)
imshow(rgb2gray(imread(image2)));
hold on;
scatter(y2, x2, w_size1 * w_size1, 's');



%--------- Question (b) -----------%
% feature matching %

% feature matching
match = featureMatching(image1, image2, row1, col1, row2, col2, w_sizeb, simThresh,
    ratioThresh);
disp('Question (b):');
disp('number of matchings:');
```

```
disp(sum(sum(match)));

% show the feature matching image
figure(3)
subplot(1, 2, 1);
imshow(rgb2gray(imread(image1)));
hold on;
length1 = size(row1);
length2 = size(row2);
for i = 1 : length1(1)
    for j = 1 : length2(1)
        if match(i, j) == 1
            plot([col1(i), col2(j)], [row1(i), row2(j)], '-');
            scatter(col1(i), row1(i), w_size1 * w_size1, 's');
        end
    end
end
subplot(1, 2, 2);
imshow(rgb2gray(imread(image2)));
hold on;
length1 = size(row1);
length2 = size(row2);
for i = 1 : length1(1)
    for j = 1 : length2(1)
        if match(i, j) == 1
            plot([col1(i), col2(j)], [row1(i), row2(j)], '-');
            scatter(col2(j), row2(j), w_size1 * w_size1, 's');
        end
    end
end


%---------- Question (c) ------------%
% outliers rejection %

% extract coordinates of matching in question (b)
point2DOrig1 = [];
point2DOrig2 = [];
for i = 1 : size(row1, 1)
    for j = 1 : size(row2, 1)
        if match(i, j) == 1
            point2DOrig1 = [point2DOrig1; [row1(i), col1(i)]];
            point2DOrig2 = [point2DOrig2; [row2(j), col2(j)]];
        end
    end
end

% MSAC method
[inlierIndex, trials] = MSAC(point2DOrig1, point2DOrig2);
inlier1 = [];
inlier2 = [];
for i = 1 : length(inlierIndex)
    if inlierIndex(i) == 1
        inlier1 = [inlier1; point2DOrig1(i, :)];
        inlier2 = [inlier2; point2DOrig2(i, :)];
    end
end
disp('Question (c):');
disp('number of inliers:');
disp(sum(inlierIndex));
disp('number of trials:');
disp(trials);

% show the feature matching image
figure(4)
```

```
subplot(1, 2, 1);
imshow(rgb2gray(imread(image1)));
hold on;
for i = 1 : length(inlierIndex)
    if inlierIndex(i) == 1
        plot([point2DOrig1(i, 2), point2DOrig2(i, 2)], ...
             [point2DOrig1(i, 1), point2DOrig2(i, 1)], '-');
        scatter(point2DOrig1(i, 2), point2DOrig1(i, 1), w_size1 * w_size1, 's');
    end
end
subplot(1, 2, 2);
imshow(rgb2gray(imread(image2)));
hold on;
for i = 1 : length(inlierIndex)
    if inlierIndex(i) == 1
        plot([point2DOrig2(i, 2), point2DOrig1(i, 2)], ...
             [point2DOrig2(i, 1), point2DOrig1(i, 1)], '-');
        scatter(point2DOrig2(i, 2), point2DOrig2(i, 1), w_size1 * w_size1, 's');
    end
end



%------------ Question (d) ------------%
% linear estimation %

[H_norm, T2, T1] = linearEstimation(inlier2, inlier1);
format longg;
H_norm = - H_norm;

% scale P with ||P||Fro = 1
H = T2 \ H_norm * T1;
H = H / norm(H, 'fro');
H_DLT = - H;
disp('Question (d):');
disp('H matrix DLT:');
disp(H_DLT);

uni = ones(size(inlier1, 1), 1);
xEst = -H_DLT * [inlier1, uni]';
paramW = xEst(3, :);
xEst = xEst ./ paramW;
% disp(xEst);



%------------- Question (e) ------------%
% nonlinear estimation %

H_init = -H_DLT;
H_LM = levenbergEst(H_init, inlier1, inlier2);
H_LM = H_LM / norm(H_LM, 'fro');
disp('Question (e):')
disp('H_LM:')
disp(H_LM)
```

```
% featureDetection.m

% feature detection
function [row, col] = featureDetection(image, w_size1, threshold, w_size2)
    % read the image in RGB format
    i = imread(image);

    % convert RGB to gray scale
    grayImage = rgb2gray(i);
```

```
    %grayImage = double(grayImage);

    % calculate gradient images
    K = [-1, 8, 0, -8, 1] / 12;
    Ix = imfilter(grayImage, K);
    Iy = imfilter(grayImage, K');

    % calculate Isquare and IxIy
    IxSquare = Ix .* Ix;
    IxIy = Ix .* Iy;
    IySquare = Iy .* Iy;

    % calculate minor eigenvalue image
    eigenImage = calEigenImage(IxSquare, IxIy, IySquare, w_size1);

    % set 0 if below threshold
    threshEigenImage = eigenImage .* (eigenImage >= threshold);

    % non maximum suppression
    % maximum filter
    Imax = ordfilt2(threshEigenImage, w_size2 * w_size2, ones(w_size2, w_size2));
    % compare two image, generate image J
    imageJ = threshEigenImage .* (threshEigenImage >= Imax);

    % find the coordinate of corner
    [row, col] = findCorner(IxSquare, IxIy, IySquare, imageJ, w_size1);
end

% calculate minor eigenvalue image
function m = calEigenImage(IxSquare, IxIy, IySquare, w_size)
    len = size(IxIy);
    m = zeros(len(1), len(2));
    for i = 1 : len(1)
        for j = 1 : len(2)
            [N, b] = calGradMatrix(IxSquare, IxIy, IySquare, i, j, w_size);
            m(i, j) = 0.5 * (trace(N) - sqrt(trace(N) ^ 2 - 4 * det(N)));
        end
    end
end

% Calculate gradient matrix
function [m, b] = calGradMatrix(IxSquare, IxIy, IySquare, i, j, w_size)
    m = zeros(2, 2);
    b = zeros(2, 1);
    half = (w_size - 1) / 2;
    len = size(IxIy);
    i_start = max(i - half, 1);
    j_start = max(j - half, 1);
    i_end = min(i + half, len(1));
    j_end = min(j + half, len(2));
    m(1, 1) = sum(sum(IxSquare(i_start : i_end, j_start : j_end)));
    m(1, 2) = sum(sum(IxIy(i_start : i_end, j_start : j_end)));
    m(2, 1) = m(1, 2);
    m(2, 2) = sum(sum(IySquare(i_start : i_end, j_start : j_end)));
    for p = i_start : i_end
        for q = j_start : j_end
            b(1) = b(1) + double(p) * double(IxSquare(p, q)) + double(q) * double(IxIy(p, q));
            b(2) = b(2) + double(q) * double(IySquare(p, q)) + double(p) * double(IxIy(p, q));
        end
    end
end

% find corner coordinates
```

```
function [row, col] = findCorner(IxSquare, IxIy, IySquare, imageJ, w_size)
    len = size(imageJ);
    row = [];
    col = [];
    for i = 1 : len(1)
        for j = 1 : len(2)
            if (imageJ(i, j) > 0)
                [N, b] = calGradMatrix(IxSquare, IxIy, IySquare, i, j, w_size);
                coord = N \ b;
                coord = coord';
                if coord(1) >= 15 && coord(1) <= len(1) - 15 && ...
                   coord(2) >= 15 && coord(2) <= len(2) - 15
                    row = [row; coord(1)];
                    col = [col; coord(2)];
                end
            end
        end
    end
end

% extract the corner coordinates
function [row, col] = extractCorner(corner)
    row = [];
    col = [];
    for i = 15 : size(corner, 1) - 15
        for j = 15 : size(corner, 2) - 15
            if corner(i, j) == 1
                row = [row, i];
                col = [col, j];
            end
        end
    end
    row = row';
    col = col';
end
```

```
% featureMatching.m

% feature matching
function match = featureMatching(image1, image2, row1, col1, row2, col2, w_size, simThresh,
    ratioThresh)
    I1 = imread(image1);
    I2 = imread(image2);
    Image1 = rgb2gray(I1);
    Image2 = rgb2gray(I2);
    length1 = size(row1);
    len1 = length1(1);
    length2 = size(row2);
    len2 = length2(1);
    match = zeros(len1, len2);
    % calculate correlation coefficient matrix
    correl = zeros(len1, len2);
    for i = 1 : len1
        [win1, size1] = fetchWindow(Image1, size(Image1), row1, col1, i, w_size);
        for j = 1 : len2
            [win2, size2] = fetchWindow(Image2, size(Image2), row2, col2, j, w_size);
            if size1 == w_size^2 && size2 == w_size^2
                correl(i, j) = corr2(win1, win2);
            else
                correl(i,j) = -1.0;
            end
        end
    end
    % one to one match
    maxValue = max(max(correl));
```

```
    count = 0;
    while maxValue > simThresh
        [X, Y] = find(correl == maxValue);
        x = X(1);
        y = Y(1);
        correl(x, y) = -1;
        nextMaxValue = max(max(correl(x,:)), max(correl(:,y)));
        if (1.0 - maxValue) < (1.0 - nextMaxValue) * ratioThresh
            count = count + 1;
            match(x, y) = 1;
        end;
        correl(x,:) = -1;
        correl(:,y) = -1;
        maxValue = max(max(correl));
    end
end

% fetch the window
function [win, size] = fetchWindow(image, len, row, col, i, w_size)
    half = (w_size - 1) / 2;
    i_start = max(round(row(i)) - half, 1);
    j_start = max(round(col(i)) - half, 1);
    i_end = min(round(row(i)) + half, len(1));
    j_end = min(round(col(i)) + half, len(2));
    win = image(i_start : i_end, j_start : j_end);
    size = (i_end - i_start + 1) * (j_end - j_start + 1);
end
```

```
% MSAC.m

% MSAC method
function [inlierIndex, trials] = MSAC(point2DOrig1, point2DOrig2)
    format longg;
    % transfer to homogeneous
    num_point = size(point2DOrig1, 1);
    point2DHomo1 = [point2DOrig1, ones(num_point, 1)];
    point2DHomo2 = [point2DOrig2, ones(num_point, 1)];

    % MSAC algorithm
    consensus_min_cost = Inf;
    trials = 0;
    max_trials = Inf;
    threshold = 0;
    prob = 0.99;
    alpha = 0.95;
    variance = 1;
    codimension = 2;
    tolerance = chi2inv(alpha, codimension);

    rand('seed', 0);

    % begin iteration
    while (trials < max_trials && consensus_min_cost > threshold)
        % generate three random samples
        sampleIndex = randperm(num_point, 4);
        % compute model H
        H1_inv = fourPoint(point2DHomo1, sampleIndex);
        H2_inv = fourPoint(point2DHomo2, sampleIndex);
        H = H2_inv / H1_inv;
        % compute cost
        cost = 0;
        for i = 1 : num_point
            % compute sampson error
            error_i = sampsonError(H, point2DHomo1(i, :), point2DHomo2(i, :));
            if error_i < tolerance
```

```
                cost = cost + error_i;
            else
                cost = cost + tolerance;
            end
        end
        % update model
        if cost < consensus_min_cost
            consensus_min_cost = cost;
            model_H = H;
            % count number of inliers
            dist_error = zeros(1, num_point);
            for i = 1 : num_point
                dist_error(i) = sampsonError(model_H, point2DHomo1(i, :), point2DHomo2(i, :)
    );
            end
            % update max_trials
            num_inliers = sum(dist_error <= tolerance);
            w = num_inliers / num_point;
            max_trials = log(1 - prob) / log(1 - w^4);
        end
            trials = trials + 1;
    end
    % count inliers
    dist_error = zeros(1, num_point);
    for i = 1 : num_point
        dist_error(i) = sampsonError(model_H, point2DHomo1(i, :), point2DHomo2(i, :));
    end
    inlierIndex = (dist_error <= tolerance);
end

% four point method for 2D projective transformation
function H_inv = fourPoint(pointHomo, sampleIndex)
    format longg;
    point1 = pointHomo(sampleIndex(1), :)';
    point2 = pointHomo(sampleIndex(2), :)';
    point3 = pointHomo(sampleIndex(3), :)';
    point4 = pointHomo(sampleIndex(4), :)';
    part1 = [point1, point2, point3];
    lam = part1 \ point4;
    H_inv = [lam(1) * point1, lam(2) * point2, lam(3) * point3];
end

% calculate Sampson error %
function error = sampsonError(H, point1, point2)
    % compute epsilon (Ah) and J
    point1 = point1';
    point2 = point2';
    epsilon = [-point1' * H(2, :)' + point2(2) * point1' * H(3, :)'; ...
                point1' * H(1, :)' - point2(1) * point1' * H(3, :)'];
    J = [-H(2, 1) + point2(2) * H(3, 1), -H(2, 2) + point2(2) * H(3, 2), ...
          0, point1(1) * H(3, 1) + point1(2) * H(3, 2) + H(3, 3); ...
          H(1, 1) - point2(1) * H(3, 1), H(1, 2) - point2(1) * H(3, 2), ...
          -(point1(1) * H(3, 1) + point1(2) * H(3, 2) + H(3, 3)), 0];
    % compute sampson error
    error = epsilon' / (J * J') * epsilon;
end
```

```
% linearEstimation.m

% DLT linear estimation %
function [H_norm, T1, T2] = linearEstimation(inlier1, inlier2)
    % data normalization
    [point1, T1] = dataNormalization(inlier1, 2);
    [point2, T2] = dataNormalization(inlier2, 2);

```

```
    % using DLT compute matrix A
    % H * point2 = point1
    A = DLTAlgorithm(point1, point2);

    % using svd compute projection matrix P
    H_norm = calProjMat(A);
end

% Data Normalization
function [point, T] = dataNormalization(data, dim)
    num_point = size(data, 1);
    % calculate mean and variance
    m = mean(data);
    v = var(data);
    % calculate normalized vector
    if (dim == 2)
        s = sqrt(2.0 / sum(v));
        T = zeros(3, 3);
        T(1, 1) = s;
        T(2, 2) = s;
        T(3, 3) = 1.0;
        T(1, 3) = -1.0 * m(1) * s;
        T(2, 3) = -1.0 * m(2) * s;
    else
        s = sqrt(3.0 / sum(v));
        T = zeros(4, 4);
        T(1, 1) = s;
        T(2, 2) = s;
        T(3, 3) = s;
        T(4, 4) = 1.0;
        T(1, 4) = -1.0 * m(1) * s;
        T(2, 4) = -1.0 * m(2) * s;
        T(3, 4) = -1.0 * m(3) * s;
    end
    % transfer inhomo to homo data
    unit = ones(num_point, 1);
    point = [data, unit];
    % normalize the data
    point = T * point';
    point = point';
end

% DLT algorithm
function matA = DLTAlgorithm(point1, point2)
    num_point = size(point1, 1);
    point1 = point1';
    point2 = point2';
    matA = [];
    % using house holder matrix
    % to calculate left null space of x
    for i = 1 : num_point
        x = point1(:, i);
        v = x + sign(x(1)) * norm(x) * [1, 0, 0]';
        Hv = eye(3) - 2.0 * (v * v') / (v' * v);
        leftNull = Hv(2 : 3, :);
        matA = [matA; kron(leftNull, point2(:, i)')];
    end
end

% using svd compute projection matrix P
function P = calProjMat(A)
    [U, S, V] = svd(A);
    P = V(:, size(V, 2));
    P = reshape(P, [3, 3]);
    P = P';
```

```
end
```

```
% levenbergEst.m

function H_LM = levenbergEst(H_init, inlier1, inlier2)
    inlier1 = inlier1';
    inlier2 = inlier2';
    num_point = size(inlier1, 2);
    pntHomo1 = [inlier1; ones(1, num_point)];
    pntHomo2 = [inlier2; ones(1, num_point)];
    H = H_init;

    % compute scene point
    pnt_scene_deparam = zeros(2, num_point);
    pnt_scene_param = zeros(2, num_point);
    for i = 1 : num_point
       pnt = sampsonCorrection(H, pntHomo1(:, i)', pntHomo2(:, i)');
       pnt = pnt';
       pnt_scene_deparam(:, i) = pnt;
       pnt = [pnt; 1] / norm(pnt);
       pnt_scene_param(:, i) = parameterize(pnt);
    end

    % compute initial cost
    error1 = inlier1 - pnt_scene_deparam;
    proj2 = zeros(3, num_point);
    proj2_homo = zeros(3, num_point);
    for i = 1:num_point
       proj2(:, i) = H * deparameterize(pnt_scene_param(:, i));
       proj2_homo(:, i) = proj2(:, i)/proj2(3, i);
    end
    proj2_inhomo = proj2_homo(1:2, :);
    error2 = inlier2 - proj2_inhomo;
    cost = norm(error1(:))^2 + norm(error2(:))^2;
    disp('cost for each iteration:');
    disp(cost);

    h = H';
    h = h(:);
    paramH = parameterize(h);
    lam = 0.001;
    n = 1;

    while 1
        % compute Jacobian matrix
        [A, B1, B2] = calJacobian(num_point, proj2, pnt_scene_param, H, paramH);

        % compute normal equations matrix
        U = zeros(8, 8);
        V = zeros(2, 2, num_point);
        W = zeros(8, 2, num_point);
        error_part1 = zeros(8, 1);
        error_part2 = zeros(2, 1, num_point);
        for i = 1:num_point
            U = U + A(:, :, i)' * A(:, :, i);
            V(:, :, i) = B1(:, :, i)' * B1(:, :, i) + ...
                         B2(:, :, i)' * B2(:, :, i);
            W(:, :, i) = A(:, :, i)' * B2(:, :, i);
            error_part1 = error_part1 + A(:, :, i)' * ...
                          error2(:, i);
            error_part2(:, :, i) = B1(:, :, i)' * error1(:, i) + ...
                                   B2(:, :, i)' * error2(:, i);
        end

        % compute augmented normal euqations
```

```
        S = U + lam * eye(8);
        epsilon = error_part1;
        for i = 1 : num_point
            S = S - W(:, :, i) * inv(V(:, :, i) + lam * eye(2)) * ...
                W(:, :, i)';
            epsilon = epsilon - W(:, :, i) * inv(V(:, :, i) + lam * eye(2)) * ...
                      error_part2(:, :, i);
        end
        delata_part1 = linsolve(S, epsilon);
        delata_part2 = zeros(2, 1, num_point);
        for i = 1 : num_point
            delata_part2(:, :, i) = inv(V(:, :, i) + lam * eye(2)) * ...
                                    (error_part2(:, :, i) - W(:, :, i)' * ...
                                    delata_part1);
        end

        % update
        paramH_update = paramH + delata_part1;
        pnt_scene_param_update = zeros(2, num_point);
        for i = 1 : num_point
            pnt_scene_param_update(:, i) = pnt_scene_param(:, i) + ...
                                           delata_part2(:, :, i);
        end
        h_update = deparameterize(paramH_update);
        deparamH_update = reshape(h_update, 3, 3);
        deparamH_update = deparamH_update';
        proj1_update = zeros(2, num_point);
        for i = 1 : num_point
            pnt = deparameterize(pnt_scene_param_update(:, i));
            pnt = pnt / pnt(3);
            proj1_update(:, i) = pnt(1 : 2);
        end

        % compute new error
        error1_update = inlier1 - proj1_update;
        proj2_inhomo_update = zeros(3, num_point);
        for i = 1 : num_point
            proj2_update(:, i) = deparamH_update * ...
                                 deparameterize(pnt_scene_param_update(:, i));
            proj2_inhomo_update(:, i) = proj2_update(:, i) / proj2_update(3, i);
        end
        proj2_inhomo_update = proj2_inhomo_update(1 : 2, :);
        error2_update = inlier2 - proj2_inhomo_update;
        cost_update = norm(error2_update(:))^2 + norm(error1_update(:))^2;

        % jump the loop
        if(cost_update > cost)
            lam = 10 * lam;
        else
            disp(cost_update);
            if((cost - cost_update) / cost < 1e-8)
                break;
            end
            n = n + 1;
            lam = lam / 10;
            paramH = paramH_update;
            H = deparamH_update;
            error1 = error1_update;
            error2 = error2_update;
            cost = cost_update;
            pnt_scene_param = pnt_scene_param_update;
            proj2 = proj2_update;
        end
    end
    disp('number of iterations:');
```

```
    disp(n);
    H_LM = H;
end

% parameterize
function paramVector = parameterize(P)
    a = P(1);
    b = P(2 : length(P));
    paramVector = (2.0 / (sinc(acos(a)))) * b;
    normP = norm(paramVector);
    if (normP > pi)
        paramVector = (1.0 - 2 * pi / normP * ceil((normP - pi) / 2 * pi)) * paramVector;
    end
end

% deparameterize
function deparamVector = deparameterize(P)
    normP = norm(P);
    deparamVector = [cos(normP / 2.0), ((sinc(normP / 2.0)) / 2.0) * P']';
end

% sinc(x)
function res = sinc(x)
    if x == 0
        res = 1.0;
    else
        res = (sin(x)) / x;
    end
end

% derivative of sinc(x)
function res = derivSinc(x)
    if x == 0
        res = 0.0;
    else
        res = cos(x) / x - sin(x) / (x * x);
    end
end

% calculate Sampson error %
function point = sampsonCorrection(H, point1, point2)
    x_hat = [point1(1 : 2), point2(1 : 2)]';
    point1 = point1';
    point2 = point2';
    % compute epsilon (Ah) and J
    epsilon = [-point1' * H(2, :)' + point2(2) * point1' * H(3, :)'; ...
               point1' * H(1, :)' - point2(1) * point1' * H(3, :)'];
    J = [-H(2, 1) + point2(2) * H(3, 1), -H(2, 2) + point2(2) * H(3, 2), ...
          0, point1(1) * H(3, 1) + point1(2) * H(3, 2) + H(3, 3); ...
          H(1, 1) - point2(1) * H(3, 1), H(1, 2) - point2(1) * H(3, 2), ...
          -(point1(1) * H(3, 1) + point1(2) * H(3, 2) + H(3, 3)), 0];
    % compute sampson error
    lam = (J * J') \ (-epsilon);
    delta = J' * lam;
    x_hat = x_hat + delta;
    point = x_hat(1 : 2)';
end

% compute jocabian matrix
function [A, B1, B2] = calJacobian(num_point, proj2, pnt_scene_param, H, paramH)
    A = zeros(2, 8, num_point);
    B1 = zeros(2, 2, num_point);
    B2 = zeros(2, 2, num_point);
    for i = 1:num_point
        pnt = proj2(:, i);
```

```
        pnt_scene = deparameterize(pnt_scene_param(:, i));

        % compute A
        A1_i = [1 / pnt(3), 0, -pnt(1) / pnt(3)^2;
                0, 1 / pnt(3), -pnt(2) / pnt(3)^2];
        A2_i = zeros(2,9);
        A2_i(1,1:3) = pnt_scene';
        A2_i(2,4:6) = pnt_scene';
        A2_i(3,7:9) = pnt_scene';

        part_H = zeros(9, 8);
        part_H(1, :) = -0.25 * (sinc(norm(norm(paramH) / 2))) * paramH';


        part_H(2:9, :) = sinc(norm(paramH) / 2) * 0.5 * eye(8) + ...
                         (1 / (4 * norm(paramH))) * ...
                         (derivSinc(norm(paramH) / 2)) * paramH * paramH';

        A_i = A1_i * A2_i * part_H;
        A(:, :, i) = A_i;

        % Compute B1
        B1_1_i = [1 / pnt_scene(3), 0, -pnt_scene(1) / pnt_scene(3)^2;
                  0, 1 / pnt_scene(3), -pnt_scene(2) / pnt_scene(3)^2];

        B1_2_i = zeros(3, 2);
        B1_2_i(1, :) = -0.25 * (sinc(norm(pnt_scene_param(:, i)) / 2)) * ...
                       pnt_scene_param(:, i)';
        B1_2_i(2 : 3, :) = sinc(norm(pnt_scene_param(:, i)) / 2) * 0.5 * ...
                           eye(2) + (1/(4 * norm(pnt_scene_param(:, i)))) * ...
                           (derivSinc(norm(pnt_scene_param(:, i)) / 2)) * ...
                           pnt_scene_param(:, i) * pnt_scene_param(:, i)';

        B1_i = B1_1_i * B1_2_i;
        B1(:, :, i) = B1_i;

        % compute B2
        B2_1_i = [1 / pnt(3), 0, -pnt(1) / pnt(3)^2;
                  0, 1 / pnt(3), -pnt(2) / pnt(3)^2];

        B2_2_i = H;

        B2_3_i = zeros(3,2);
        B2_3_i(1, :) = -0.25 * (sinc(norm(pnt_scene_param(:, i)) / 2)) * ...
                      pnt_scene_param(:, i)';
        B2_3_i(2 : 3, :) = sinc(norm(pnt_scene_param(:, i)) / 2) * 0.5 * ...
                           eye(2) + (1/(4 * norm(pnt_scene_param(:, i)))) * ...
                           (derivSinc(norm(pnt_scene_param(:, i)) / 2)) * ...
                           pnt_scene_param(:, i) * pnt_scene_param(:, i)';
        B2_i = B2_1_i * B2_2_i * B2_3_i;
        B2(:, :, i) = B2_i;
    end
end
```