CSE 252B - Homework 1

Lingting Ge A53212800

1/23/2017

1 Line-plane intersection

The line in 3D defined by the join of the points $\mathbf{X_1} = (X_1, Y_1, Z_1, T_1)^{\mathrm{T}}$ and $\mathbf{X_2} = (X_2, Y_2, Z_2, T_2)^{\mathrm{T}}$ can be represented as a Plucker matrix $L = \mathbf{X_1}\mathbf{X_2}^{\mathrm{T}} - \mathbf{X_2}\mathbf{X_1}^{\mathrm{T}}$ or pencil of points $\mathbf{X}(\lambda) = \lambda\mathbf{X_1} + (1-\lambda)\mathbf{X_2}$ (i.e., $\mathbf{X}$ is a function of λ). The line intersects the plane $\boldsymbol{\pi} = (a, b, c, d)^{\mathrm{T}}$ at the point $\mathbf{X}_L = L\boldsymbol{\pi}$ or $\mathbf{X}(\lambda_\pi)$, where λ_π is the determined such that $\mathbf{X}(\lambda_\pi)^{\mathrm{T}}\boldsymbol{\pi} = 0$ (i.e., $\mathbf{X}(\lambda_\pi)$ is the point on $\boldsymbol{\pi}$).Show that $\mathbf{X}_L$ is equal to $\mathbf{X}(\lambda_\pi)$ up to scale.

Proof:

$$\begin{aligned}\mathbf{X}_L &= L\boldsymbol{\pi} \\ &= (\mathbf{X_1}\mathbf{X_2}^{\mathrm{T}} - \mathbf{X_2}\mathbf{X_1}^{\mathrm{T}})\boldsymbol{\pi} \\ &= \mathbf{X_1}(\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}) - \mathbf{X_2}(\mathbf{X_1}^{\mathrm{T}}\boldsymbol{\pi}) \end{aligned} \tag{1}$$

$$\begin{aligned}\mathbf{X}(\lambda_\pi)^{\mathrm{T}}\boldsymbol{\pi} &= 0 \\ (\lambda_\pi\mathbf{X_1} + (1-\lambda_\pi)\mathbf{X_2})^{\mathrm{T}}\boldsymbol{\pi} &= 0 \\ \lambda_\pi(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi} &= \mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi} \\ \lambda_\pi &= \frac{\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}} \end{aligned} \tag{2}$$

$$\begin{aligned}\mathbf{X}(\lambda_\pi) &= \frac{\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}}\mathbf{X_1} - (1 - \frac{\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}})\mathbf{X_2} \\ &= \frac{\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}}\mathbf{X_1} - \frac{\mathbf{X_1}^{\mathrm{T}}\boldsymbol{\pi}}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}}\mathbf{X_2} \\ &= \frac{1}{(\mathbf{X_2} - \mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}}[\mathbf{X_1}(\mathbf{X_2}^{\mathrm{T}}\boldsymbol{\pi}) - \mathbf{X_2}(\mathbf{X_1}^{\mathrm{T}}\boldsymbol{\pi})] \end{aligned} \tag{3}$$

Since $\frac{1}{(\mathbf{X_2}-\mathbf{X_1})^{\mathrm{T}}\boldsymbol{\pi}}$ is a constant scalar, compare the equation (1) and (3), we can find that $\mathbf{X}_L$ is equal to $\mathbf{X}(\lambda_\pi)$ up to scale.

2 Line-quadric intersection

In general, a line in 3D intersections a quadric Q at zero, one (if the line is tangent to the quadric), or two points. If the pencil of points $\mathbf{X}(\lambda) = \lambda\mathbf{X_1} + (1-\lambda)\mathbf{X_2}$ represents a line in 3D, the (up to two) real roots of the quadratic polynomial $c_2\lambda_Q^2 + c_1\lambda_Q + c_0 = 0$ are used to solve for the intersection point(s) $\mathbf{X}(\lambda_Q)$. Show that $c_2 = \mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} - 2\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} + \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2}$, $c_1 = 2(\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} - \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2})$, and $c_0 = \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2}$.

Proof:

From the information given by the problem, we have:

$$
\begin{aligned}
&\mathbf{X}(\lambda_Q)^{\mathrm{T}}\mathrm{Q}\mathbf{X}(\lambda_Q) = 0 \\
&[\lambda_Q\mathbf{X_1} + (1-\lambda_Q)\mathbf{X_2}]^{\mathrm{T}}\mathrm{Q}\lambda_Q\mathbf{X_1} + (1-\lambda_Q)\mathbf{X_2} = 0 \\
&(\lambda_Q\mathbf{X_1}^{\mathrm{T}} + \mathbf{X_2}^{\mathrm{T}} - \lambda_Q\mathbf{X_2}^{\mathrm{T}})\mathrm{Q}(\lambda_Q\mathbf{X_1} + \mathbf{X_2} - \lambda_Q\mathbf{X_2}) = 0 \\
&(\lambda_Q\mathbf{X_1}^{\mathrm{T}}\mathrm{Q} + \mathbf{X_2}^{\mathrm{T}}\mathrm{Q} - \lambda_Q\mathbf{X_2}^{\mathrm{T}}\mathrm{Q})(\lambda_Q\mathbf{X_1} + \mathbf{X_2} - \lambda_Q\mathbf{X_2}) = 0 \\
&\lambda_Q^2\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} + \lambda_Q\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} - \lambda_Q^2\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} + \lambda_Q\mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} + \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} \\
&- \lambda_Q\mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} - \lambda_Q^2\mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} - \lambda_Q\mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} + \lambda_Q^2\mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} = 0 \\
&(\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} - 2\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} + \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2})\lambda_Q^2 + 2(\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} - \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2})\lambda_Q \\
&+ \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} = 0
\end{aligned} \tag{4}
$$

Therefore, from equation (4), we can see that $c_2 = \mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_1} - 2\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} + \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2}$, $c_1 = 2(\mathbf{X_1}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2} - \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2})$, and $c_0 = \mathbf{X_2}^{\mathrm{T}}\mathrm{Q}\mathbf{X_2}$.

3 Automatic feature detection and matching

3.1 Feature detection

Download input data from the course website. The file $price_center20.JPG$ contains image1 and the file $price_center21.JPG$ 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

$$\mathrm{N} = \begin{bmatrix} \Sigma_w I_x^2 & \Sigma_w I_x I_y \\ \Sigma_w I_x I_y & \Sigma_w I_y^2 \end{bmatrix} \tag{5}$$

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.

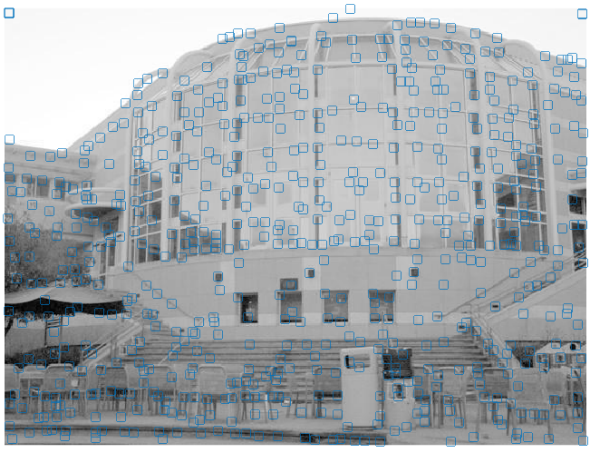
Solution:

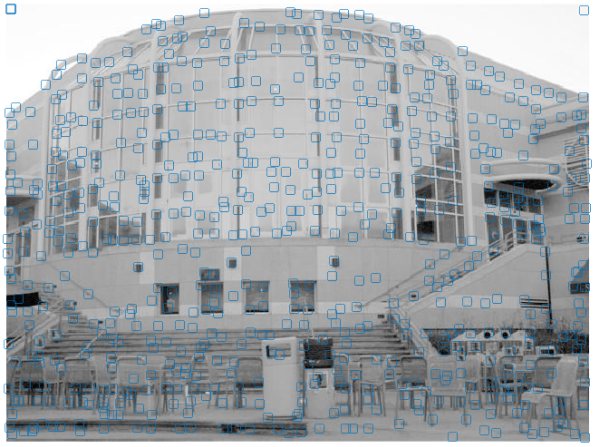
The parameters I used are as follows: size of the feature detection window is 9 * 9; The minor eigenvalue threshold value is 60; the size of the local nonmaximum suppression window is 11 * 11.

The resulting numbers of features detected are 638 and 648 for the first and second image respectively.

Here is the procedure that I solve this problem: First, read the image and transform the image into gray scale image; Second, calculate gradient images I_x and I_y respectively; Third, select a window size and calculate gradient matrix, and further calculate the minor eigenvalue image; Then, using a threshold to filter the minor eigenvalue image; Finally, do the non maxima suppression and using Forstner corner point method to find the coordinates of the corners. The source code are at the end of this paper.

The detected features are shown as follows.





3.2 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).

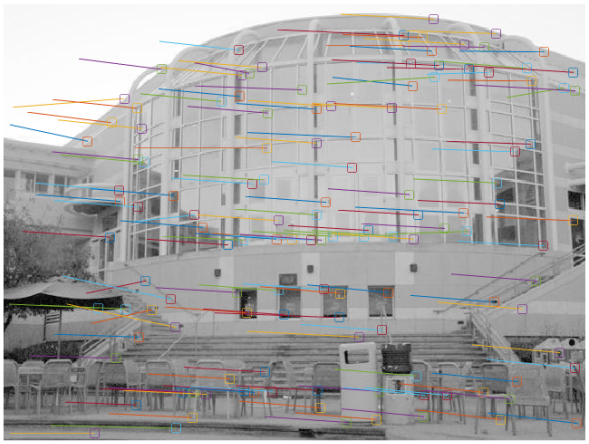
Solution:

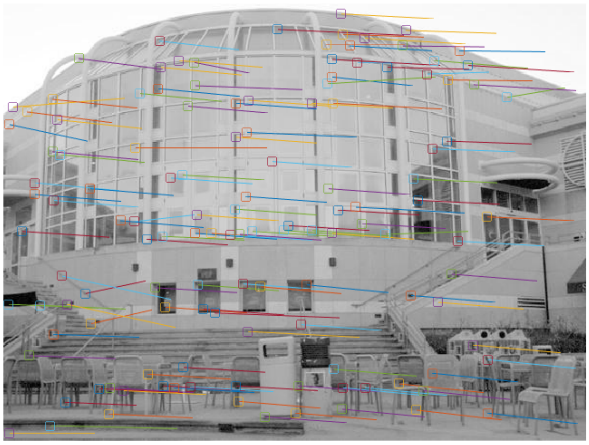
The parameters I used are as follows: the correlation coefficient threshold value is 0.5; the distance ratio threshold value is 0.825; And the size of the proximity is 150 and 50, where I used the Euclidean Distance,

which means I only consider the matches that are within the Euclidean Distance between 50 and 150.

The resulting number of putative feature correspondencs is 226.

The final feature matching images are shown as follows.





Appendix: Source code for problem 3

```
% main.m

% This function is going to detect the %
% corner of two images and match them  %

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

% set parameter
w_size1 = 9;
threshold = 60;
w_size2 = 11;
simThresh = 0.5;
ratioThresh = 0.825;

% 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(size(x1));
disp(size(x2));

% show the feature image
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');

% feature matching
match = featureMatching(image1, image2, row1, col1, row2, col2, w_size2, simThresh,
    ratioThresh);
disp(sum(sum(match)));

% show the feature matching image
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
```

```
% 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');
```

```
16
17      % calculate Isquare and IxIy
18      IxSquare = Ix .* Ix;
19      IxIy = Ix .* Iy;
20      IySquare = Iy .* Iy;
21
22      % calculate minor eigenvalue image
23      eigenImage = calEigenImage(IxSquare, IxIy, IySquare, w_size1);
24
25      % set 0 if below threshold
26      threshEigenImage = eigenImage .* (eigenImage >= threshold);
27
28      % non maximum suppression
29      % maximum filter
30      Imax = ordfilt2(threshEigenImage, w_size2 * w_size2, ones(w_size2, w_size2));
31      % compare two image, generate image J
32      imageJ = threshEigenImage .* (threshEigenImage >= Imax);
33
34      % find the coordinate of nonzero elements
35      %[row, col] = find(imageJ);
36
37      % find the coordinate of corner
38      corner = findCorner(IxSquare, IxIy, IySquare, imageJ, w_size1);
39      [row, col] = find(corner);
40
41  % calculate minor eigenvalue image
42  function m = calEigenImage(IxSquare, IxIy, IySquare, w_size)
43      len = size(IxIy);
44      m = zeros(len(1), len(2));
45      for i = 1 : len(1)
46          for j = 1 : len(2)
47              [N, b] = calGradMatrix(IxSquare, IxIy, IySquare, i, j, w_size);
48              m(i, j) = 0.5 * (trace(N) - sqrt(trace(N) ^ 2 - 4 * det(N)));
49          end
50      end
51
52  % Calculate gradient matrix
53  function [m, b] = calGradMatrix(IxSquare, IxIy, IySquare, i, j, w_size)
54      m = zeros(2, 2);
55      b = zeros(2, 1);
56      half = (w_size - 1) / 2;
57      len = size(IxIy);
58      i_start = max(i - half, 1);
59      j_start = max(j - half, 1);
60      i_end = min(i + half, len(1));
61      j_end = min(j + half, len(2));
62      m(1, 1) = sum(sum(IxSquare(i_start : i_end, j_start : j_end)));
63      m(1, 2) = sum(sum(IxIy(i_start : i_end, j_start : j_end)));
64      m(2, 1) = m(1, 2);
65      m(2, 2) = sum(sum(IySquare(i_start : i_end, j_start : j_end)));
66      for p = i_start : i_end
67          for q = j_start : j_end
68              b(1) = b(1) + double(p) * double(IxSquare(p, q)) + double(q) * double(IxIy(p, q)
    );
69              b(2) = b(2) + double(q) * double(IySquare(p, q)) + double(p) * double(IxIy(p, q)
    );
70          end
71      end
72
73  % find corner coordinates
74  function m = findCorner(IxSquare, IxIy, IySquare, imageJ, w_size)
75      len = size(imageJ);
76      m = zeros(len(1), len(2));
77      for i = 1 : len(1)
78          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';
                m(round(coord(1)), round(coord(2))) = 1;
            end
        end
    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;
            % constrain to the proximity
            distance = sqrt((row1(x) - row2(y))^2 + (col1(x) - col2(y))^2);
            if (distance < 150 && distance > 50 && col1(x) > col2(y) && abs(row1(x) - row2(y
    )) < 30)
                match(x, y) = 1;
            end
        end;
        correl(x,:) = -1;
        correl(:,y) = -1;
        maxValue = max(max(correl));
    end
    disp(count);

% fetch the window
function [win, size] = fetchWindow(image, len, row, col, i, w_size)
    half = (w_size - 1) / 2;
    i_start = max(row(i) - half, 1);
    j_start = max(col(i) - half, 1);
```

```
    i_end = min(row(i) + half, len(1));
    j_end = min(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);
```