Computer Vision 252B

Hao-en Sung (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Abstract

This is the report for CSE 258 Hw1.

1 Line-plane intersection (5 points)

Since $X(\lambda_\pi)^T$ is the intersection point on the plane π, I have

$$0 = X(\lambda_\pi)^T \cdot \pi = \lambda_\pi X_1^T \pi + (1 - \lambda_\pi) X_2^T \pi \tag{1}$$

$$X_1^T \pi = \frac{\lambda_\pi - 1}{\lambda_\pi} X_2^T \pi. \tag{2}$$

I then can replace $X_1^T \pi$ into $L\pi$ as follows.

$$\begin{aligned}
L\pi &= X_1 X_2^T \pi - X_2 X_1^T \pi && (3)\\
&= X_1 X_2^T \pi - X_2 \cdot \frac{\lambda_\pi - 1}{\lambda_\pi} \cdot X_2^T \pi && (4)\\
&= \left(X_1 - \frac{\lambda_\pi - 1}{\lambda_\pi} \cdot X_2 \right) \cdot X_2^T \pi && (5)\\
&= \frac{X_2^T \pi}{\lambda_\pi} \cdot (\lambda_\pi X_1 + (1 - \lambda_\pi) X_2) && (6)\\
&= \frac{X_2^T \pi}{\frac{-X_2^T \pi}{X_1^T \pi - X_2^T \pi}} \cdot X(\lambda_\pi) && (7)\\
&= \left(X_2^T \pi - X_1^T \pi \right) \cdot X(\lambda_\pi) && (8)
\end{aligned}$$

It is obvious that $X_L = L\pi$ is equal to $X(\lambda_\pi)$ up to a scale factor $X_2^T \pi - X_1^T \pi$.

2 Line-quadric intersection (5 points)

Since λ_Q is one of the solutions for the intersection point of $X(L)$ and quadric Q, I know that $X(\lambda_Q)^T Q X(\lambda_Q) = 0$. Thus, I have

$$\begin{aligned}
0 &= \left[\lambda_Q X_1 + (1 - \lambda_Q) X_2\right]^T Q \left[\lambda_Q X_1 + (1 - \lambda_Q) X_2\right] && (9)\\
&= \lambda_Q^2 X_1^T Q X_1 + \lambda_Q (1 - \lambda_Q) X_2^T Q X_1 + \lambda_Q (1 - \lambda_Q) X_1^T Q X_2 + (1 - \lambda_Q)^2 X_2^T Q X_2 && (10)\\
&= \lambda_Q^2 \cdot \left(X_1^T Q X_1 - X_2^T Q X_1 - X_1^T Q X_2 + X_2^T Q X_2 \right) && (11)\\
&\quad + \lambda_Q \cdot \left(X_2^T Q X_1 + X_1^T Q X_2 - 2 X_2^T Q X_2 \right) + X_2^T Q X_2 .
\end{aligned}$$

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

Due to Q is a symmetric matrix, $X_2^TQX_1$ is the same as $X_1^TQX_2$, I can rewrite the last equation as

$$\begin{aligned}0 = \ & \lambda_Q^2 \cdot \left(X_1^TQX_1 - 2X_1^TQX_2 + X_2^TQX_2\right) \\ & + \lambda_Q \cdot 2\left(X_1^TQX_2 - X_2^TQX_2\right) + X_2^TQX_2\end{aligned} \quad . \tag{12}$$

Hence, the coefficients can be derived as follows.

$$c_2 = X_1^TQX_1 - 2X_1^TQX_2 + X_2^TQX_2 \tag{13}$$

$$c_1 = 2\left(X_1^TQX_2 - X_2^TQX_2\right) \tag{14}$$

$$c_0 = X_2^TQX_2 \tag{15}$$

3 Programming: Automatic feature detection and matching (35 points)

3.1 Feature detection (20 points)

The original pair of gray-scaled figures are shown in Fig. 1.

Figure 1: Original Picture

Firstly, I used built-in *conv2* function to calculate the gradient for x and y directions by convolute the original image with kernel vector

$$K = \frac{1}{12} \cdot [-1, 8, 0, -8, 1]\,. \tag{16}$$

Secondly, I detect all potential corners by solving the gradient matrix

$$N = \begin{bmatrix} \sum_w I_x^2 & \sum_w I_xI_y \\ \sum_w I_xI_y & \sum_w I_y^2 \end{bmatrix}, \tag{17}$$

with specified window size, which is 7 in my setting, and create $\lambda_{\min}$ matrix, where

$$\lambda_{\min} = \frac{\text{Trace}(N) - \sqrt{\text{Trace}(N)^2 - 4\text{det}(N)}}{2}. \tag{18}$$

Thirdly, to avoid too many clustered corner candidates, I utilize *Non-maximum Suppression* policy with window size 7 again to filter them out. Fourthly, I filter out corners whose λ is below *lambda threshold*, which is 2200 in my setting. At the end, I recenter the coordinate of corner candidates by solving the following equations.

$$\begin{bmatrix} \sum_w I_x^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y^2 \end{bmatrix} \cdot \begin{bmatrix} x_{\text{corner}} \\ y_{\text{corner}} \end{bmatrix} = \begin{bmatrix} \sum_w x I_x^2 + y I_x I_y \\ \sum_w x I_x I_y + y I_y^2 \end{bmatrix} \tag{19}$$

The corresponding code is implemented as Code 1 and run by Code 3 in Appendix.

The figure with detected corners are included as Fig. 2. It is noticeable that I detect 624 features in the first image and 649 features in the second image.

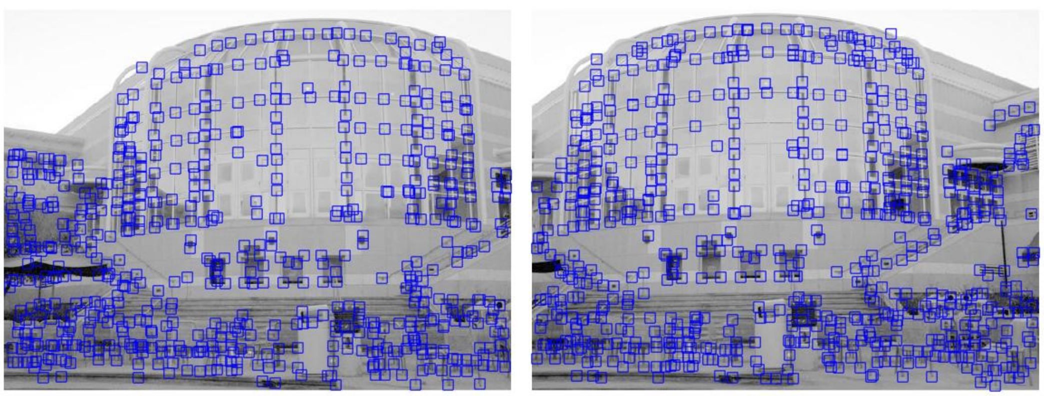

Figure 2: Detected Corner Picture

3.2 Feature matching (15 points)

Firstly, I build a matrix corrw, where $\text{corrw}_{i,j}$ is the correlation coefficient between window i in the first figure and window j in the second figure. Secondly, I use a brute force procedure, which is described as follows, to link up the corresponding corner pairs in two figures.

1. Iteratively find out the maximum correlation coefficient coordinate, i.e. best_v, throughout the whole table. If that value is smaller than a *similarity threshold*, which is 0.5 in my setting, stop iterative loop.
2. Find possible best matched corner in either images, i.e. sub_best_v. In implementation, I just search through the same row and same column of the point stated in first step. It is noticeable that I additionally put a proximal constraint on the matching process such that the

absolute distance of x and y for two matched pairs cannot exceed 100 pixels. On top of that, I will keep this feature match only if (1 - best_v) is smaller than *distance ratio threshold* times (1 - sub_best_v), which is 0.75 in my setting, . Otherwise, I will reject this match.

The corresponding code is implemented as Code 2 and run by Code 3 in Appendix.

The figure with detected corners are included as Fig. 3. It is noticeable that I finally find 143 paired corners.

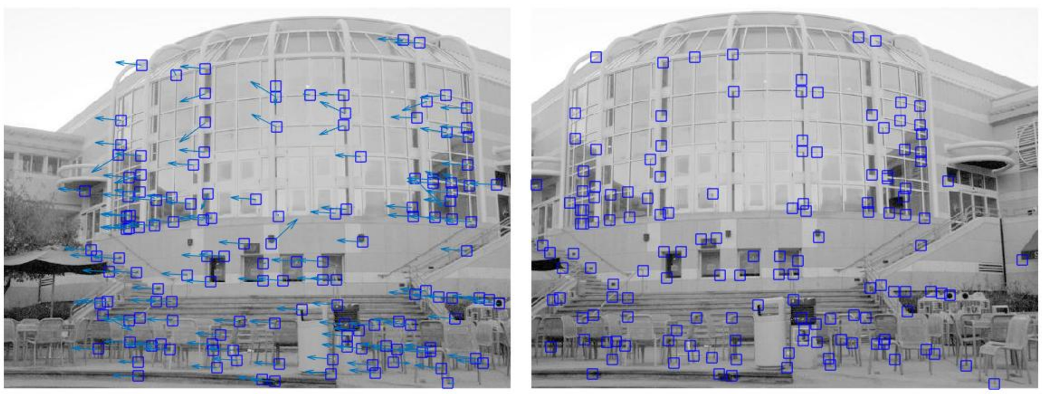

Figure 3: Matched Corner Picture

Appendix

Code Listing 1: Feature Detection

```
function mat = featureDetect(I, lambda_threshold, nw)

%% Parameters
hw = int32(floor(nw/2));
[n, m] = size(I);

%% Convolutional Kernel
k = [-1; 8; 0; -8; 1] / 12;

%% Calculate Gradient
Gx = conv2(double(I), k', 'same');
Gy = conv2(double(I), k, 'same');

%% Precalculate Squared Gradient
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;

%% Corner Detection
em = zeros(n, m);
for r = 1+nw:n-nw
    for c = 1+nw:m-nw
        vxx = sum(sum(Gxx(max(r-hw, 1):min(r+hw,n), ...
            max(c-hw, 1):min(c+hw,m))));
        vyy = sum(sum(Gyy(max(r-hw, 1):min(r+hw,n), ...
            max(c-hw, 1):min(c+hw,m))));
        vxy = sum(sum(Gxy(max(r-hw, 1):min(r+hw,n), ...
            max(c-hw, 1):min(c+hw,m))));
        ATA = [vxx, vxy; vxy, vyy];
        em(r, c) = (trace(ATA) - sqrt(trace(ATA)^2 - 4 * det(ATA))) /
                                                2;
    end
end

%% Non-maximum Suppression
mat = [];
for r = 1+nw:n-nw
    for c = 1+nw:m-nw
        vmax = max(max(em(max(r-hw, 1):min(r+hw,n), ...
            max(c-hw, 1):min(c+hw,m))));
        if em(r, c) == vmax && vmax > lambda_threshold
            mat = [mat; [c, r]];
        end
    end
end

%% Find Real Corners
[idx_x, idx_y] = meshgrid(1:m, 1:n);
xGxx = idx_x .* Gxx;
xGxy = idx_x .* Gxy;
yGxy = idx_y .* Gxy;
yGyy = idx_y .* Gyy;

for i = 1:size(mat,1)
    c = mat(i, 1);
    r = mat(i, 2);
    vxx = sum(sum(Gxx(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    vyy = sum(sum(Gyy(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    vxy = sum(sum(Gxy(max(r-hw, 1):min(r+hw,n), ...
```

```
        max(c-hw, 1):min(c+hw,m))));
    xVxx = sum(sum(xGxx(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    xVxy = sum(sum(xGxy(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    yVxy = sum(sum(yGxy(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    yVyy = sum(sum(yGyy(max(r-hw, 1):min(r+hw,n), ...
        max(c-hw, 1):min(c+hw,m))));
    A = [vxx, vxy; vxy, vyy];
    b = [xVxx + yVxy; xVxy + yVyy];
    mat(i, :) = (A \ b)';
end
```

Code Listing 2: Feature Match

```
function [lx,ly,lu,lv,rx,ry] = featureMatch(preI, nxtI, pref, nxtf, nw
                                    )

%% Parameters
hw = int32(floor(nw/2));
[n,m] = size(preI);
similarity_threshold = 0.5;
dist_threshold = 0.75;

%% Create Correlations
corrw = zeros(size(pref,1), size(nxtf,1));
for i = 1:size(pref,1)
    for j = 1:size(nxtf,1)
        prew = preI(max(pref(i,2)-hw, 1):min(pref(i,2)+hw,n), ...
            max(pref(i,1)-hw, 1):min(pref(i,1)+hw,m));
        nxtw = nxtI(max(nxtf(j,2)-hw, 1):min(nxtf(j,2)+hw,n), ...
            max(nxtf(j,1)-hw, 1):min(nxtf(j,1)+hw,m));
        corrw(i,j) = corr2(prew,nxtw);
    end
end

%% Find Largest Element Iteratively
lx = [];
ly = [];
lu = [];
lv = [];
rx = [];
ry = [];
while true
    [maxv,maxi] = max(corrw(:));
    % stop while the maximum one is not large enough
    if maxv < similarity_threshold
        break
    end

    % get the window corrdinate
    [r,c] = ind2sub(size(corrw),maxi);
    corrw(r,c) = -1;

    % check proximality
    if abs(pref(r,1) - nxtf(c,1)) > 100 ...
            || abs(pref(r,2) - nxtf(c,2)) > 100
        continue
    end

    % get the potential two window corrdinates
    [nrv,~] = max(corrw(r,:));
    [ncv,~] = max(corrw(:,c));
```

```
    % stack them into arrays
    if nrv > ncv
        if (1-maxv) < (1-nrv) * dist_threshold
            lx = [lx, pref(r,1)];
            ly = [ly, pref(r,2)];
            lu = [lu, nxtf(c,1)-pref(r,1)];
            lv = [lv, nxtf(c,2)-pref(r,2)];
            rx = [rx, nxtf(c,1)];
            ry = [ry, nxtf(c,2)];
        end
    else
        if (1-maxv) < (1-ncv) * dist_threshold
            lx = [lx, pref(r,1)];
            ly = [ly, pref(r,2)];
            lu = [lu, nxtf(c,1)-pref(r,1)];
            lv = [lv, nxtf(c,2)-pref(r,2)];
            rx = [rx, nxtf(c,1)];
            ry = [ry, nxtf(c,2)];
        end
    end

    % reset to -1
    for j = 1:size(nxtf,1)
        corrw(r,j) = -1;
    end
    for i = 1:size(pref,1)
        corrw(i,c) = -1;
    end
end
```

Code Listing 3: Main Procedure

```
%% Read Files
preI = imread('../dat/price_center20.JPG');
preI = rgb2gray(preI);
nxtI = imread('../dat/price_center21.JPG');
nxtI = rgb2gray(nxtI);

%% Parameters
lambda_threshold = 2200;
nw = 7;

%% Write Original Figures
res = figure('visible','off');
ha = tight_subplot(1, 2, 0.02, 0.01, 0.01);

axes(ha(1));
imshow(preI);

axes(ha(2));
imshow(nxtI);

saveas(res, '../res/pc_origin.jpg');

%% Extract Features
pref = featureDetect(preI, lambda_threshold, nw);
fprintf('Number of extracted features: %d\n', size(pref, 1));
nxtf = featureDetect(nxtI, lambda_threshold, nw);
fprintf('Number of extracted features: %d\n', size(nxtf, 1));

%% Write Feature-Deteced Figures
res = figure('visible','off');
ha = tight_subplot(1, 2, 0.02, 0.01, 0.01);

axes(ha(1));
```

```
imshow(preI);
hold on
for i = 1:size(pref,1)
    plot(pref(i,1),pref(i,2),'bs', 'MarkerSize', nw);
end
hold off

axes(ha(2));
imshow(nxtI);
hold on
for i = 1:size(nxtf,1)
    plot(nxtf(i,1),nxtf(i,2),'bs', 'MarkerSize', nw);
end
hold off

saveas(res, '../res/pc_detect.jpg');

%% Match Features
[lx, ly, lu, lv, rx, ry] = featureMatch(preI, nxtI, pref, nxtf, nw);
fprintf('Number of matches: %d\n', size(lx, 2));

%% Draw Figures
res = figure('visible','off');

ha = tight_subplot(1, 2, 0.02, 0.01, 0.01);
axes(ha(1));
imshow(preI);
hold on
for i = 1:size(lx,2)
    plot(lx, ly, 'bs', 'MarkerSize', nw);
end
quiver(lx, ly, lu, lv);
hold off

axes(ha(2));
imshow(nxtI);
hold on
for i = 1:size(rx,2)
    plot(rx, ry, 'bs', 'MarkerSize', nw);
end
hold off

saveas(res, '../res/pc_match.jpg');
```