CLab2 Report

ENGN6528
You Li
u6430173

02 May 2019

Contents

Chapter 1

Task1 Harris Corner Detector

1.1 Harris Corner Detection Algorithm

The Harris Matrices is

$$A_{\text{Harris}} = \begin{bmatrix} \left\langle G_x^2 \right\rangle & \langle G_{xy} \rangle \\ \langle G_{xy} \rangle & \left\langle G_y^2 \right\rangle \end{bmatrix}$$

The key simplification of the Harris algorithm is estimating the eigenvalues of the Harris matrix as the determinant minus the scaled trace squared.

$$\lambda_1 = \frac{Tr(A)}{2} + \sqrt{\frac{Tr^2(A)}{4} - \det(A)}$$
$$\lambda_2 = \frac{\text{Tr}(A)}{2} - \sqrt{\frac{Tr^2(A)}{4} - \det(A)}$$

The corner metric response is

$$R = \det\left(A_{\text{Harris}}\right) - k \cdot \text{Tr}^2\left(A_{\text{Harris}}\right)$$

The corresponding eigenvalues are

$$\lambda_1 = \left(\frac{\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle}{2}\right) + \sqrt{\left(\frac{\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle}{2}\right)^2 - \left(\left\langle G_x^2 \right\rangle \cdot \left\langle G_y^2 \right\rangle - \langle G_{xy} \rangle^2\right)}$$

$$\lambda_2 = \left(\frac{\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle}{2}\right) - \sqrt{\left(\frac{\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle}{2}\right)^2 - \left(\left\langle G_x^2 \right\rangle \cdot \left\langle G_y^2 \right\rangle - \langle G_{xy} \rangle^2}$$

$$\lambda_1, \lambda_2 > \quad k_{\text{minimum}}$$
$$\lambda_1 - \lambda_2 < k_{\text{thresh}}$$

$$2\sqrt{\tfrac{Tr^2(A)}{4} - \det(A)} < \quad k_{\text{thresh}}$$
$$\tfrac{Tr^2(A)}{4} - \det(A) < \left(\tfrac{k_{\text{thresh}}}{2}\right)^2$$

$$\det(A) - \frac{Tr^2(A)}{4} \geq \left(\frac{k_{\text{thresh}}}{2}\right)^2$$

$$\left(\left\langle G_x^2 \right\rangle \cdot \left\langle G_y^2 \right\rangle - \left\langle G_{xy} \right\rangle^2\right) - \left(\frac{\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle}{2}\right)^2 \geq \left(\frac{k_{\text{thresh}}}{2}\right)^2$$

Therefore, the response metric is

$$R = \left(\left\langle G_x^2 \right\rangle \cdot \left\langle G_y^2 \right\rangle - \left\langle G_{xy} \right\rangle^2\right) - k \cdot \left(\left\langle G_x^2 \right\rangle + \left\langle G_y^2 \right\rangle\right)^2$$

1.2 *Harris.m* and comments

```
% My Harris function
% Reference: https://au.mathworks.com/help/visionhdl/examples/corner-de
function H_corners = my_harris(task_image)
% check task_image dimension
if (tsk_size == 3)
    task_image = rgb2gray(task_image);
end
%%%%%%%%%%%%%%Intrest Points %%%%%%%%%%%%%%%
sigma=2; Thrshold=0.01; sensitivity=11; % disp=0;
dx = [-1 0 1; -1 0 1; -1 0 1]; % The Mask
dy = dx';
% compute x and y derivatives of image

Ix = conv2(gray_task, dx, 'same');
Iy = conv2(gray_task, dy, 'same');

%%%%%% Gaussien Filter
% Generate Gaussian filter 'g'
% of size 9x9 and standard deviation Sigma.
g = fspecial('gaussian',max(1,fix(6*sigma)), sigma);

Ix2 = conv2(Ix.^2, g, 'same');
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g,'same');
% Smooth the squared image derivatives to obtain Ix2, Iy2 and IxIy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task: Perform non-maximum suppression and               %
%       thresholding, return the N corner points          %
%       as an Nx2 matrix of x and y coordinates           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Build Harris autocorrelation matrix
% for every singe pixel considering a window of size 3x3
HarrisA =  [Ix2 Ixy, Ixy Iy2];
th_k = 0.04;
% threshold k
% Compute Response Matrix R which contains
% for every point the cornerness score from HarrisA.
```

```
        Response = (Ix2.*Iy2 - Ixy.^2) - th_k*(Ix2 + Iy2).^2;
        R=Response;
        % add constrain matrix
        sze = 2*sensitivity+1;
        MX = ordfilt2(Response,sze^2,ones(sze));
        count = 0;
        img_row = 5:size(Response,2)-5;
        img_col = 5:size(Response,1)-5;
        % init the corner num
        loop=0;
        while (loop<30)
            % When the response is larger than a predefined threshold,
            % a corner is detected:
            Response = (R==MX)&(R>Thrshold);
            count = sum(sum(Response(img_col, img_row)));
            % Get the coordinates with maximum cornerness responses
            loop=loop+1;
        end
        R=R*0;
        R(img_col, img_row) = Response(img_col, img_row);
        [col, row] = find(R);
        %The mapping from linear indexes
        % to subscript equivalents for the matrix
        % Get the coordinates with maximum cornerness responses
        H_corners=[col, row];
        % returns the corners matrix
        end
```

end of matlab code.

1.3 Result images and compare

The image at left is the resul of my_Harrias, and the image at the RHS is the result of *corner*().

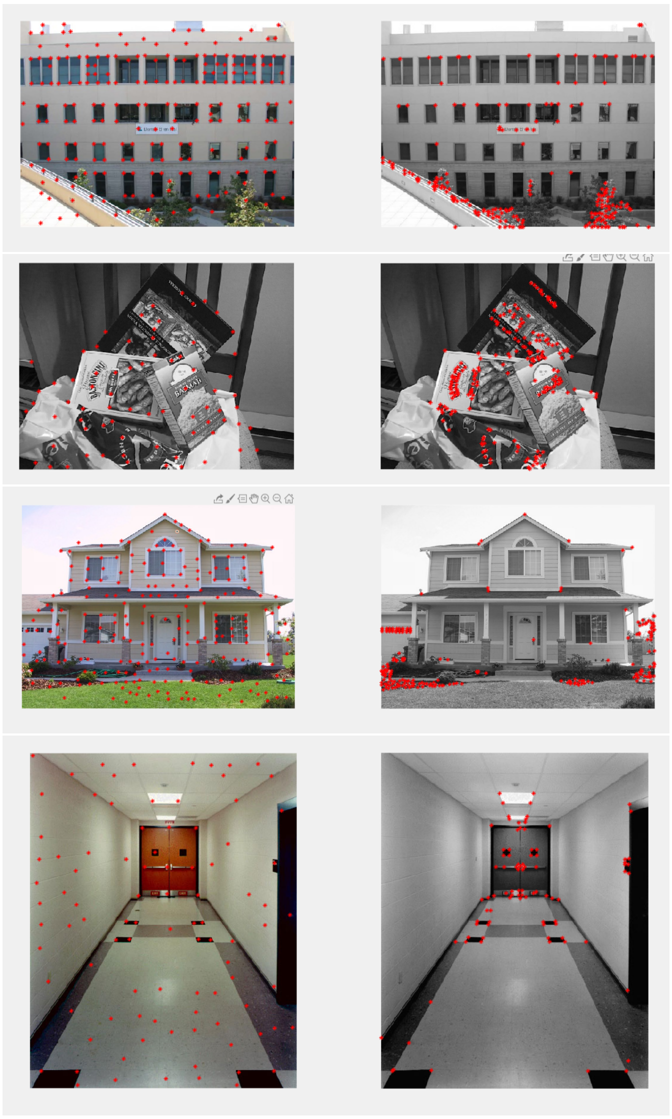

Figure 1.1: task1.4

1.4 factors of $corner()$

According to matlab document, $C = corner(\_\_\_, Name, Value)$ specifies parameters and corresponding values that control various aspects of the corner detection algorithm. This would provide a hint for Harris variable experiments.

1.4.1 $FilterCoefficients$

the smoothing filter, i.e. the sigma of the Gaussian filter. If the sigma is increased, the image will be blurred and only strong or sharp corner will be calculated. For image $Harris\_4$, as the result above, the sigma value is set as 8 to reduce the corners detection on the wall. Increasing threshold also does improve the result.

1.4.2 $QualityLevel$

Minimum accepted quality, i.e. the threshold of the Harris function. Larger values of Q used to remove erroneous corners.
In the experiment in $Harris\_1.pgm$, the threshold is increased in order to eliminate the corners on the curtain. Hovever, increasing the threshold will decrease the points detected on the books at the same time.

1.4.3 $SensitivityFactor$

The smaller the sensitivity factor, the more likely the algorithm is to detect sharp corners.
In the experiment of $Harris\_3$, the sensitivity of correction matrix is increased. Otherwise, the corners on the edge of eadh windows and the grass will be calculated. The tests on $Harris\_1.jpg$ also prove that the sensitivity should be increased to reduce the redundant corners.

Chapter 2

Task2 K-Means

2.1 Implementation of $my_kmeans()$

```
%% K-means
function [CENTS, DAL] = km_fun(F, K, KMI)
CENTS = F( ceil(rand(K,1)*size(F,1)) ,:);
% Cluster Centers
DAL   = zeros(size(F,1),K+2);
% Distances and Labels
for n = 1:KMI

   for i = 1:size(F,1)
      for j = 1:K
        DAL(i,j) = norm(F(i,:) - CENTS(j,:));
      end
      [Distance, CN] = min(DAL(i,1:K));
      % 1:K are Distance from Cluster Centers 1:K
      DAL(i,K+1) = CN;
      % K+1 is Cluster Label
      DAL(i,K+2) = Distance;
      % K+2 is Minimum Distance
   end
   for i = 1:K
      A = (DAL(:,K+1) == i);
      % Cluster K Points
      CENTS(i,:) = mean(F(A,:));
```

```
            % New Cluster Centers
            if sum(isnan(CENTS(:))) ~= 0
            % If CENTS(i,:) Is Nan Then Replace It With Random Point
               NC = find(isnan(CENTS(:,1)) == 1);
               % Find Nan Centers
               for Ind = 1:size(NC,1)
               CENTS(NC(Ind),:) = F(randi(size(F,1)),:);
               end
            end
         end
      end
      end
```

2.2 Apply K-Means in image segmentation

2.3 K-Means++

The process of K-Means ++ is listed as follow.

1. Choose one center uniformly at random from among the data points.
2. For each data point x, compute D(x), the distance between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to D(x)2.
4. Repeat Steps 2 and 3 until k centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard k-means clustering.

The convergence speed is significantly improved by K-Means ++, since it force the initialization of seed. However, the segmentation result depends on the initial chosen random value. Mostly the performance is better than normal K-Means. Further experiment could be studied on the performance of K-Means with different initialization values using deep learning.

Chapter 3

Task3 Eigenface

3.1 Alignment for Eigenface

The training pictures should have been taken under the same lighting conditions, and must be normalized to have the eyes and mouths aligned across all images with common pixel resolution. This is the alignment process. Because each image is treated as one vector, it is assumed that all images of the training set are stored in a single matrix T, where each column of the matrix is an image. Eigenface is then the eigen vector which are derived from the covariance matrix of this matrix of parsed face data. Alignment increase accuracy and reduce noise.

3.2 PCA mean face

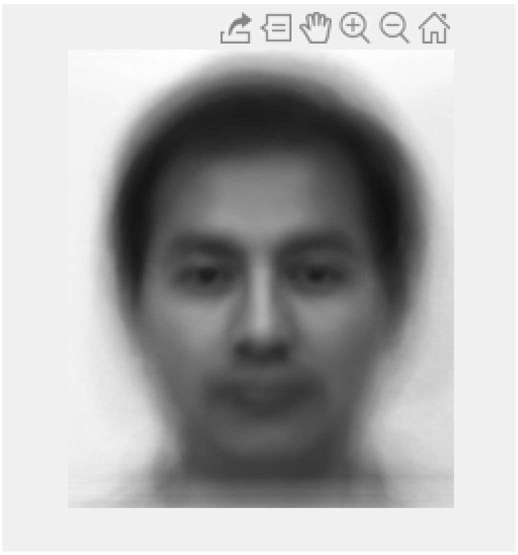

Figure 3.1: task3 - mean face

3.3 Improve on calculation of eigen vectors

For a MxN matrix, the maximum number of non-zero eigenvalues that its covariance matrix can have is min[M-1,N-1]. The number of dimensions (pixels) of each image vector is very high compared to number of test images here, so number of non-zero eigenvalues of C will be maximum P-1 (P being the number of test images) if we calculate eigenvalues & eigenvectors of $= A \times A'$, then it will be very time consuming as well as memory. so we calculate eigenvalues & eigenvectors of $L = A' \times A$, whose eigenvectors will be linearly related to eigenvectors of C. these eigenvectors being calculated from non-zero eigenvalues of C, will represent the best feature sets.

```
        L_eig_vec = [];
        for i = 1 : size(V,2)
            if( D(i,i) > 1 )
                L_eig_vec = [L_eig_vec V(:,i)];
                % if corresponding eigenvalue is greater than 1,
                % then the eigenvector will be chosen for creating eigenface
        end
    end
```

3.4 Top k Eigenfaces

Assume that most face images lie on a low-dimensional subspace determined by the first $k (k < d)$ directions of maximum variance. Use PCA to determine the eigenfaces $u_1, ...u_k$ that span that subspace. Represent all face images in the training set as linear combinations of eigenfaces.

The top k = 10 eigenfaces plotted as following

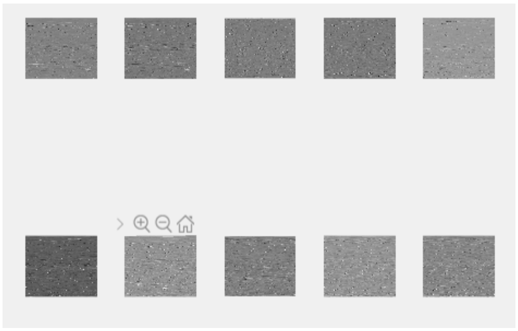

Figure 3.2: task3 - top k eigenfaces

3.5 Recognition of Test Images

Then project each training image x_i onto subspace spanned by principal components.Face x in "face space" coordinates is

$$\mathbf{x} = \boldsymbol{\mu} + \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

with error

$$E_k = \frac{\sum_{i=k+1}^{K} \lambda_i}{\sum_{i=1}^{K} \lambda_i}$$

The most similar faces are:

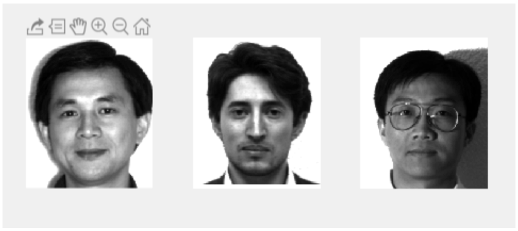

Figure 3.3: task3 - top 3 of test set

3.6 Test on front photos

The most nearest photo of the normal face are following. Its explainable that the result comes out as the faces without expression.

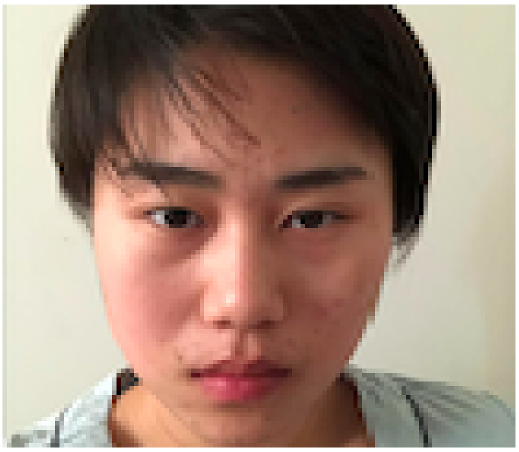

Figure 3.4: task3 - front test-1

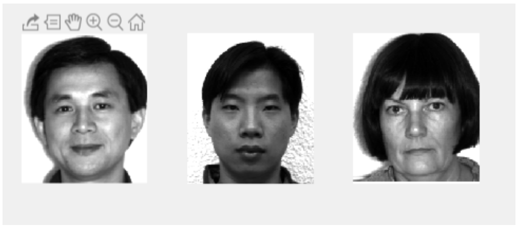

Figure 3.5: task3 - front test1

For the rest of the images, the top 3 faces are

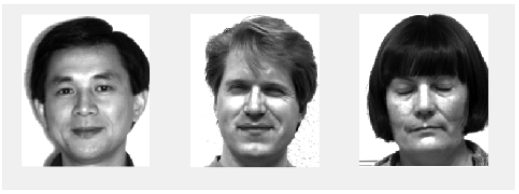

Figure 3.6: task3 - front test-2