Face detection

Inseong Kim, Joon Hyung Shim, and Jinkyu Yang

Introduction

In recent years, face recognition has attracted much attention and its research has rapidly expanded by not only engineers but also neuroscientists, since it has many potential applications in computer vision communication and automatic access control system. Especially, face detection is an important part of face recognition as the first step of automatic face recognition. However, face detection is not straightforward because it has lots of variations of image appearance, such as pose variation (front, non-front), occlusion, image orientation, illuminating condition and facial expression.

Many novel methods have been proposed to resolve each variation listed above. For example, the *template-matching methods* [1], [2] are used for face localization and detection by computing the correlation of an input image to a standard face pattern. The *feature invariant approaches* are used for feature detection [3], [4] of eyes, mouth, ears, nose, etc. The *appearance-based methods* are used for face detection with *eigenface* [5], [6], [7], *neural network* [8], [9], and *information theoretical approach* [10], [11]. Nevertheless, implementing the methods altogether is still a great challenge. Fortunately, the images used in this project have some degree of uniformity thus the detection algorithm can be simpler: first, the all the faces are vertical and have frontal view; second, they are under almost the same illuminate condition. This project presents a face detection technique mainly based on the color segmentation, image segmentation and template matching methods.

Color Segmentation

Detection of skin color in color images is a very popular and useful technique for face detection. Many techniques [12], [13] have reported for locating skin color regions in the input image. While the input color image is typically in the RGB format, these techniques usually use color components in the color space, such as the HSV or YIQ formats. That is because RGB components are subject to the lighting conditions thus the face detection may fail if the lighting condition changes. Among many

color spaces, this project used YCbCr components since it is one of existing Matlab functions thus would save the computation time. In the YCbCr color space, the luminance information is contained in Y component; and, the chrominance information is in Cb and Cr. Therefore, the luminance information can be easily de-embedded. The RGB components were converted to the YCbCr components using the following formula.

$$\begin{aligned} \mathbf{Y} &= 0.299\mathbf{R} + 0.587\mathbf{G} + 0.114\mathbf{B} \\ \mathbf{Cb} &= -0.169\mathbf{R} - 0.332\mathbf{G} + 0.500\mathbf{B} \\ \mathbf{Cr} &= 0.500\mathbf{R} - 0.419\mathbf{G} - 0.081\mathbf{B} \end{aligned}$$

In the skin color detection process, each pixel was classified as skin or non-skin based on its color components. The detection window for skin color was determined based on the mean and standard deviation of Cb and Cr component, obtained using 164 training faces in 7 input images. The Cb and Cr components of 164 faces are plotted in the color space in Fig.1; their histogram distribution is shown in Fig. 2.

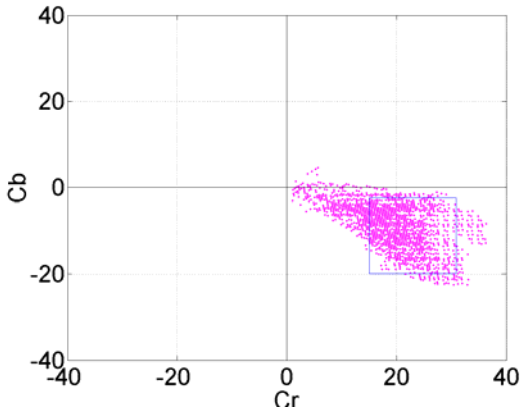


Fig. 1 Skin pixel in YCbCr color space.

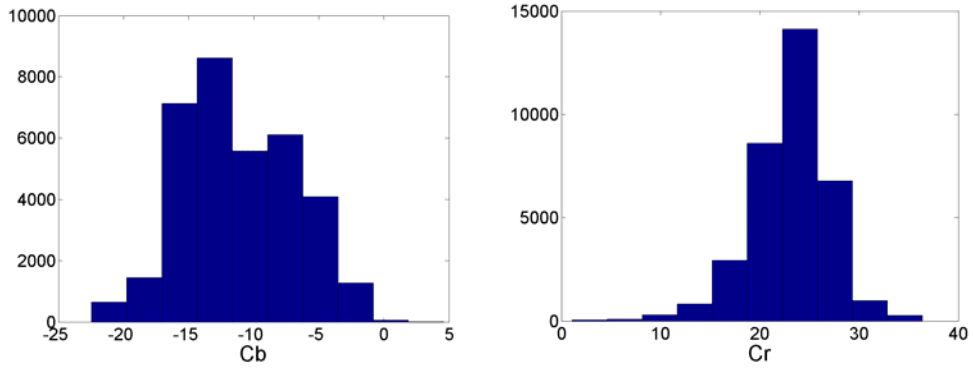


Fig. 2 (a) Histogram distribution of Cb. (b) Histogram distribution of Cr.

The color segmentation has been applied to a training image and its result is shown in Fig. 3. Some non-skin objects are inevitably observed in the result as their colors fall into the skin color space.

Fig. 3 Color segmentation result of a training image.

Image Segmentation

The next step is to separate the image blobs in the color filtered binary image into individual regions. The process consists of three steps. The first step is to fill up black isolated holes and to remove white isolated regions which are smaller than the minimum face area in training images. The threshold (170 pixels) is set conservatively. The filtered image followed by initial erosion only leaves the white regions with reasonable areas as illustrated in Fig. 4.

Fig. 4. Small regions eliminated image.

Secondly, to separate some integrated regions into individual faces, the *Roberts Cross Edge detection algorithm* is used. The *Roberts Cross Operator* performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial gradients that often correspond to edges. (Fig. 5.)The highlighted region is converted into black lines and eroded to connect crossly separated pixels.

Fig.5. Edges detected by the Roberts cross operator.

Finally, the previous images are integrated into one binary image and relatively small black and white areas are removed. The difference between this process and the initial small area elimination is that the edges connected to black areas remain even after filtering. And those edges play important roles as boundaries between face areas after erosion. Fig. 6. shows the final binary images and some candidate spots that will be compared with the representative face templates in the next step are introduced in Fig. 7.

Fig.6. Integrated binary image.

Fig.7. Preliminary face detection with red marks.

Image Matching

Eigenimage Generation

A set of eigenimages was generated using 106 test images which were manually cut from 7 test images and edited in *Photoshop* to catch exact location of faces with a square shape. The cropped test images were converted into gray scale, and then eigenimages were computed using those 106 test images. In order to get a generalized shape of a face, the largest 10 eigenimages in terms of their energy densities, have been obtained as shown in the Fig. 8. To save computing time, the information of eigenimages was compacted into one image which was acquired after averaging the first 9 eigenimages excluding the eigenimage 1, the highest-energy one. The first image was excluded due to its excessive energy concentration which will eliminate the details of face shapes that can be shown from other eigenimages from eigenimage 2 to eigenimage 10. The averaged eigenimage is shown in Fig. 9.

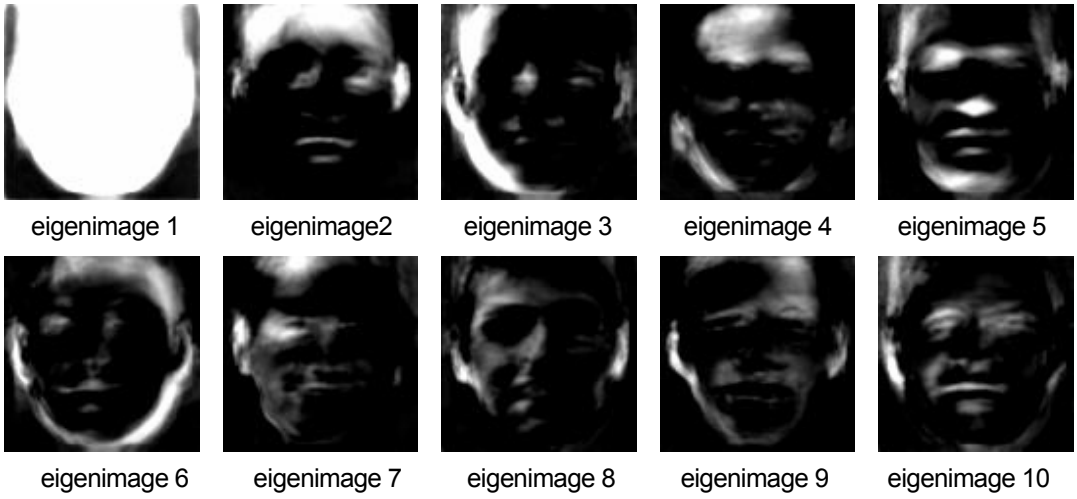


Fig.8. Eigenimages

Fig.9. Average image using eigenimages

Building Eigenimage Database

In order to save time to magnify or shrink an eigenimage to meet the size of the test image, a group of eigenimages was stored in the database so that an appropriate eigenimage can be called with ease without going through image enlarging or shrinking process. The eigenimages were stored in 20 files from 30 pixel-width square image to 220 pixel-width square image with 10-pixel step. The stored eigenimages were normalized by means of dividing the image matrix by its 2^{nd} *norm* so that the effect of eigenimage size does not affect the face detection algorithm.

Test Image Selection

After the color-based segmentation process, skin-colored area can be taken apart as shown in Fig. 6. Given this binary image, a set of small test images needs to be selected and passed to the image matching algorithm for the further process. The result of image selection solely based on the color information is shown in Fig. 10. A square box was applied on each segment with the quantified window size which was selected to meet the size of a face.

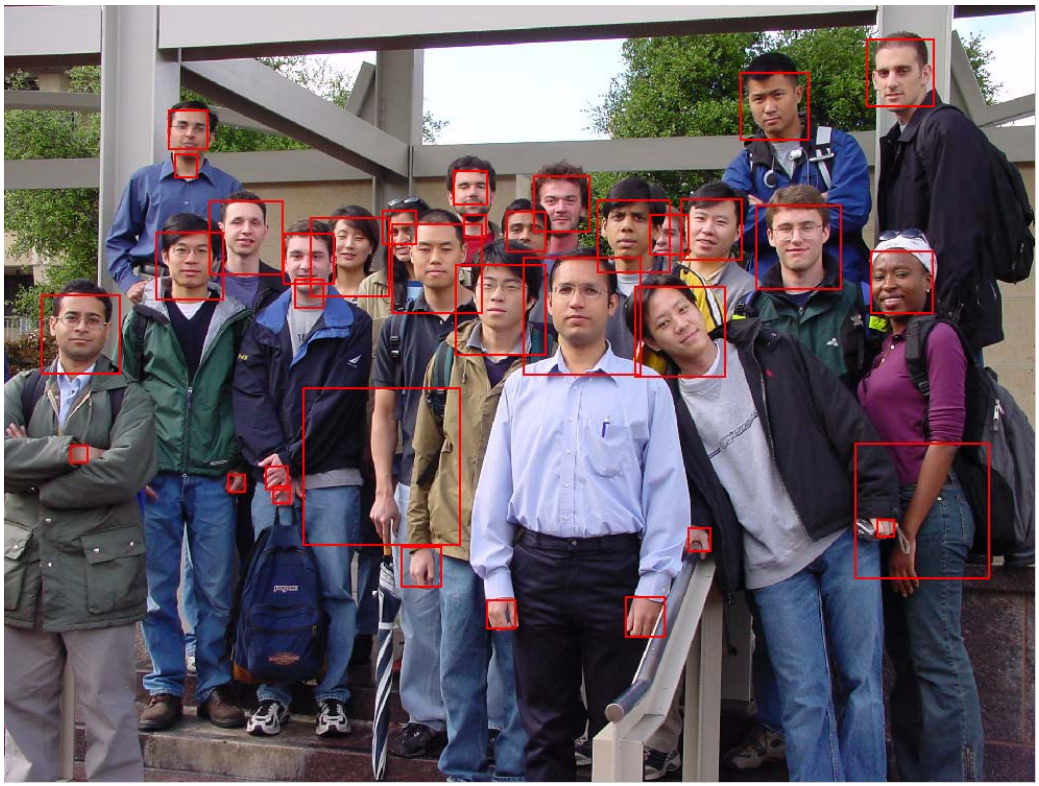
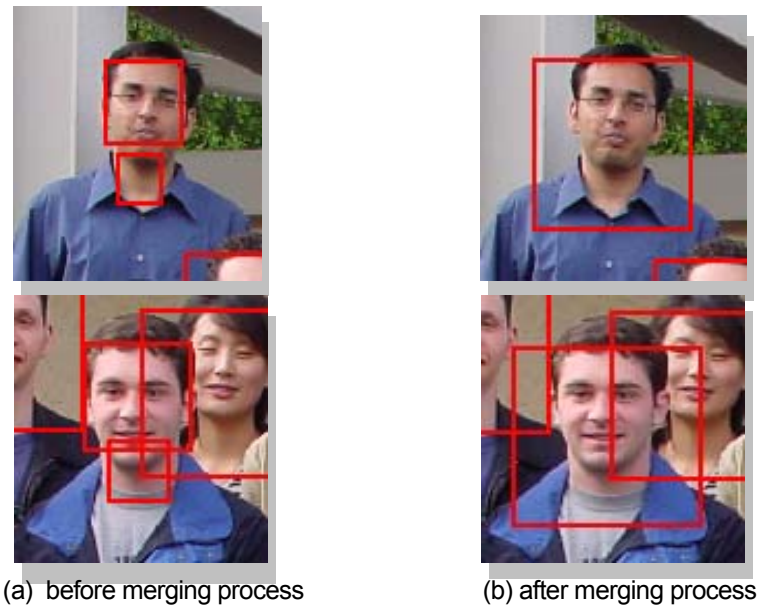


Fig. 10. Test Image Selection using Color-Based Image Segmentation.

If Fig. 10 is examined closely, some faces are divided into several pieces, for example the face being separated into its upper part and neck part as seen in Fig. 11 (a). This is due to the erosion process which was applied to evade *occlusion*. To merge these separate areas into one area, *box-merge algorithm* was used which simply merges two or more adjacent square boxes into one. Since this phenomenon happens between face and neck part most of times, distance threshold was set small for horizontal direction, while set large for vertical direction. The results after merging two boxes in Fig. 11 (a) are shown in Fig. 11 (b). After applying this algorithm, it can be found that only one box is placed per face most of times in Fig. 12.



(a) before merging process (b) after merging process

Fig. 11. Test Image Selection: Merging of Adjacent Boxes.

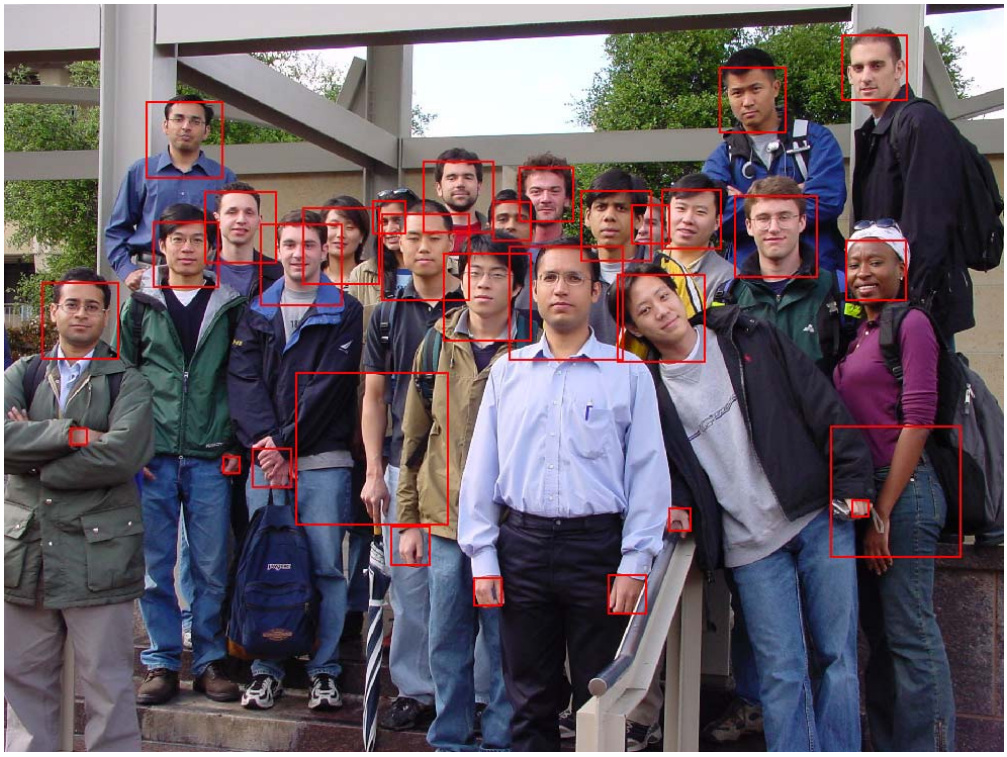


Fig. 12. Test Image Selection after Applying *Box-Merge Algorithm.*

Correlation

The test images selected by an appropriate square window can be passed to the image matching algorithm. Before the image matching process, the test image need to be converted to gray scale, and should be divided by the *average brightness* of the image in order to eliminate the effect of the brightness of the test image in the process of image matching. *Average brightness* was defines as 2^{nd} *norm* of the skin-colored area of the test image. Note that it is not the 2^{nd} *norm* applied to the total area of the test image, since the value that we are looking for is not the average brightness of the test image, but the average brightness of the skin colored parts only.

With the normalized test image, the image matching can be simply accomplished by loading a correspondent file of eigenimage from the database, then performing correlation of the test image with respect to the loaded eigenimage. The results of image matching are illustrated in Fig. 13. The number inside each window means the ranking of the correlation value.

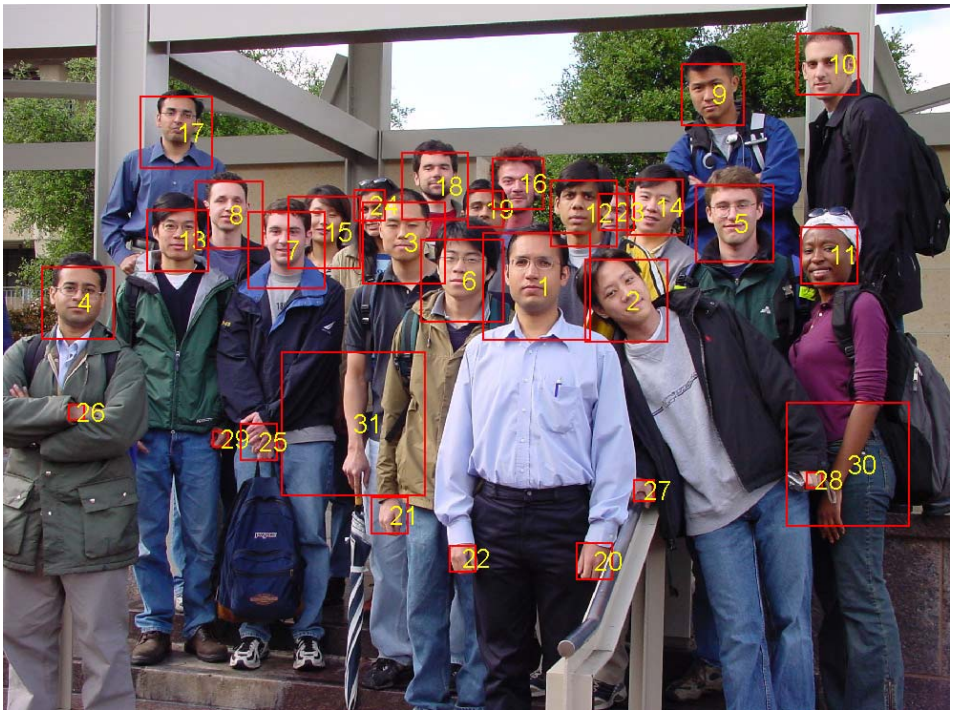


Fig. 13. Selected Test Images with Correlation Ranking Information.

Distance Compensation

Since the figure to be tested will be a group picture, faces in the figure are located close to the each other in the central area of the figure. However, hands, arms, or legs are relatively located far from the faces in the figure. Therefore, the mean square distance of a test image with respect to other test images can be calculated, and then its reciprocal can be multiplied to the correlation value obtained above, to take the geographical information into account. In other words, a test image which is located close to the other test images will get larger correlation value, while a test image which is far from the other group will have smaller correlation value. The ranking of the correlation values of the test images after this evaluation is shown in Fig. 14.

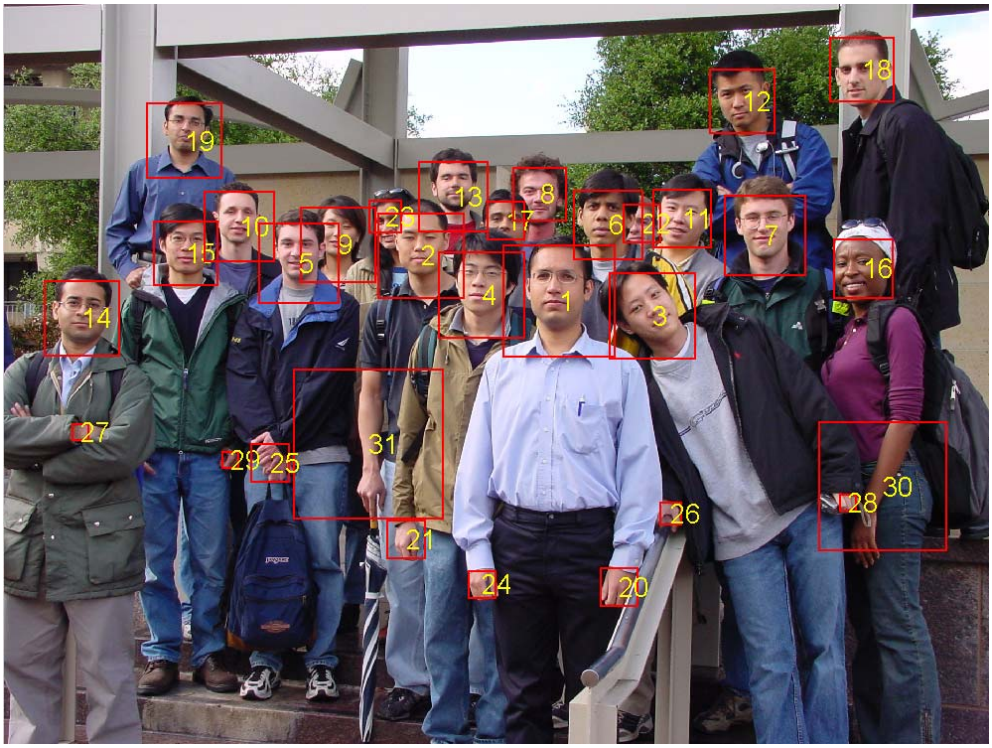


Fig. 14. Correlation Ranking after Geographical Consideration.

Filtering Non-facial Test Images using Statistical Information

The next step is filtering out non-facial test images from the figure. Several approaches have been taken, but it was not easy to find an absolute threshold value which can be applied to various pictures with different light condition and composition. Approaches using luminance, *average brightness,* and etc., have been tried, but they turned out not to be good enough to set an appropriate threshold for filtering out non-facial test images. Lastly statistical method was tried. As seen in the Fig. 15, the histogram of the correlation values after geographical consideration shows wide distribution of the output values. The leftmost column corresponds to the test images which have smallest correlation values among the set of the test images.

After filtering out the leftmost column elements, which is 12 test images for this example figure, Fig. 16 was obtained. Out of 21 faces in the picture, the algorithm has detected 19 faces within acceptable error of the location of faces. The two undetected faces are partially blocked by the other faces. Conclusively, this statistical approach, which is seemingly rough estimation, works great in this picture, and is turned out to produce good results for other pictures as well.

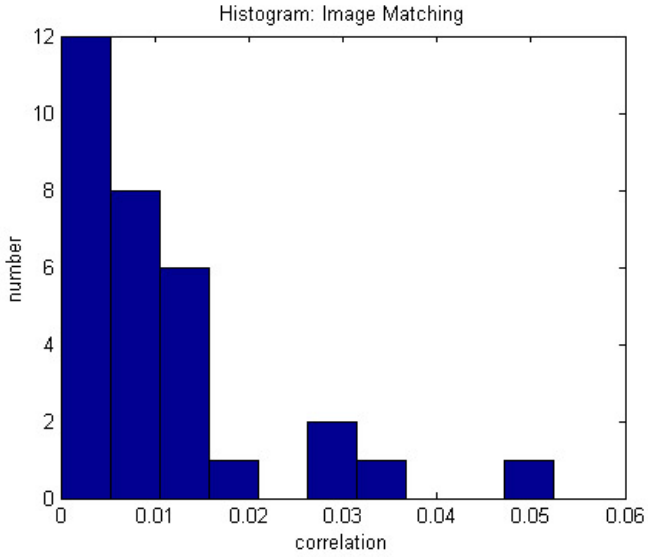


Fig. 15. Histogram: Image Matching.

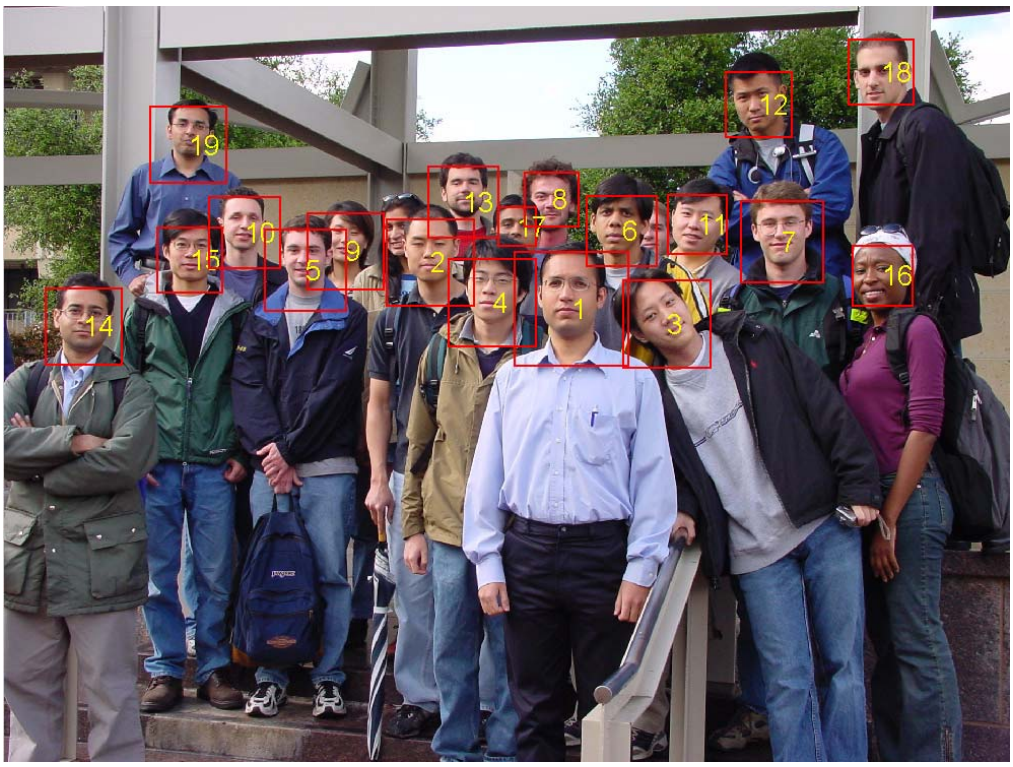


Fig. 16. Correlation Ranking after Geographical Consideration.

Results

The test was performed using 7 training images. The results are as followed in Table 1.

Table 1. Face Detection Results using 7 Training Images.

	numFaces	numHit	numRepeat	numFalse	run time [sec]
Training_1.jpg	21	19	0	0	111
Training_2.jpg	24	24	0	1	101
Training_3.jpg	25	23	0	1	89
Training_4.jpg	24	21	0	1	84
Training_5.jpg	24	22	0	0	93
Training_6.jpg	24	22	0	3	100
Training_7.jpg	22	22	0	1	95

numFaces : total number of faces in the picture
numHit : number of faces successfully detected
numRepeat : number of faces repeatedly detected
numFalse : number of case where a non-face is reported
run time : run time of the face detection routine
* run time measure in Pentium3 700 MHz, 448 Mega Memory; run time includes gender detection algorithm

The face detection algorithm shows 93.3 % of right hit rate, and 0 % of repeat rate, and 4.2 % of false hit rate. The average run time is 96 seconds.

In order to see if this algorithm works for other than the 7 training images, last year's sample picture was test, and the result is as shown in Fig. 17. The results show that 20. out of 24 faces have been successfully located, and there was no repeat or false detection.

Fig. 17. Face Detection of Last Year's Picture.

Gender Recognition

Gender recognition algorithm has been implemented to detect at most 3 females in a test photo. Since the number of females to detect is only 3, *average faces* of the three females have been calculated as shown in Fig. 18, and image matching was performed for each of these average faces.

Fig. 18. Average Faces of the Females in the Class.

The test images obtained from *Test Image Selection Algorithm* which explained in the previous section were used to match with these three female average faces. The information about the average faces was stored in the database using the identical method as performed for saving eigenfaces before.

After running the algorithm for the 7 training image sets, the results are that the *average face image matching method* did not detect any female faces at all, but detected the face which has the largest correlation value for general face detection. However, the inaccuracy of the *average face image matching* was expected, because in order to have selectivity of this algorithm, a test image should be taken very precisely so that it exactly overlaps the *eigenface* in terms of its center location and box size. In the real case, the box which defined the contour of a face was bigger or smaller than it should be, and the center was hardly overlapping either.

More sophisticated algorithm will be required in order to accomplish gender recognition, or further, the face recognition of a certain character.

Discussion

In color segmentation, a rectangular window was used for the skin color detection while the actual distribution was a cone shape. As a result, some of actual skin color was excluded and conversely some of non-skin color was included. More precise skin color detection is expected if the window shape is closer to the actual distribution, such as triangle. Despite this imperfect windowing, the overall results of skin color detection were very encouraging.

In some cases, unnecessary noises were added in the process of edge integration in image segmentation. Though the *Roberts cross operator* minimized the problem associated with the following small-hole removal process, some black region connected to peripheral face edges were magnified by series of erosions and even divided into separate face regions that were recognized as several small faces. The *Sobel cross filter* or the *Prewitt filter* with pre-rejection of small clutters were proved to work more effectively in some training images, but the threshold from which the area is regarded as a 'small clutter' varies even in a single picture and it discouraged the filter applied in our algorithm. However, the threshold decided in a more interactive way with the original image is able to discriminate face edges from other edge lines effectively and lots of applications have been presented.

Based on the color segmentation, skin-colored areas were taken apart into small and squared test images. In some cases, several squared test images were generated within a face due to excessive erosion in the color segmentation process. Before performing eigenimage matching with the test images, these unnecessary squares should be merged into one so that only one test image can be taken

per face. In order to judge if it corresponds to a face or not, each test image needs to be matched with eigenimage which is simply called from the data base which were already built in. After eigenimage matching process, the test images can be ordered from the most-like facial piece to the least-like facial piece. At this point, an absolute criterion or threshold is needed to filter out the non-facial images. However, due to the inconsistency of the condition of each picture, it is concluded that there does not exist such an absolute number. Therefore, statistical approach has been taken so that the test images which belong to the least-like facial group are rejected. The number of the rejected files totally depends on the histogram results of the eigenimage matching results. This approach works satisfactorily for the 7 training images with higher than 90% of hit rate, and less than 5% of false detection rate. The algorithm has been tested for a general image which was used for the last year's project, and the performance was satisfactory.

The developed algorithm has successfully performed the face detection. However, more features with sophisticated algorithm need to be added in order to use it for more general applications. Nevertheless, this project was effective enough to endow the group members with a chance to glance at the world of digital image processing.

References

[1] I. Craw, D. Tock, and A. Bennett, "Finding face features," Proc.of 2nd European Conf. Computer Vision. pp. 92-96, 1992.

[2] A. Lanitis, C. J. Taylor, and T. F. Cootes, "An automatic face identification system using flexible appearance models," Image and Vision Computing, vol.13, no.5, pp.393-401, 1995.

[3] T. K. Leung, M. C. Burl, and P. Perona, "Finding faces in cluttered scenes using random labeled graph matching," Proc. 5th IEEE int'l Conf. Computer Vision, pp. 637-644, 1995.

[4] B. Moghaddam and A. Pentland, "Probabilistic visual learning for object recognition," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no.7. pp. 696-710, July, 1997.

[5] M. Turk and A. Pentland, "Eigenfaces for recognition," J. of Cognitive Neuroscience, vol.3, no. 1, pp. 71-86, 1991.

[6] M. Kirby and L. Sirovich, "Application of the Karhunen-Loeve procedure for the characterization of human faces," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.12, no.1, pp. 103-108, Jan. 1990.

[7] I. T. Jolliffe, Principal component analysis, New York: Springer-Verlag, 1986.

[8] T, Agui, Y. Kokubo, H. Nagashi, and T. Nagao, "Extraction of face recognition from monochromatic photographs using neural networks," Proc. 2nd Int'l Conf. Automation, Robotics, and Computer Vision, vol.1, pp. 18.81-18.8.5, 1992.

[9] O. Bernier, M. Collobert, R. Feraud, V. Lemaried, J. E. Viallet, and D. Collobert, "MULTRAK: A system for automatic multiperson localization and tracking in real-time," Proc, IEEE. Int'l Conf. Image Processing, pp. 136-140, 1998.

[10] A. J. Colmenarez and T. S. Huang, "Face detection with information-based maximum discrimination," Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 782-787, 1997.

[11] M. S. Lew, "Information theoretic view-based and modular face detection," Proc. 2nd Int'l Conf. Automatic Face and Gesture Recognition, pp. 198-203, 1996.

[12] H. Martin Hunke, Locating and tracking of human faces with neural network, Master's thesis, University of Karlsruhe, 1994.

[13] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. "Neural network based face detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(I), pp.23-38, 1998.

Reference: Roberts Cross Edge Detection Algorithm

In theory, the operator consists of a pair of 2×2 convolution masks as shown in Figure 1. One mask is simply the other rotated by 90°.

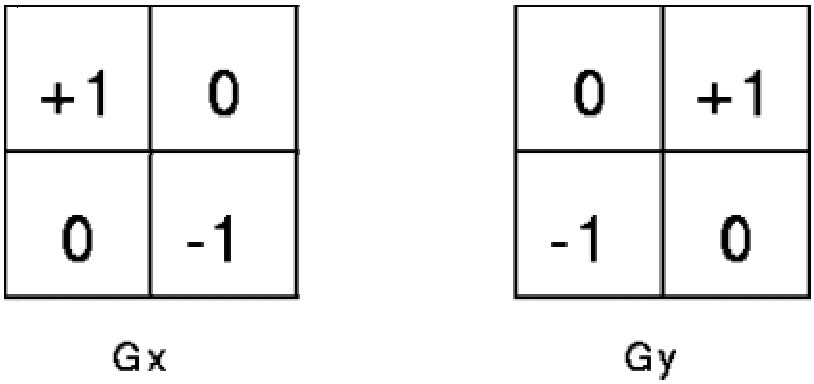


Fig 1 Roberts Cross convolution masks

These masks are designed to respond maximally to edges running at 45° to the pixel grid, one mask for each of the two perpendicular orientations. The masks can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these G_x *and* G_y). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = (G_x^2 + G_y^2)^{1/2}$$

although typically, an approximate magnitude is computed using:

$$|G| = |G_x| + |G_y|$$

which is much faster to compute.

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\theta = arctan\ (G_y / G_x) - 3\pi/4$$

In this case, orientation 0 is taken to mean that the direction of maximum contrast from black to white runs from left to right on the image, and other angles are measured anti-clockwise from this.

Often, the absolute magnitude is the only output the user sees. The two components of the gradient are conveniently computed and added in a single pass over the input image using the pseudo-convolution operator shown in Figure 2.

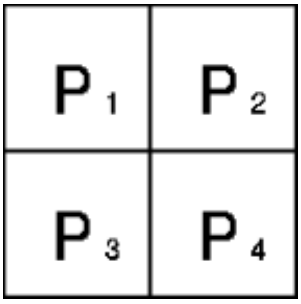


Fig 2 Pseudo-Convolution masks used to quickly compute approximate gradient magnitude

Using this mask the approximate magnitude is given by:

$$|G| = |P_1 - P_4| + |P_2 - P_3|$$

Source Code

faceDection.m

```
function outFaces = faceDetection(img)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function 'outFaces' returns the matrix with the information of
% face locations and gender.
%
% outFaces = faceDetection(img)
%  img: double formatted image matrix

% coefficients
effect_num=3;
min_face=170;
small_area=15;
imgSize = size(img);

uint8Img = uint8(img);
gray_img=rgb2gray(uint8Img);

% get the image tranformed through YCbCr filter
filtered=ee368YCbCrbin(img,161.9964,-11.1051,22.9265,25.9997,4.3568,3.9479,2);

% black isolated holes rejection
filtered=bwfill(filtered,'holes');

% white isolated holes less than small_area rejection
filtered=bwareaopen(filtered,small_area*10);

% first erosion
filtered = imerode(filtered,ones(2*effect_num));

% edge detection with the Roberts method with sensitivity 0.1
edge_img=edge(gray_img,'roberts',0.1);

% final binary edge image
edge_img=~edge_img;

% integeration of two images, edge + filtered image
filtered=255*(double(filtered) & double(edge_img)); % double

% second erosion
filtered=imerode(filtered,ones(effect_num));

% black isolated holes rejection
filtered=bwfill(filtered,'hole');

% small areas less than the minumum area of face rejection
filtered=bwareaopen(filtered,min_face);
```

```
% group labeling in the filtered image
[segments, num_segments] = bwlabel(filtered);

% Based on the binary image, squared windows are generated
boxInfo = [];
for i = 1:num_segments,
   [row col] = find(segments == i);
   [ctr, hWdth] = ee368boxInfo(row, col);
   boxInfo = [boxInfo; ctr hWdth];
end

% Overlapping squares are merged
boxInfo = ee368boxMerge(boxInfo, num_segments, imgSize(1));
num_box = length(boxInfo);

% mean squared distance of the boxes wirh respect to others are calculated
boxDist = [];
for i = 1:num_box,
   boxDist = [boxDist; sqrt((sum((boxInfo(:,1) - boxInfo(i,1)).^2) + ...
         sum((boxInfo(:,2) - boxInfo(i,2)).^2))/(num_box-1))];
end

% conversion to 'double' format and 'gray' format
gdOrgImg = double(rgb2gray(uint8Img));
filtered = double(filtered);

outFaces = [];

for k=1:num_box,
   ctr = boxInfo(k, 1:2);
   hWdth = boxInfo(k,3);

   % Based on the box information, images are cut into squares
   testImg = ee368imgCut(gdOrgImg, filtered, imgSize, boxInfo(k,:));

   % normalized with an average brightness
   avgBri = sqrt(sum(sum(testImg.^2))/bwarea(testImg));
   testImg = testImg/avgBri;

   % test images are compared to the eigen images or femaale average images
   corr = ee368imgMatch(testImg, hWdth);
   fCorr = ee368imgMatchFe(testImg, hWdth);
   outFaces = [outFaces; ctr 1 corr/boxDist(k), hWdth, fCorr];
end

% sorting of the correlation values
[Y I] = sort(outFaces(:,4));
outFaces = outFaces(I, :);

% elimination of small correlation values using histogram
B = hist(Y);
if B(1) < 0.5*num_box,
```

```
   outFaces = outFaces(B(1)+1:end, :);
end

% results of the correlation with respect to the women's faces
[Fe1, ordFe1] = max(outFaces(:, 6));
[Fe2, ordFe2] = max(outFaces(:, 7));
[Fe3, ordFe3] = max(outFaces(:, 8));

outFaces = outFaces(:, 1:3);
outFaces([ordFe1, ordFe2, ordFe3],3) = 2;
```

ee368YCbCrseg.m

```
function ee368YCbCrseg
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% a function for color component analysis  %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% training image size
size_x=100;
size_y=100;

folder=['a' 'b' 'c' 'd' 'e' 'f' 'g'];
image_q=[13 17 20 14 11 19 12];

folder_num=size(image_q);

n=0;

% make a YCrCb color matrix set
for i=1:folder_num(2)
   for k = 1:image_q(i)
      if k < 10 face = imread(sprintf('testImages/traincolor_%d/%c0%d.jpg',i,folder(i),k));
      else face = imread(sprintf('testImages/traincolor_%d/%c%d.jpg',i,folder(i),k));
      end
      n=n+1;
      RGBface = double(face);
      YCbCrface = rgb2ycbcr(RGBface);
      YCbCrfaces(:,:,:,n)=YCbCrface;
   end
end

% discrimination of each component from the color matrix set
[m_YCbCr, n_YCbCr, p_YCbCr, num_YCbCr] = size(YCbCrfaces);
Y = reshape(YCbCrfaces(:,:,1,:),m_YCbCr*n_YCbCr*num_YCbCr,1);
Cb = reshape(YCbCrfaces(:,:,2,:),m_YCbCr*n_YCbCr*num_YCbCr,1);
Cr = reshape(YCbCrfaces(:,:,3,:),m_YCbCr*n_YCbCr*num_YCbCr,1);

% histogram of each component
subplot(131); hist(Y); title('histogram of Y');
subplot(132); hist(Cb); title('histogram of Cb');
subplot(133); hist(Cr); title('histogram of Cr');

clear all;
```

ee368YCbCrseg.m

```
function result=ee368YCbCrbin(RGBimage,meanY,meanCb,meanCr,stdY,stdCb,stdCr,factor)
% ee368YCbCrbin returns binary image with skin-colored area white.
%
% Example:
% result=ee368YCbCrbin(RGBimage,meanY,meanCb,meanCr,stdY,stdCb,stdCr,factor)
%   RGBimage: double formatted RGB image
%   meanY:  mean value of Y of skin color
%   meanCb: mean value of Cb of skin color
%   meanCr: mean value of Cr of skin color
%   stdY:   standard deviation of Y of skin color
%   stdCb:  standard deviation of Cb of skin color
%   stdCr:  standard deviation of Cr of skin color
%   factor: factor determines the width of the gaussian envelop.
%
% All the parameters are based on the training facial segments taken from 7 training images

YCbCrimage=rgb2ycbcr(RGBimage);

% set the range of Y,Cb,Cr
min_Cb=meanCb-stdCb*factor;
max_Cb=meanCb+stdCb*factor;
min_Cr=meanCr-stdCr*factor;
max_Cr=meanCr+stdCr*factor;
% min_Y=meanY-stdY*factor*2;

% get a desirable binary image with the acquired range
imag_row=size(YCbCrimage,1);
imag_col=size(YCbCrimage,2);

binImage=zeros(imag_row,imag_col);

Cb=zeros(imag_row,imag_col);
Cr=zeros(imag_row,imag_col);

Cb(find((YCbCrimage(:,:,2) > min_Cb) & (YCbCrimage(:,:,2) < max_Cb)))=1;
Cr(find((YCbCrimage(:,:,3) > min_Cr) & (YCbCrimage(:,:,3) < max_Cr)))=1;
binImage=255*(Cb.*Cr);

result=binImage;
```

ee368boxInfo

```
function [ctr, hWdth] = ee368boxInfo(row, col)
% Given the row and column information of the white area of a binary image,
% this function returns the value of center and width of the squared window.
%
% Example
% [ctr, hWdth] = ee368boxInfo(row, col)
%   row: row cordinates of the white pixels
%   col: column cordinates of the white pixels
%   ctr: (row cordinate of the center, column cordinate of the center)
%   hWdth: half of the size of the window

minR = min(row); maxR = max(row);
minC = min(col); maxC = max(col);

ctr = round([(minR + maxR)/2, (minC + maxC)/2]);

hStp = 5;

if (maxR-minR) > (maxC-minC),
   hWdth = round((maxR - minR)/2/hStp)*hStp;
else
   hWdth = round((maxC - minC)/2/hStp)*hStp;
end
```

ee368boxMerge

```
function boxInfo = ee368boxMerge(boxInfo, num_segments, nRow)
% Given the information of the squared windows, this function merges superpo-
% sing squares. Additioally, this function also rejects too small or large
% windows.
%
% Example:
% boxInfo = ee368boxMerge(boxInfo, num_segments, nRow)
%   boxInfo: center coordinates and half widths of the boxes
%   num_segments: number of segments or boxes
%   rRow:  total number of rows of the given image
%   boxInfo: newly generated boxInfo


rGapTh = 70;
cGapTh = 25;
hStp = 5;
rThr = 200;
adjBoxCor = [];

for i = 1: num_segments-1,
   rGap = boxInfo(i+1:end, 1) - boxInfo(i, 1);
   cGap = boxInfo(i+1:end, 2) - boxInfo(i, 2);
   rCandi = find((abs(rGap) < rGapTh) & (abs(cGap) < cGapTh));
   adjBoxCor = [adjBoxCor; i*ones(length(rCandi),1) i+rCandi];
end

numAdj = size(adjBoxCor,1);

for j = 1: numAdj,
   fstPnt = adjBoxCor(j,1);
   fstCtrR = boxInfo(fstPnt, 1);
   fstCtrC = boxInfo(fstPnt, 2);
   fstHwd = boxInfo(fstPnt, 3);

   sndPnt = adjBoxCor(j,2);
   sndCtrR = boxInfo(sndPnt, 1);
   sndCtrC = boxInfo(sndPnt, 2);
   sndHwd = boxInfo(sndPnt, 3);

   if fstCtrR-fstHwd < sndCtrR-sndHwd,
      rTop = fstCtrR-fstHwd;
   else
      rTop = sndCtrR-sndHwd;
   end

   if fstCtrR+fstHwd > sndCtrR+sndHwd,
      rBot = fstCtrR+fstHwd;
   else
```

```
      rBot = sndCtrR+fstHwd;
    end

    if fstCtrC-fstHwd < sndCtrC-sndHwd,
      cLeft = fstCtrC-fstHwd;
    else
      cLeft = sndCtrC-sndHwd;
    end

    if fstCtrC+fstHwd > sndCtrC+sndHwd,
      cRight = fstCtrC+fstHwd;
    else
      cRight = sndCtrC+sndHwd;
    end

    ctr = round([(rTop+rBot)/2, (cLeft+cRight)/2]);

    if rBot-rTop > cRight-cLeft,
      hWdth = round((rBot-rTop)/2/hStp)*hStp;
    else
      hWdth = round((cRight-cLeft)/2/hStp)*hStp;
    end

    boxInfo(sndPnt, :) = [ctr, hWdth];
    boxInfo(fstPnt, :) = [ctr, hWdth]; % added line
  end

  adjBoxCor2 = adjBoxCor;

  for i = 2: size(adjBoxCor, 1),
    if adjBoxCor(i,1) == adjBoxCor(i-1,1),
      adjBoxCor2(i,1) = adjBoxCor(i-1, 2);
    end
  end

  boxInfo(adjBoxCor2(:,1), :) = [];
  boxInfo(find(boxInfo(:,1) > nRow-rThr), :) = [];
```

ee368imgCut

```
function testImg = ee368imgCut(img, filtered, imgSize, boxInfo);
% This function returns a squared test image with a standardized image size.
% The output image is masked with a binary image so that non-facial area
% is blacked out. In addition, this program also compensates for the image
% which is adjacent to the edge of the picture to make it square formed by
% means of filling out zeros.
%
% Example
% testImg = ee368imgCut(img, filtered, imgSize, boxInfo)
%   img: double formatted test image
%   filtered: binary image which contains mask information
%   imgSize: the size of img
%   boxInfo: information of center point and width of a box
%   testImg: square shaped segment to be tested in image matching algorithm

nRow = imgSize(1);
nCol = imgSize(2);
ctr = boxInfo(1:2);
hWdth = boxInfo(3);

testImg = img(abss(ctr(1)-hWdth): abss2(ctr(1)+hWdth-1, nRow), abss(ctr(2)-hWdth): abss2(ctr(2)+hWdth-1, nCol));
maskImg = filtered(abss(ctr(1)-hWdth): abss2(ctr(1)+hWdth-1, nRow), abss(ctr(2)-hWdth): abss2(ctr(2)+hWdth-1, nCol));
testImg = testImg.*maskImg;

[nRowOut, nColOut] = size(testImg);

if ctr(1)-hWdth-1 < 0, % image is sticking out to the top
   testImg = [zeros(-ctr(1)+hWdth+1, nColOut); testImg];
elseif ctr(1)+hWdth-1 > nRow, % image is sticking out to the bottom
   testImg = [testImg; zeros(ctr(1)+hWdth-nRow-1, nColOut)];
end

if ctr(2)-hWdth -1 < 0, % image is sticking out to the left
      testImg = [zeros(2*hWdth, -ctr(2)+hWdth+1), testImg];
elseif ctr(2)+hWdth-1 > nCol, % image is sticking out to the right
      testImg = [testImg zeros(2*hWdth, ctr(2)+hWdth-nCol-1)];
end
```

ee368imgMatch

```
function corr = ee368imgMatch(testImg, hWdth)
% This function correlates a given test image with
% a reference image which has a tailored image size
% and is called from the database.
%
% Example
% corr = ee368imgMatch(testImg, hWdth)
%   testImg: square shaped test image
%   hWdth: half of the width of the testImg
%   corr: correlation value

lowThr = 30;
higThr = 220;
wdth = 2*hWdth;
if wdth < lowThr,
   corr = 0;
elseif wdth > higThr,
   corr = 0;
else
   eval(['load eigFace', num2str(wdth)]);
   corr =reshape(testImg, wdth^2, 1)'*eigFace;
end
```

ee368imgMatchFe

```
function  fCorr = ee368imgMatchFe(testImg, hWdth)
% Image matching for a female image

lowThr = 40;
higThr = 220;
wdth = 2*hWdth;
if wdth < lowThr,
   fCorr = [0 0 0];
elseif wdth > higThr,
   fCorr = [0 0 0];
else
   eval(['load fImg1_', num2str(wdth)]);
   eval(['load fImg2_', num2str(wdth)]);
   eval(['load fImg3_', num2str(wdth)]);
   fImg = [fImg1 fImg2 fImg3];
   fCorr =reshape(testImg, 1, wdth^2)*fImg;
end
```

abss. m

```
function val=abss(inp);
% Function 'abss' prevents the coordinate of test image
% exceeds the boundary of the original image

if(inp>1)
   val=inp;
else
   val=1;
end
```

abss2. m

```
function val=abss2(inp, thr);
% Function 'abss' prevents the coordinate of test image
% exceeds the boundary of the original image

if(inp>thr)
   val=thr;
else
   val=inp;
end
```