**CS3330 REPORT**

**1. Background**

Eyes detection:

The eyes are particularly stable feature of the human face. Therefore, it's useful to extract the eyes then use its position to extract other features of the human face.

Eyes detection in human face images plays an important part in facial recognition. Accurately extracting the eyes region help us performing normalisation of the human face (i.e., facial features alignment), which is a crucial pre-processing step of various face recognition techniques.

**Addressing eyes detection with image processing techniques**

The eyes have many different features that distinguish it from other feature. Many researchers make use of the colour difference of the eyes region to extract it from other part of the human face. In this report, we go through a method with a different approach that is extracting eyes region base on its high edge densities

**Related work**

There are several existing approaches for extracting the location of the eyes.

Many of modern face detection method are formed based Viola-Jones object detection framework [1], which based on series of Haar like features. Haar cascades are the combination of multiple weak classifiers. The downside of this algorithm is that it requires a learning process which can be time consuming and large amount of features.

Rajpathak [2] was able to detect the face's skin region by performing six-sigma operation on mixture of NTSC, HSV and RGB spaces. When an eye is well illuminated, it has a very sharp reflection point .Because of this fact; morphological operations had been used to detect the region between the two eyes. Having to use three colour space models, a high level of complexity is the main limitation of this method.

The third algorithm that I have looked into was proposed by T.Kawaguchi [3] which detects eyes with circular Hough transformation. In order for this technique to work, the size of the eye's pupil has to be given. Slow and expensive computationally due to having to perform circle fitting was the major disadvantage of this algorithm.

**2. Algorithm**

The algorithm I have chosen is relatively simple and quite fast compared to the other methods. This method works effectively for all different skin colours because it does not base on skin colour and works on facial image of all ethnics of people [3].

**Limitation of the algorithm**

There are some drawbacks to this algorithm. One of the weaknesses of this algorithm is that eyes regions are not extracted accurately when performed on images with varying sizes. Moreover, the original image has to be a frontal view head-shoulder image with both of the person's eyes visible and opened and their face tilts to the sides no more than 45.0 degrees.

**How the algorithm works**

1. Convert image to gray scale image

The image is first converted to a grey image. This step is compulsory because edge detection does not work on colour images.

2. Detecting edges

In this step, we apply Sobel operator on the gray scale image to detect the edges. Performing edge detection helps separate the person's face from the image's background. Figure 1 shows the binary image with the edges detected.

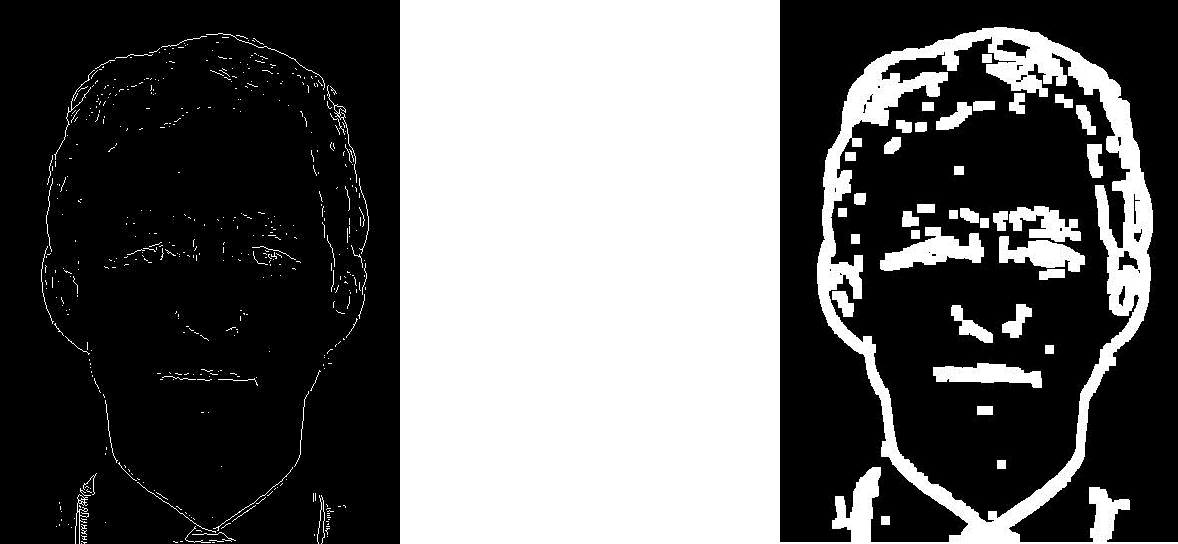


(a) (b)

Figure (a) Gray scale image (b) Edge detected image

3. Dilate the edged image twice

The edge image is then dilated to improve the eyes region. The structuring element we used was a disk shape element with the radius of 3.



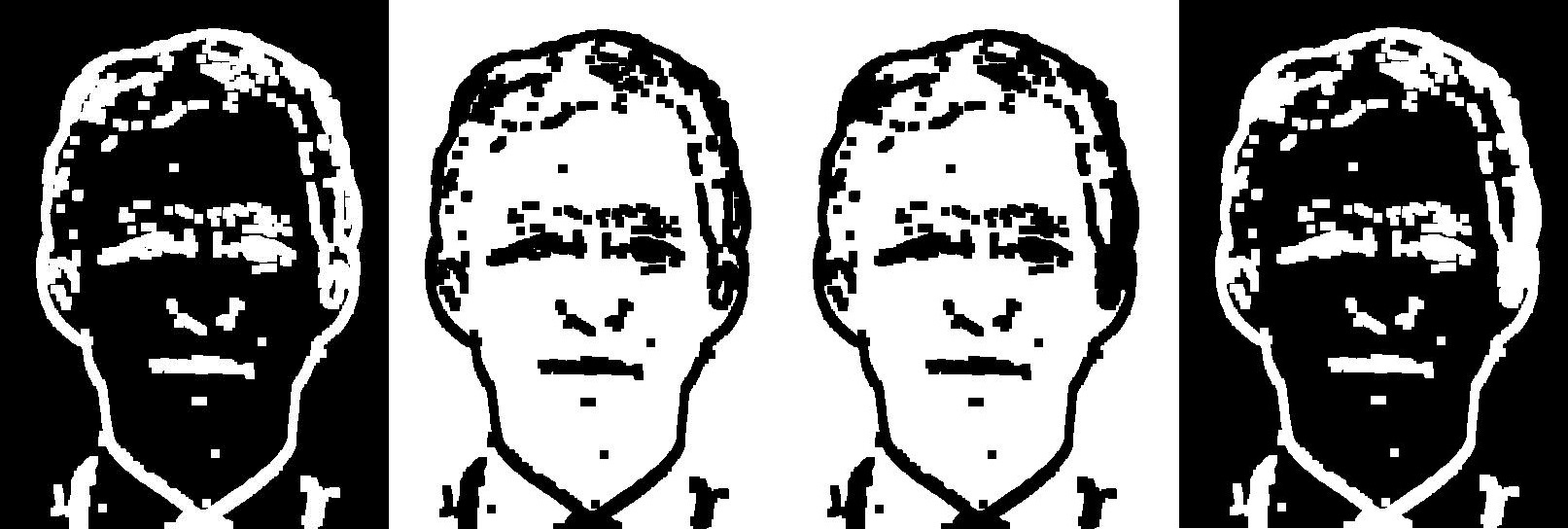
(a) (b)

Figure (a) Edged image (b) Dilated image

4. Fill small holes in the eyes region

The dilated image is going to be eroded to extract the blobs. Small holes inside the eyes region can be destructive when we perform erosion on the image. To remove these small holes we first take the complement of the dilated image. In the complement of the dilated image, we remove the connected components that have area less than 300. We then take the negative of the image again.

Figure 3 illustrates how the holes filling process work.



(a) (b) (c) (d)

C:\Users\Thai Con\AppData\Local\Microsoft\Windows\INetCache\Content.Word\right eyse holes compare.jpg

(e) (f)

Figure 3 Holes Filling (a) Dilated image with holes (b) complement  of the dilated image (c) Holes filled by remove small component (d) Inverse the image back (e) Eye with small holes (f) Eyes with holes filled

5. Erosion

After dilating the image, the next step in our algorithm is to erode the image three times to remove redundant edges that were enhanced during the dilation and holes filling phase.



(a) (b)

Figure 4 (a) Dilated image (b) Eroded image

6. Remove small components

I added one more step to the original algorithm. In this step we simply remove components that have areas less than 120. I found this necessary because these small components can be easily mistaken as eyes regions by the algorithm.

7. Extracting eyes region by applying rules

After the blobs extraction phase, we extract the eyes by finding the components that satisfies all the rules in this set of rules:

- The width over height ratio is greater than 0.8 and less than 4.0.

- The component's orientation is less than 45.0.

- One eye must not be three times larger than another eye.

- The orientation difference between the two eyes is less than 30.0

- The angle between the line joining the two eyes and the horizontal axe is less than 15.0

- The eyes are far enough from the image's border

**3. EXPERIMENTAL RESULTS**

I tested my algorithm on PICS [6] facial images database. 90 random facial photos were chosen including both females and males of more than 10 different individuals.

The results show that the blobs extraction phase has a high success rate up to 96% while the eyes extracting process needs to be further improved with the success rate of 76%.

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Total photos | Eyes not extracted | accuracy |
| Blobs extraction | 90 | 4 | 96% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Total photos | Eyes not detected | Detected non-eyes region | accuracy |
| Eyes detection | 90 | 14 | 12 | 71% |

**4. Conclusions**

In conclusion, the chosen algorithm is very good at the extract to initial blobs but needs some improve in the eyes extracting phase.

During my research, I could only cover three alternative methods to compare it with my chosen algorithm. Another limitation of my research was that I was not able implement the other mentioned methods to test them on my data in order to show more the advantage and disadvantage of the methods.

function [ eyes\_detected\_img ] = eyes\_detect( face\_img )  
%EYES\_DETECT Detect eyes region then draw box around them  
  
%convert image to gray image  
 gray\_img = rgb2gray(face\_img);  
%Edge detection with sobel  
 edge\_img = edge(gray\_img,'sobel');  
% Dilate the image twice  
 dilated\_img = dilate(edge\_img);  
%Inverse the image then fill the small holes  
 negative\_dilated\_img = imcomplement(dilated\_img);  
 filled\_img = imcomplement(rm\_small\_cmp(negative\_dilated\_img, 300));  
%Erode the image three times  
 eroded\_img = erode(filled\_img);  
  
%Aspect ratio rule  
 aspect\_img = aspect\_ratio\_rule(eroded\_img);  
  
%The orientation angle of eyes is not greater than 45 degrees.  
 angle\_img = rm\_large\_orient(aspect\_img,100.0);  
  
%Remove small components  
 rm\_img = rm\_small\_cmp(angle\_img, 120);  
  
%Apply rules that compare two eyes  
 hpadding = 40.0;  
 vpadding = 80.0;  
 eyes\_ratio = 3.0;  
 max\_orient\_diff = 30.0;  
 eyes\_slope\_angle = 15.0;  
 eyes\_binary\_img = two\_eyes\_rule(rm\_img,hpadding,vpadding,eyes\_ratio, max\_orient\_diff, eyes\_slope\_angle);  
%Draw box around the eyes  
 eyes\_detected\_img = draw\_box(eyes\_binary\_img, face\_img );  
% eyes\_detected\_img = dilated\_img;  
  
end

function [eyes\_binary\_img] = two\_eyes\_rule(img,hpadding,vpadding,eyes\_ratio, max\_orient\_diff, eyes\_slope\_angle)  
 CC = bwconncomp(img,4);  
 stats = regionprops(CC,'all');  
  
 idx\_size = [];  
 [y,x] = size(img);  
for i1 = 1 : length(stats)  
 comp1 = stats(i1);  
for i2 = 1:length(stats)  
if i1 == i2  
continue;  
end  
 comp2 = stats(i2);  
  
 far\_from\_border = far\_from\_border\_rule( comp1,comp2, hpadding,vpadding,x,y);  
 ratio\_match = ratio\_rule (comp1, comp2, eyes\_ratio);  
 orient\_match = orient\_rule (comp1, comp2, max\_orient\_diff);  
 slope\_angle\_match = slope\_angle\_rule (comp1, comp2 ,eyes\_slope\_angle);  
if far\_from\_border && ratio\_match && orient\_match && slope\_angle\_match  
 idx\_size = [idx\_size i1 i2];  
break;  
end  
end  
end  
 eyes\_binary\_img = ismember(labelmatrix(CC), idx\_size);  
end

function [ far\_from\_border ] = far\_from\_border\_rule( comp1,comp2 ,hpadding,vpadding,x,y)  
 bb1 = comp1.BoundingBox;  
 far\_from\_border1 = hpadding<bb1(1) && bb1(1)+bb1(3)<x-hpadding && vpadding<bb1(2)&& bb1(2)+bb1(4)<y-vpadding;  
 bb2 = comp2.BoundingBox;  
 far\_from\_border2 = hpadding<bb2(1) && bb2(1)+bb2(3)<x-hpadding && vpadding<bb2(2)&& bb2(2)+bb2(4)<y-vpadding;  
 far\_from\_border = far\_from\_border1 && far\_from\_border2;  
end

function [ratio\_match] = ratio\_rule (comp1, comp2, eyes\_ratio)  
 ratio = comp1.Area/comp2.Area;  
 ratio\_match = ratio > 1/eyes\_ratio && ratio < eyes\_ratio;  
end

function [orient\_match] = orient\_rule (comp1, comp2, max\_orient\_diff)  
 orient\_match = comp1.Orientation - comp2.Orientation < max\_orient\_diff;  
end

function [slope\_angle\_match] = slope\_angle\_rule (comp1, comp2, eyes\_slope\_angle)  
 bb1 = comp1.BoundingBox;  
 bb2 = comp2.BoundingBox;  
 center1 = [bb1(1)+bb1(3)/2,bb1(2)+bb1(4)/2 ];  
 center2 = [bb2(1)+bb2(3)/2, bb2(2)+bb2(4)/2];  
 slope\_angle = atan2(center2(2)-center1(2),center2(1)-center1(1))\* 180/pi;  
 slope\_angle = abs(slope\_angle);  
if slope\_angle > 90.0  
 slope\_angle = 180.0 - slope\_angle;  
end  
 slope\_angle\_match = slope\_angle < eyes\_slope\_angle;  
end

function [ aspect\_ratio\_img ] = aspect\_ratio\_rule( img )  
% aspect\_ratio: Keep components that have 0.8 < w/h < 4.0  
  
 CC = bwconncomp(img,4);  
 stats = regionprops(CC,'all');  
%Aspect ratio rule  
 idx\_boundingbox = [];  
for k = 1 : length(stats)  
 BB = stats(k).BoundingBox;  
 aspect\_ratio = BB(3)/BB(4);  
if aspect\_ratio > 0.8 && aspect\_ratio < 4.0  
 idx\_boundingbox = [idx\_boundingbox k];  
end  
end  
  
 aspect\_ratio\_img = ismember(labelmatrix(CC), idx\_boundingbox);  
  
end

function [ dilated\_img ] = dilate( edge\_img )  
%dilate Dilate img twice  
 SE = strel('disk', 3);  
 dilated\_img = imdilate(edge\_img,SE);  
 dilated\_img = imdilate(dilated\_img,SE);  
  
end

function [ eroded\_img ] = erode( filled\_img )  
%erode Erode image three times  
  
 SE = strel('disk', 3);  
 eroded\_img = imerode(filled\_img,SE);  
 eroded\_img = imerode(eroded\_img,SE);  
 eroded\_img = imerode(eroded\_img,SE);  
  
end

function [ angle\_img ] = rm\_large\_orient( img, max\_orientation )  
%rm\_large\_orient: Remove component with orientation larger than a choosen  
%angle  
  
CC = bwconncomp(img,4);  
stats = regionprops(CC,'Orientation');  
idx = find([stats.Orientation] <= max\_orientation);  
angle\_img = ismember(labelmatrix(CC), idx);  
% figure(),imshow(angle\_img),title('angle');  
  
end

function [ rm\_img ] = rm\_small\_cmp( img, min\_area )  
%rm\_small\_cmp remove components that are small than min\_area  
% Detailed explanation goes here min\_are rm2 = 120  
 CC = bwconncomp(img,4);  
 stats = regionprops(CC,'Area');  
 idx = find([stats.Area] > min\_area);  
 rm\_img = ismember(labelmatrix(CC), idx);  
end

function [ eyes\_detected\_img ] = draw\_box( final\_binary\_img, original\_img )  
%draw\_box: Draw box around the eyes region  
% Detailed explanation goes here  
  
CC = bwconncomp(final\_binary\_img,4); %final image  
stats = regionprops(CC,'BoundingBox');  
eyes\_detected\_img = original\_img;  
for k = 1 : length(stats)  
 BB = stats(k).BoundingBox;  
 f = @() rectangle('Position', [BB(1),BB(2),BB(3),BB(4)]);  
 params = {{'EdgeColor','r','LineWidth',2}};  
 eyes\_detected\_img = insertInImage(eyes\_detected\_img,f,params);  
% rectangle('Position', [BB(1),BB(2),BB(3),BB(4)], 'EdgeColor','r','LineWidth',2 );  
end  
  
end

function [blob\_extracted] = blob\_extract (img, edge\_detect\_type)  
% blob\_extract: Extract blob before applying rules  
grey\_img = rgb2gray(img);  
edge\_img = edge(grey\_img,edge\_detect\_type);  
SE = strel('disk', 3);  
%Dilate image twice  
dilated\_img = imdilate(edge\_img,SE);  
dilated\_img = imdilate(dilated\_img,SE);  
%Inverse the dilated img to fill holes  
negative\_dilated\_img = imcomplement(dilated\_img);  
CC = bwconncomp(negative\_dilated\_img,4);  
stats\_first = regionprops(CC,'Area');  
idx = find([stats\_first.Area] > 300);  
ne\_img\_with\_holes = ismember(labelmatrix(CC), idx);  
filled\_img = imcomplement(ne\_img\_with\_holes);  
%Erode image three times  
eroded\_img = imerode(filled\_img,SE);  
eroded\_img = imerode(eroded\_img,SE);  
eroded\_img = imerode(eroded\_img,SE);  
blob\_extracted = eroded\_img;  
end

**References**

[1] Kawaguchi, T. and Rizon, M. (2003). Iris detection using intensity and edge information. *Pattern Recognition*, 36(2), pp.549-562.

[2] Rajpathak, T., Kumar, R. and Schwartz, E. (2009). Eye detection using morphological and color image processing. *Proceeding of Florida Conference on Recent Advances in Robotics*, pp.1-6.

[3] Shafi, M. and Chung, P. (2008). Eyes extraction from facial images using edge density”. *7th IEEE International Conference on Cybernetic Intelligent Systems (CIS)*, pp.1-6.

[4] Viola, P. and Jones, M. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2), pp.137-154.

[5] Singh, A., Singh, M. and Singh, B. (2016). Face detection and Eyes extraction using Sobel Edge Detection and Morphological Operations. *2016 Conference on Advances in Signal Processing (CASP)*

[6] PICS, 2003. Psychological Image Collection at Stirling (PICS image data base). Available from , University of Stirling Psychology Department.