Face Based Indexing of Digital Photo Albums

Matt Veino
Computer Science Thesis
May, 2004

Advisor: Professor David Martin

I. Abstract

Managing a large collection of digital snapshots -- even for the average amateur digital photographer -- is a chore. Searching is often limited to captions, but writing a descriptive caption for each photo is overly tedious. We aim to simplify the management of a collection of photos by enabling searches on the content of the photographs themselves while minimizing the amount of supervision by the user. We focus on the problem of finding faces in photographs. The user identifies a small number of faces in the photo collection. Once we have these faces, we can run algorithms on other pictures to search through and find faces in the photographs. Once we have locations of faces, we will attempt to recognize the individuals.

II. Introduction

Computer Vision has very interesting applications when it comes to research and other technologies, but doesn't apply much yet when it comes to the typical user's desktop computer. Yet with the sudden emergence of digital photography, this research can be integrated into an average user's desktop in many ways. One such way is to recognize who is in a particular photograph. Because most users will be taking pictures of the same groups of people, we should have enough information to create a system to match certain faces to other faces.

This technology has gained national attention due to sudden focus of terrorism. Face recognition devices have been tested in airports to detect whether criminals on wanted lists can be picked out of a crowd. A CNN article written a few weeks after the 9/11 terrorist attacks claims, "The technology is so precise, that it can't be fooled by

disguises such as wigs or fake beards."[1] However, tests in Boston's Logan Airport "failed miserably" when it came to detecting terrorists (or, in the test cases, individuals who were supposed to be recognized by the system). When the system was set to be too sensitive, it would "recognize" various objects in the scene such as plants and dogs. And when the system was set to be too precise, it could be fooled by different lighting conditions or by an individual simply wearing a hat or a pair of glasses.[2] Therefore in the realm of security, face recognition simply cannot work since one cannot sufficiently control pose and illumination. However, in a typical user's photo album, people's appearances stay somewhat constant. Usually the subjects are posed, smiling and looking directly at the camera. Therefore we hope to use these constant conditions to our advantage.

To reach the goal of automatically recognizing someone in a photograph there are many steps. First, the system must know where the faces are. To automatically do this, we must train the system to know what a face looks like. Since faces all look fairly similar, training the program to recognize the shape of a face should be possible. This is where the majority of the processing power (as well as my research) will take place, since the process of recognizing a face doesn't take many computations.

Also, by using a manually entered set of faces, a clustering algorithm will be run on the set to determine which faces are the same in the photographs. Once these clusters are obtained, the user can manually enter a name for each of the clusters. Once we have the location of the faces in a new image, we can look up each face and hopefully

[1] "Face Recognition May Enhance Airport Security", http://www.cnn.com/2001/US/09/28/rec.airport.facial.screening/
[2] "Face Recognition Fails in Boston Airport", http://www.theregister.co.uk/2002/07/20/face_recognition_fails_in_boston/

recognize which cluster each belongs to. We could then write this information into a central database which can be queried to return photographs of particular individuals.

III. Training and Finding Faces in the Images

a. Obtaining Training Images

I started with my own collection of over six thousand digital images. Out of these, I chose 200 photographs I felt would be suitable to training a face finding and recognizing algorithm (with the individuals smiling, facing the camera, etc.). This brings up the first problem: not all photographs have faces in them. Out of my six thousand photographs, less than 25% contained people in them. If an algorithm were not trained correctly, it might find faces in these 75% of photographs, which would throw off the system.

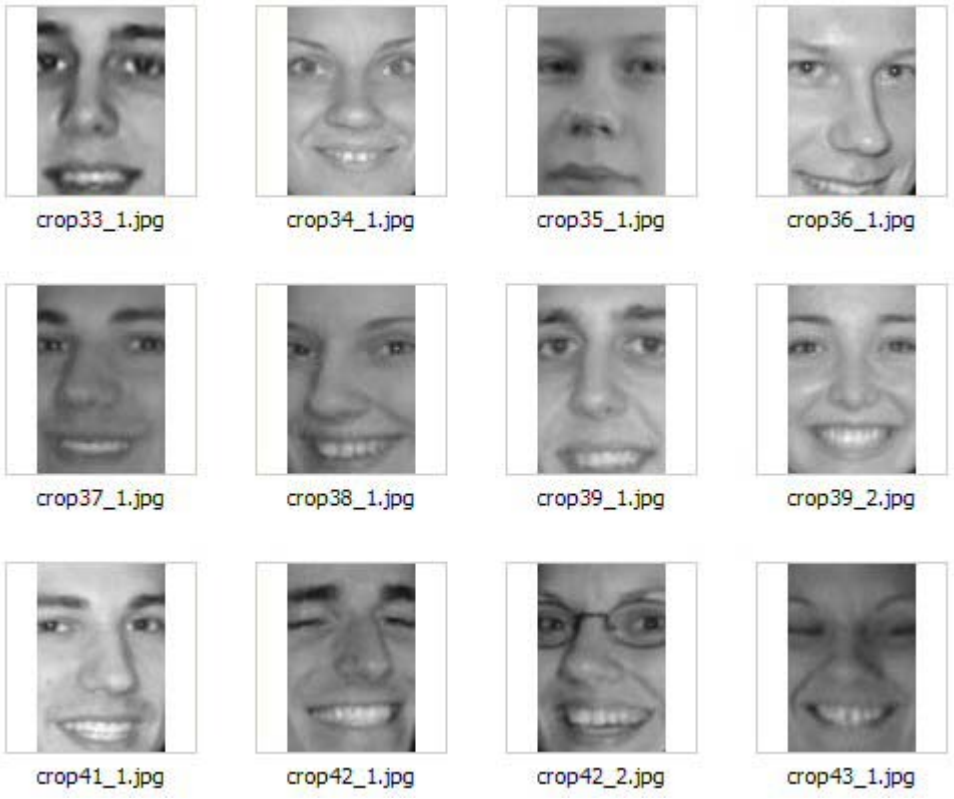
To train the system to know what a face looks like, I manually annotated the locations of faces within the images. I randomly numbered the files from one to two hundred. I then created a file (facialpoints.m) to read through the first one hundred images and show them on the screen. For each face, I would click the outside corner of the eye to the left, then to the right, and then on the tip of the person's nose. I chose these landmarks because a lot of information (size of the face, orientation of the face, etc.) can be determined from the distances and angles between the points.

Once we have the locations of the eyes and nose for each face in the photograph, we must create an image of each face. I wrote a function to read in the annotation information and do this for us. The original image is read in as well as the locations of the landmarks in the image. First, however, we must pad the image in case our calculations will push the face bounding box outside the original area. To determine how

large the face is, the distance between the eyes is calculated. Our first crop is slightly larger than the actual face, because another calculation must be determined: Since we want all the eyes to be in the same position, we must rotate the image based on the angle between the two eyes. By taking the arc Tan of the height and width difference we can determine the angle, and rotate accordingly. Once we do this, we can crop tighter based on the midpoint of the eyes and the distance between the eyes. The image is then resized to 120 by 80 (or any size of ratio 3:2) and written in the format crop_imagenumber_facenumber.jpg. Therefore, for each filename we have access to which picture it was in and what number face it was. This creates images that look like the following:

Original:

Faces:



Although these images are in black and white, masks are also created for the images which show where the faces occur in each image. Therefore, if we need access to each face in color, we can go through the original color pictures and look to the mask to determine if the location is a face pixel.

Once we have these images the path breaks off into two directions. We will now determine the "faceness" of a particular pixel based on a combination of color and eigenface reconstruction.

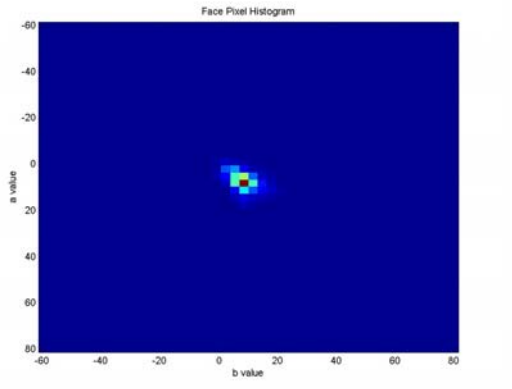
b) Faceness Based on Color

While color is not always present as a cue, the problem is different enough that we must use any cue available. While it can't distinguish a face from other skin, color is good at distinguishing skin from other objects in a picture. So in combination with other cues selective to the geometry of a face, we should be able to find our faces.

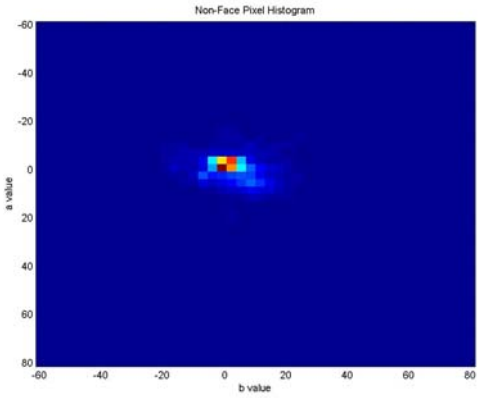
To begin, I wrote a function (getColorHist.m) that scans through all the images and creates a histogram for each channel in the image. The image is first converted from an RGB color space to Lab color space. Lab color space is beneficial to our goal because by ignoring the L channel, most of the color information (the hue, essentially) is represented in 2 channels. The L channel represents the brightness for each pixel, so by getting rid of it we can fix problems such as shadows. Therefore, it is the a and b channels that we are concerned with when it comes to the histograms.

A separate histogram is created for where a face occurs and where a face does not occur. The function scans through the image and checks with its respective mask whether the pixel in question is a face pixel or not. Then, the a and b values are converted into bin indices based on the number of bins and my pre-calculated maximum and minimum values for a and b. The bin in the corresponding histogram is then incremented. After running, we are left with the following histograms:

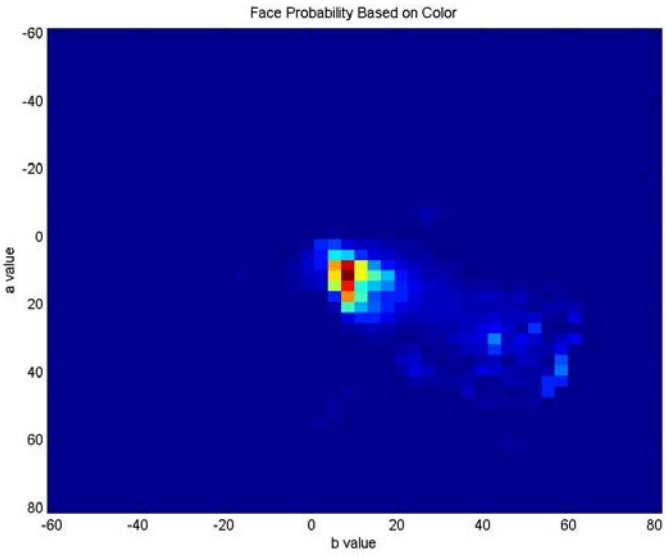
Faces:



Non Faces:

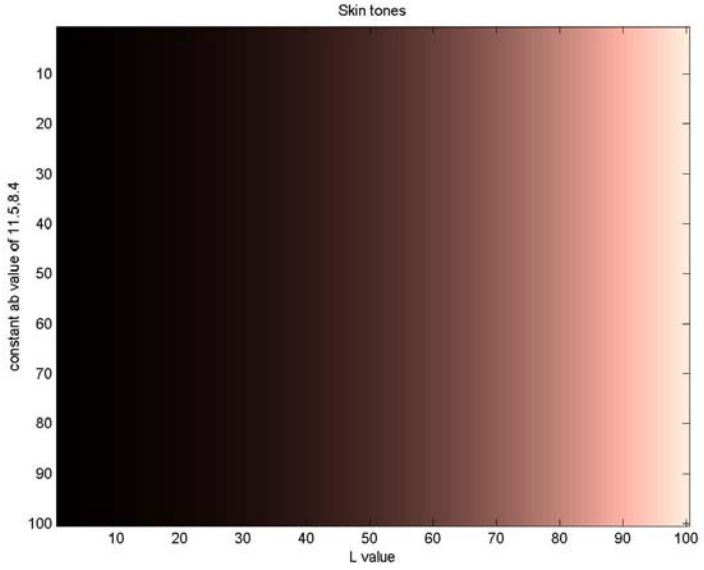


As we can see, the outcome from the histograms is favorable because the majority of the face pixels are located in a separate bin from the non face pixels. Therefore, we can come up with a probability matrix simply by dividing the face histogram by the sum of the non face and the face histograms. This results in the following probability matrix:

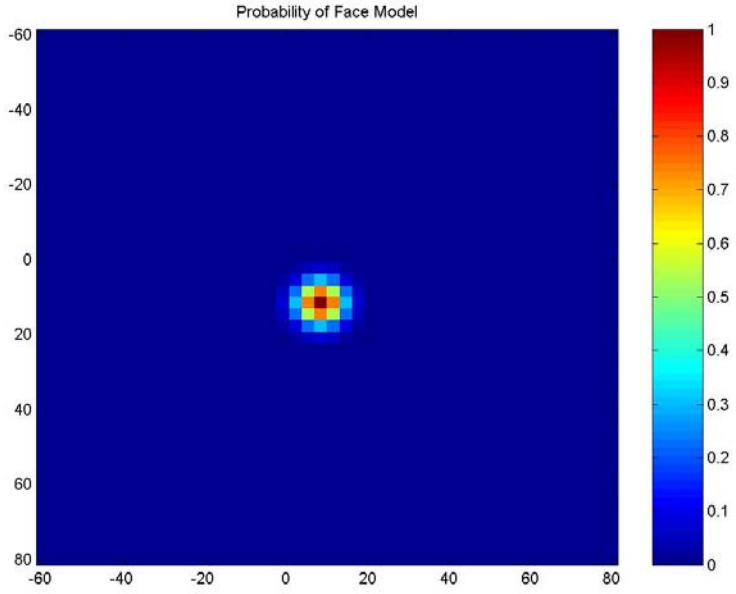


We can model this distribution with a single parametric model because the majority of the face pixels are located in one bin. With the bin indices I was able to determine that the majority of the face pixels had an a value of 11.5 and a b value of 8.4.

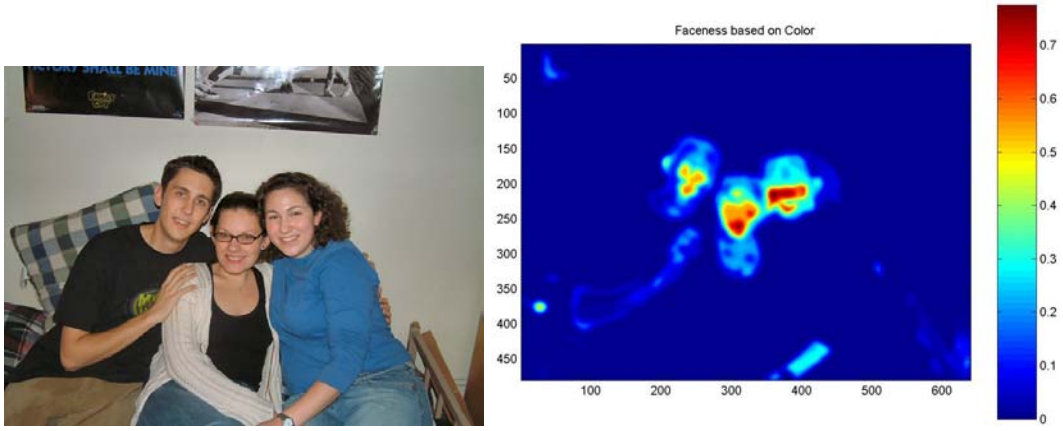
A plot of this a and b value with L values ranging from zero to one hundred from left to right shows the color at this peak as being skin toned:



With these values we are able to create a Gaussian model based on the distance from these a and b values. We are left with the following parametric model which we will directly use to determine the probability a particular point is a face:



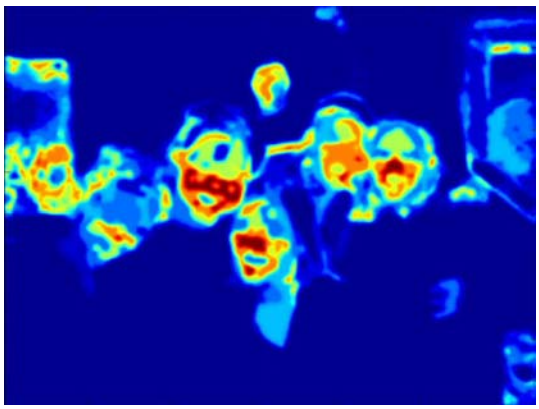
At this point, a faceness map based on color can be obtained for any image. The function I created (colorFaceness.m) converts the image to Lab color space, and using the same values for the min, max, and number of bins, converts the a and b values into bin indices. These indices are looked up in the parametric faceness model, and that particular value is set for that pixel. After we go through all the pixels, we're left with an image that expresses the faceness for each area of the picture. The following is an example of a color faceness map:



In this figure, the dark blue areas signify areas where there is no chance of a face occurring. Essentially, the function finds areas where a skin tone occurs. This information is very valuable when we want to find our face, because it wipes out a lot of the areas where a face could occur. However, this function alone cannot determine where a face occurs because the colors on someone's face aren't much different than the rest of an individual's skin. Therefore, we will use a combination of color faceness and another measure of faceness to come up with a faceness map. The example below shows an example in which the function finds the correct colors (in the mid to upper left area of the photograph), but the area does not contain skin.

Original:

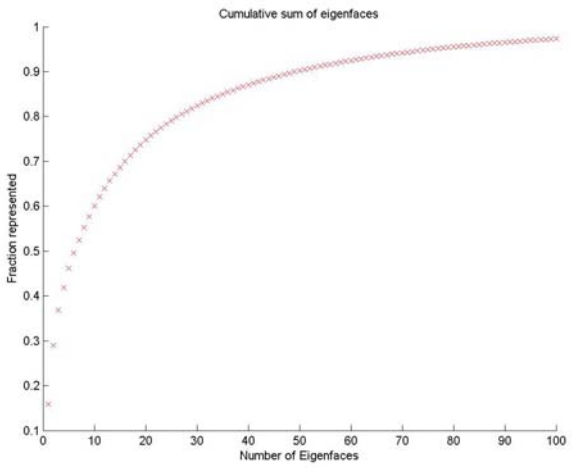
Color map:



c) Faceness based on Eigenface Reconstruction

Another measure of faceness I worked with was eigenface reconstruction. To begin, each face (obtained from part a) is reshaped to be one long horizontal vector, L1 normalized, and placed in a large matrix by the function faceMatrix.m. We normalize each face to make it so the values are in the same range of data so they can later be compared correctly. The matrix created by this function is of size m by n where m is the number of faces and n is the width times the height that is passed into the function.

From here, this matrix is passed into the function PCA.m (principle component analysis). This function returns the eigenfaces and the mean face. The mean face is simply the average of all the faces in the matrix. Each eigenface is a fraction of all the other images that went into the original function. The eigenfaces represent variation from the mean face, so by using a combination of each eigenface, a particular face that was in the original training set can be recreated perfectly if all the eigenfaces are used. The reason this is useful is because the majority of the variation can be represented by the first twenty or so eigenfaces, as shown by the figure below.



Eighty percent of the variation in the images is represented by the first twenty eigenfaces. Therefore, each face can be represented by twenty variables (which we'll refer to as face space coordinates) instead of n variables, with n being the width of the image times its height. This topic will arise again later on when we want to recognize a particular face, but for now, what we want is to see whether a particular portion of the image is a face or not. The following is what the mean face and eigenfaces look like, with the upper left face being the mean face and the eigenfaces increasing in number down the rows to the right:

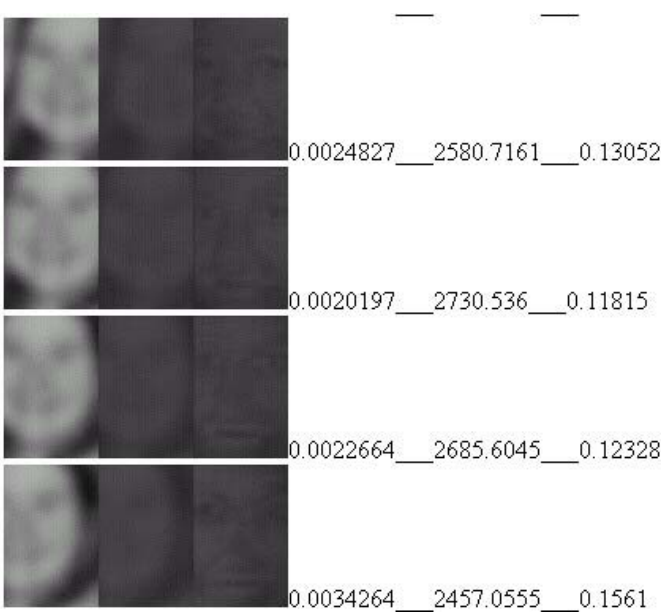
To obtain a faceness map we go through each pixel much like we did for the color faceness map. However, essentially what we want to do is try to recreate a portion of the image around each pixel. Therefore, we first crop out an area around each pixel. This information is sent into a function (faceVal.m) that first reshapes the cropped selection to be one long vector, and subtracts the mean face off of the image. The dot product is taken with the eigenfaces which maps the selection into face space and represents the portion by twenty variables. Then, the original portion is recreated by multiplying the face space coordinates by the eigenfaces.

Since the eigenfaces were created from images of faces, any recreation using the eigenfaces will look like a face as well. Therefore, a recreation of a chair, for example, will look like a face, not a chair. This is the basis of our eigenface faceness algorithm. We obtain a numerical value for the difference between the original image sent to faceVal and the recreated image from the eigenfaces. This value should be small (close to zero) if the input is a face, because the output will look like a face as well.

However, a problem arose when viewing the data returned by a simple distance between the two values. Many places besides the faces had a high degree of "faceness" when compared to the rest of the image. Yet, this makes some sense; for instance, a dark box is still located in face space, it's essentially just a face in the dark! Therefore, the standard deviation for each of the images is calculated as well to make sure there is some variation in the image. In our algorithm, I'm assuming that a "face" in which there is little to no variation cannot be a face.

Pictured below are sequential bounding boxes as they pass through the image, and their corresponding distances (first number) and standard deviations (last number). The reason the normalized and recreated images are so dark is that each face is L1

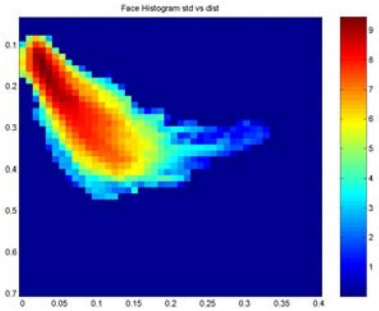
normalized, meaning the pixel values in each image add up to one. As can be seen, the more facelike a particular bounding box, the smaller the distance between the recreated image (third picture per row) and the normalized original image (second picture per row).



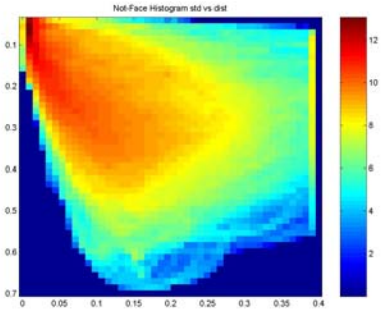
As can be seen by the distances, the image in the second row is much more facelike than the other bounding boxes which aren't centered on the face. Therefore, the distance in the second row is the least out of these four examples.

We are now faced with the task of constructing a faceness function based on two cues: the face space distance and the image variance. I created histograms for faces and non faces of the standard deviation and the distance between the points. While the results weren't as disjoint as the color histograms, it gave us valuable information because a majority of faces were located in a specific distance versus standard deviation bin. Located below are figures of the log of the histograms.

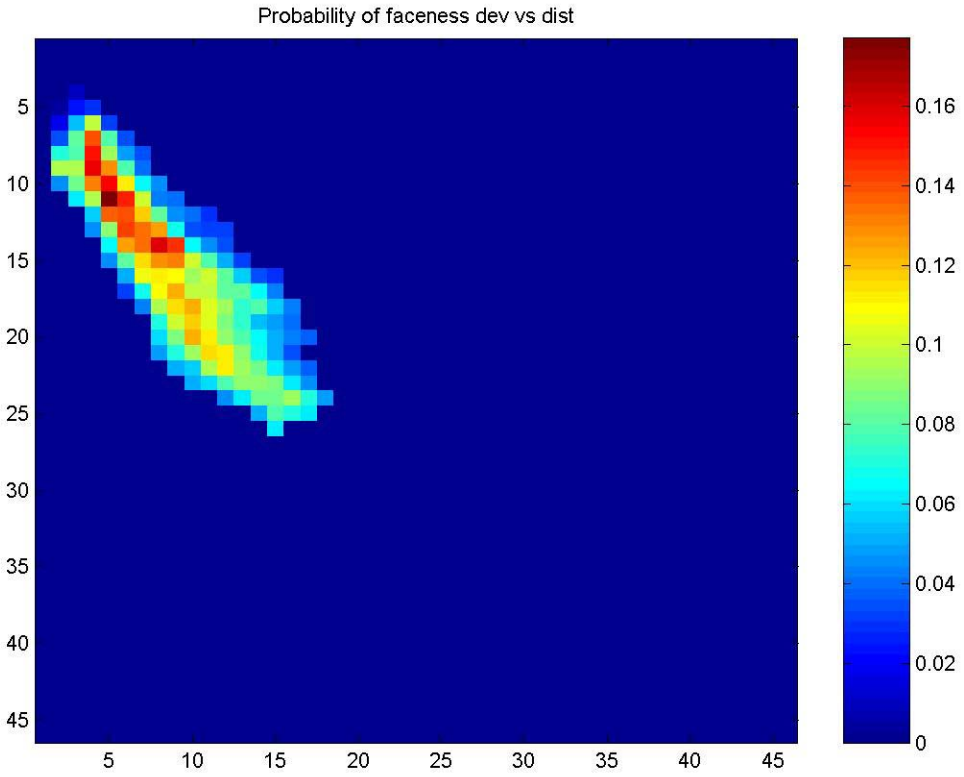
Deviation vs. Distance for Faces:



Deviation vs. Distance for Non Faces:

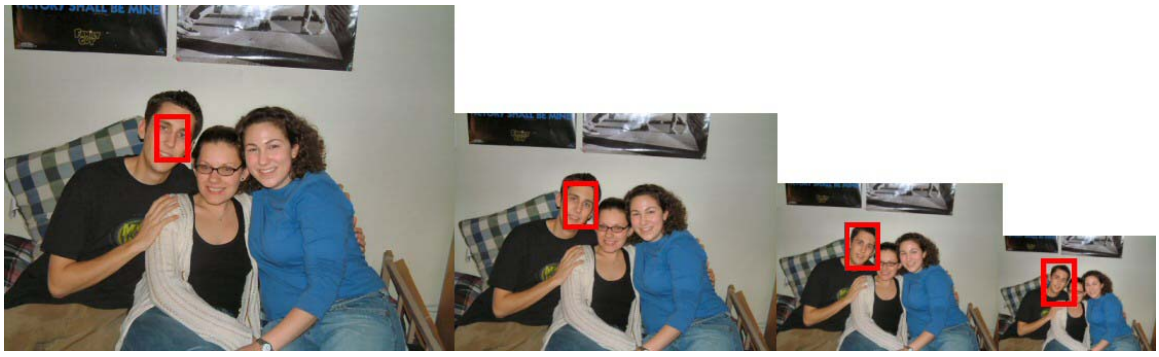


From these two matrices, a probability matrix can be composed by dividing the face histogram by the sum of the face and non face histograms. The following probability matrix is created, which we can pass into a function and look up values much like we did for the color faceness map.



Since the majority of faces don't fall in one bin, we will use this non-parametric model to determine faceness. Now we can look up faceness probabilities based on eigenfaces much like we did for the color eigenfaces. By calculating the standard deviation of the original cropped image and the distance from the recreated image to the original, we can look up a value in our faceness model and assign a particular value.

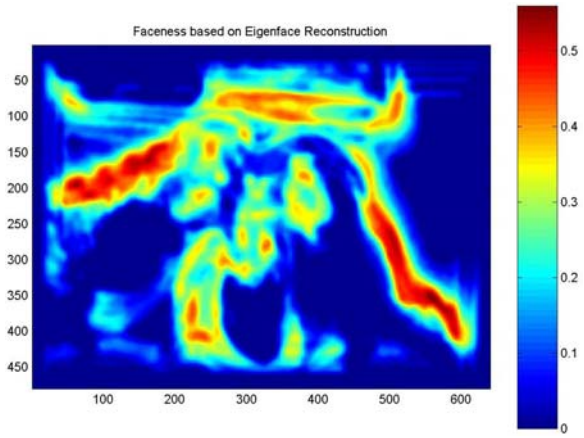
Yet, this brings up another problem with the eigenface method of finding faces. Faces can be many different sizes in pictures, depending on the size of the image, how close the person is to the camera, the age of the individual, etc. Therefore, we must search for many different size faces. To do this, we will run the algorithm on an image pyramid of the original image. The algorithm will first resize the image to 400 x 300 (assuming a 4:3 ratio as most digital photographs have). It will run the algorithm on this size, and create a faceness map. Then, the image will be resized by a factor of one half octave. At this point, our bounding box covers a larger portion of the image and as a result will search for faces that are larger than the original run through. An example of this is shown below.



The faceness will be calculated for each size, and then resized to the size of the original and the values added together. This will be done until the image is a sufficiently

small size (resizing five times will reduce a 400x300 image to 80x60) at which there could be no faces the size of the bounding box.

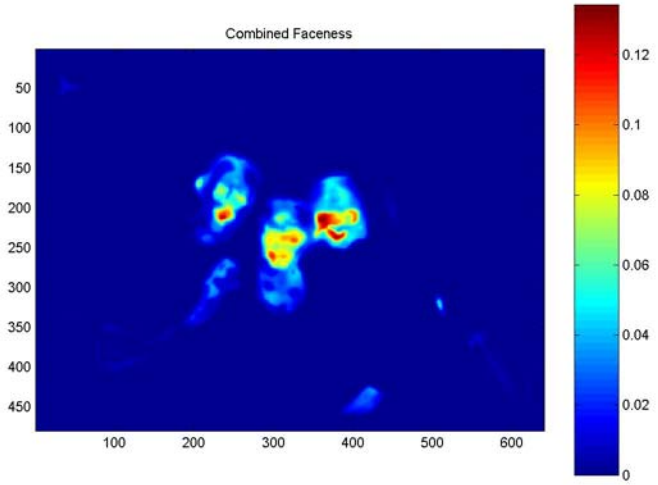
At the end of this function, we are left with a faceness map based on eigenface reconstruction. There should be peaks in places in which faces occur. The following is an example of a faceness map based on eigenface reconstruction.



d) Creating a Combined Faceness Map

Once we have these two faceness maps, combining them is done via multiplication. This simple calculation will work because of what we wanted from each of the faceness maps. The color faceness map should find areas of skintone, while the eigenface faceness map should find in which the geometry is similar to a face. A multiplication of these two will create high values in areas with high eigenface faceness and color faceness, which will hopefully be a face. The reasoning for combining the two maps is to get rid of false positives left by either the color facemap (other areas of skin) or the eigenface facemap (which is quite unreliable). False positives are expected in both cases, but hopefully there will be few misses. The color and the geometry should be disjoint cues, therefore the result of their multiplication will cancel out any false positives

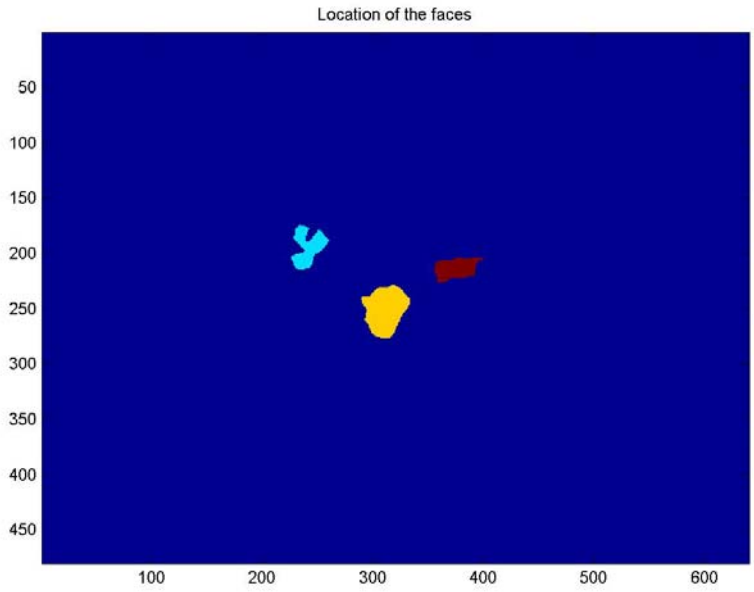
returned by either. The result is an image that looks like the following, with the peaks of color representing what we ultimately want, our faces:



e) Finding the Faces from the Facemap Matrix

There are still some areas of color where faces don't occur. To correct this, I looked at the values and obtained a numerical value for what I believed separated faces from non faces. I wrote the function findFaces.m, which has the faceness matrix as its only input. It turns the image into a binary image whose values depend on the pre determined threshold. Once the binary image is obtained, it is passed into a Matlab function called bwmorph which cleans up the binary image and fills in holes. Then, the matrix is passed into another Matlab function called bwlabel, which finds connected components and assigns them all the same value depending on what group a pixel is in. Then, I run the image through a threshold yet again, getting rid of any small groups that probably aren't faces. Ultimately, findFaces will return a matrix with values ranging from zero to the number of faces in the image. Areas where the matrix is one are areas

where the first face occurs, areas where the matrix is two is where the second face occurs, etc. The following is an example of what is returned by findFaces:



At this point, for each of these groups we can find a center point of mass, which will should be the center of the face. For each group, an average x and y value is determined, and the center of mass will be the center of the face. We now should have the location of all the faces in the image, and can then crop out these portions for use in recognition.

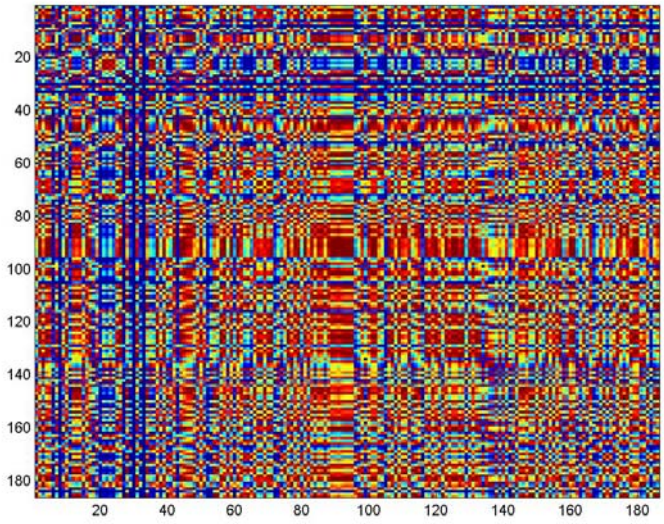
IV. "Recognizing" Faces

The results of this part of my research were somewhat unsuccessful. However, the algorithm is still interesting enough to detail. To begin, I tried to group the original cropped faces into groups of the same person. To do this, I worked with the eigenfaces obtained when training the algorithm.

Basically, there are many ways to go about grouping the images. I believed the best (and fastest) way would to be to obtain the face space coordinates of each face, and

plug this data into a k-means algorithm. For those unfamiliar with k-means, it is a clustering algorithm which goes through data and tries to find clusters of data that are close to one another. Therefore, by plugging the face space coordinates into k-means, the returned clusters should technically represent faces that are similar to one another, or in our case, the same person. Yet, this was not the case, as it seemed as if the results depended more on the shading than on who the actual individual is.

Professor Martin suggested another way of going about the clustering of faces called "spectral clustering". I composed a similarity matrix, a 2-d matrix whose rows and columns corresponded to how similar two faces were. A distance matrix was created first, whose values were computed by determining the distance between the two face space coordinates. This matrix was then run through a Gaussian, which resulted in our similarity matrix, as shown below.



If the results of this were as hoped, there would be blocks down the diagonal of the matrix which represent the same person (for instance, in the figure above, the first fifty faces are one person, the second fifty are another, etc).

This similarity matrix is then passed into k-means, which returns the indexes and locations for picture in the cluster. The following is an example of three different clusters returned as a result of this algorithm.

The first image in these clusters is a representation of the cluster center for each particular group, and gives us an idea of what in common these faces have. For instance, in cluster five, most of the people are looking to their right somewhat, and the smiles are all somewhat similar. Instead of grouping the same people into the same cluster, it's grouping people together who are posed the same way and under the same lighting conditions.

This brings up one of the main reasons why any recognition is difficult. Lighting conditions, pose variation, and simple disguises can throw off a system very easily. Under lab conditions, face recognition is easily possible by using eigenfaces (e.g. mugshots). If particular faces are known to be certain people, a new entry that looks similar to the photographs used to train the image can just be projected into face space, and whichever group it's closest to, that's who the photograph is of. This was the basis

of a homework assignment completed for Professor Martin's Computer Vision class, and it worked quite well since the photos were all shot under lab conditions. But airports, and as I've found out in my photo album, everyday conditions, don't make good photographs to be used in face recognition. Unfortunately, the variables which I believed would be mainly constant in a photo album turned out to be more variable than hoped.

V. Discussion (Successes and Failures)

When viewing the research as a whole, the project was a moderate success. The method of finding a face based on color seems to be a pretty good indicator of where a face could occur. Yet it's the other method of finding faces, the eigenface reconstruction method, in which most of the problems occur. The combination of the two faceness algorithms produces pretty good results (it usually finds most of the faces) but I feel this is more of a result of the color algorithm than the eigenface algorithm. While the multiplication of these two matrices does get rid of the majority of the false positives, when looking at the facemap from eigenface reconstruction one notices that its' results are less than ideal.

One of the biggest problems I faced with the eigenface reconstruction is getting the images in the same range of values. Keeping track of what was normalized became difficult because there were so many different images that were being used (the original images, the original faces, creation of the face matrix and the eigenfaces, the bounding box within the faceness algorithm, etc.). As a result, in the eigenface reconstruction, many times the comparison between the recreated image and the original image was incorrect.

Another problem with the eigenface reconstruction was determining the right size for the eigenfaces and the bounding box that travels through the picture. The size of this box determines the size faces it will find on a particular run through. Of course, an image pyramid is created and as a result it will find bigger faces, but they won't necessarily cancel out possible faces it found in one size if a face didn't actually occur there. Therefore, it can find faces in a lot of different places.

Ultimately, I wasn't happy with the eigenface reconstruction algorithm. The idea sounds like it should work: recreating something that isn't a face by adding together different multiples of pictures of faces should result in a large difference, and therefore, not be a face. A face recreated from different faces will, of course, look like a face and result in a small distance. Yet, the algorithm doesn't seem to work very well because it finds many false positives for a face. This is most likely due to the fact that the algorithm can easily find patterns present in a face (for instance, three points that could be mistaken as two eyes and a nose). Also, the algorithm doesn't account for faces that may be at an angle, so it's not good at finding these faces either.

Clustering of the faces didn't work as well as I had hoped, as discussed in the previous section. I didn't get into this topic as much as I had originally planned because I ran into so many problems finding the faces in an image. Yet, the clustering seems to group according to shading in the image instead of based on facial features like I had hoped.

As for successes, surprisingly by obtaining a facemap using just the color faceness algorithm it returns pretty good results for where faces are. It may bring up a few false positives (hands, etc.) but it usually finds all the faces. However, it is surprisingly pretty smart at picking out skin tones on the face from other skin tones.

The biggest problem with the finding faces is that if the threshold is incorrect, it will either recognize separate faces as one face or it may break down one face into separate segments, and therefore more than one face (for example, it may find the two cheeks as two separate faces because they're two peaks, but our threshold is too high so it thinks it's two separate faces). Another problem is when two faces are very close together, it has a difficult time seeing the data as two separate faces.

VI. Conclusion

In conclusion, the creation of an annotated digital photo album using automatic face recognition is a very difficult task. Using my system, there are too many variables that affect a person's appearance to make it possible for any normal user's desktop to be able to automatically find faces and then determine who they are. As it is, my algorithm to find faces takes two to three minutes per picture (a 640x480 image, while most cameras take pictures at much larger resolutions nowadays) and isn't completely accurate.

While I didn't fully succeed in my task, I did get some very interesting results. My algorithm does work on some photographs, but not on all. When tweaked with the right threshold, the algorithm can usually find the faces. Yet this still requires some manual data entry, when I had hoped to have everything automated.

With continued work on my algorithm, I believe that my method of finding faces can be optimized. The section that needs the biggest improvement is the eigenface reconstruction. My color faceness algorithm accurately finds skin tones. What we need in addition to this is an algorithm that actually finds faces. Even if it finds many false

positives, as long as it can be optimized to not find faces in areas where a skin tone occurs, the combination of these two will wipe out the false positives.

As far as recognition of the faces, a lot of work has to be done to optimize this. The algorithm I used may not even be the right tool for the job; other options should be researched to see if they can get the job done.

The algorithm should also be optimized to run faster. As it is written now, it takes two to three minutes per image to merely find the faces. This will not be fast enough for a large photo album. Ideally, a face finding algorithm should have a runtime of less than 10 seconds per image. Of course, computers are getting faster each year, but we don't want this to be an excuse to have slow code.

Overall, I'd say the project was a success, at least when it comes to finding faces. Although not 100% accurate for all images, the results returned for most images are acceptable and just require another method of canceling out false positives.

```
function [] = facialPoints()
% [] = facialPoints()
% Reads in files 1.jpg -> 100.jpg for purposes of locating faces
% Click the left eye, then the right eye, then the nose.  Press enter
when
% finished choosing all the points in an image

for I=1:100
    fid = fopen('facialpoints.txt','a');
    fprintf(fid,'%d\n',I);
    fclose(fid);
    im = rgb2gray(imread([int2str(I), '.jpg']));
    figure(1);clf;imagesc(im);truesize;axis image;colormap gray;
    [a,b] = ginput;
    fid = fopen('facialpoints.txt','a');

fprintf(fid,'%5d',a);fprintf(fid,'\n');fprintf(fid,'%5d',b);fprintf(fid
,'\n');
    fclose(fid);
end
```

```
function [] = getFaces()
% function [] = getFaces()
% Uses facialpoints.txt, created by facialPoints.m, to create cropped
% images for each face in the picture

a = dlmread('facialpoints.txt');
index=0;

for I=1:100
    [q,numfaces] = size(find(a((3*I)-1,:)~=0));
    numfaces=numfaces/3;
    filename = [ int2str(I) , '.jpg' ];
    im = rgb2gray(double(imread(filename))/255);
    im = padarray(im,[200 200], 0, 'both');
    for J=1:numfaces
        index = index+1;
        fid = fopen('index.html','a');
        xs=a(3*I-1,(3*J)-2:(3*J));
        ys=a(3*I,(3*J)-2:(3*J));
        xs = xs + 200;
        ys = ys + 200;
        points = [ xs ; ys ];   %The 3 points are now in points, for
the face we have
        dist = sqrt((points(1,1)-points(1,2))^2 + (points(2,1) -
points(2,2))^2);
        midpoint = [ mean([points(1,1) points(1,2)]) mean([points(2,1)
points(2,2)]) ];

        %Now we'll find the angle
            angle = atan2(points(2,2)-points(2,1),points(1,2)-
points(1,1));
            angle = angle * (180/pi);
            angles(index)=angle;
            width = round(1 * dist);
            width = width + (1-mod(width,2));
            height = round(1.5 * dist);
            height = height + (1-mod(height,2));
            RECT = [ midpoint(1)-width midpoint(2)-height width*2
height*2 ];
            cropped = imcrop(im,RECT);
            [y,x] = size(cropped); midpt = [ x/2 y/2 ];

        %Now we'll rotate
        cropped = imrotate(cropped,angle,'bicubic', 'crop');
        cropped = imcrop(cropped, [ midpt(1)-width/2 midpt(2)-height/3
width height ]);

        cropped = imresize(cropped,[120 80], 'bicubic');
        figure(1);clf;imagesc(cropped);colormap gray;axis
image;truesize;
        croppedname = [ 'crop', int2str(I) , '_' , int2str(J), '.jpg'
];
        %print('-djpeg',croppedname);
        %nor = cropped - min( cropped(:));
        %normalized = nor / max( nor(:));
        imwrite(cropped,croppedname,'jpeg');
```

```
        fprintf(fid, '<img
src="');fprintf(fid,'%s_%s',int2str(I),int2str(J));fprintf(fid,'">\n');
        fclose(fid);
    end
end
```

```
function [A] = faceMatrix(width,height)
%  function [] = faceMatrix()
%  OUTPUT:
%     A :  A matrix of all the crop*.jpg faces in the directory
%          M by N dimensions, M is the number of faces
%     S :  A similarity

files = dir('crop*.jpg');
[numFaces,x] = size(files);
A = zeros(numFaces,width*height);
%filt = d2gauss(8,5,8,5,0);
for I=1:numFaces%91
    im = double(imread(files(I).name))/255;
    im = imresize(im, [height width], 'bilinear');
    im = im / sum(abs(im(:)));
    %im = imfilter(im,filt);
    %figure(1);clf;imagesc(im);axis image;truesize;colormap gray;
    A(I,:) = reshape(im,1,width*height);
end
```

```
function [p,x,y,faceHist,noFaceHist] =
getColorHist(bins,amin,amax,bmin,bmax);
%
%

txt = dlmread('facialpoints.txt');
index=0;
faceHist=zeros(bins+1,bins+1);
noFaceHist=zeros(bins+1,bins+1);
mina = 9999;  maxa = -9999;
minb = 9999;  maxb = -9999;

for I=1:100
    [q,numfaces] = size(find(txt((3*I)-1,:)~=0));
    numfaces=numfaces/3;
    filename = [ int2str(I) , '.jpg' ];
    im = imread(filename);
    %im = padarray(im,[200 200], 0, 'both');
    [one,two,three] = size(im);
    mask = zeros(one,two);
    for J=1:numfaces
        index = index+1;
        xs=txt(3*I-1,(3*J)-2:(3*J));
        ys=txt(3*I,(3*J)-2:(3*J));
        %xs = xs + 200;
        %ys = ys + 200;
        points = [ xs ; ys ];   %The 3 points are now in points, for
the face we have
        dist = sqrt((points(1,1)-points(1,2))^2 + (points(2,1) -
points(2,2))^2);
        midpoint = [ mean([points(1,1) points(1,2)]) mean([points(2,1)
points(2,2)]) ];

            width = round(1 * dist);
            width = width + (1-mod(width,2));
            height = round(1.5 * dist);
            height = height + (1-mod(height,2));
            RECT = [ midpoint(1)-(width/2) midpoint(2)-(height/3) width
height ];
            %im = imcrop(im,RECT);
            %figure(2);clf;imagesc(im);axis image;truesize;
            RECT = round(RECT);
            if RECT(2)+height >480
                RECT(4) = height - mod(RECT(2)+height,480);
            end
            if RECT(1)+width >6400
                RECT(3) = height - mod(RECT(1)+height,640);
            end

mask(RECT(2):(RECT(2)+RECT(4)),RECT(1):(RECT(1)+RECT(3)))=1;
    end
    I
    [L,a,b] = RGB2Lab(im);
    %size(mask)
    %figure(1);clf;imagesc(mask);axis image;truesize;
    %Now we have the mask, we're going to plot the a and b values now
    %imwrite(mask,['mask' int2str(I) '.jpg'],'jpeg');
```

```
%     a=a+100;b=b+100;

    if max(a(:))>maxa
        maxa=max(a(:));
    end
    if min(a(:))<mina
        mina=min(a(:));
    end
    if max(b(:))>maxb
        maxb=max(b(:));
    end
    if min(b(:))<minb
        minb=min(b(:));
    end


%     a = round(a);b = round(a);

    [one,two,three]=size(im);
    for I=1:one
        for J=1:two
            aval = 1+floor(bins*(a(I,J)-(amin))/(amax-(amin)));
            bval = 1+floor(bins*(b(I,J)-(bmin))/(bmax-(bmin)));
            if aval>bins
                aval=bins;
            end
            if aval<1
                aval=1;
            end
            if bval>bins
                bval=bins;
            end
            if bval<1
                bval=1;
            end
            if mask(I,J) == 0
                noFaceHist(aval,bval)=noFaceHist(aval,bval)+1;
            end
            if mask(I,J) == 1
                faceHist(aval,bval)=faceHist(aval,bval)+1;
            end
        end
    end
end
x = linspace(amin,amax,bins);
y = linspace(bmin,bmax,bins);
figure(1);imagesc(x,y,log(1+faceHist));colorbar;title('Face
Histogram');
figure(2);imagesc(x,y,log(1+noFaceHist));colorbar;title('Not-Face
Histogram');
p = faceHist ./ (1 + faceHist + noFaceHist);
figure(3);imagesc(x,y,p);colorbar;title('Probability of Face');
[ mina maxa minb maxb ]
```

```
function [eigFaceHist,eigNoFaceHist,p] =
getEigHist(bins,eigFaces,meanFace,width,height);
%function [eigFaceHist,eigNoFaceHist] =
getEigHist(bins,eigFaces,meanFace,width,height);
%

eigFaceHist=zeros(bins+1,bins+1);
eigNoFaceHist=zeros(bins+1,bins+1);
mindev = 0;  maxdev = .4;
mindist = .04;  maxdist = .7;
%bigmatrix = zeros(101,390*580);
% mina=9999;maxa=-9999;minb=9999;maxb=-9999;

for I=50:100
    counter=1;
    filename = [ int2str(I) , '.jpg' ];
    im = rgb2gray(double(imread(filename))/255);
    %im = padarray(im,[200 200], 0, 'both');
    h = fspecial('gaussian',31,4);
    im = imfilter(im,h);
    [m,n] = size(im);
    filename2 = ['mask' int2str(I) '.jpg']
    mask = double(imread(filename2))/255;

    topleft = [(height/2)+1 (width/2)+1];
    bottomright = [m-(height/2) n-(width/2)];

    for I= topleft(1):bottomright(1)
       for J= topleft(2):bottomright(2)
           RECT = [J-(width/2) I-(height/2) width-1 height-1];
           thispic = imcrop(im,RECT);

           %figure(1);clf;imagesc(thispic);axis image;truesize;
           imbefore = thispic;
                  thispic  = thispic / sum(abs(thispic(:)));
                  thispic = reshape(thispic,1,width*height);

                  %subtract the mean face
                  im2 = double(thispic) - meanFace;
                  myWeights = im2 * eigFaces';
                  recreatedFace = meanFace + myWeights*eigFaces;

            dev = std(imbefore(:));
            dist = sum(abs(thispic - recreatedFace));
            %bigmatrix(I,counter) = dev;bigmatrix(I+1,counter)=dist;
            counter=counter+1;

            devval = 1+floor(bins*(dev-(mindev))/(maxdev-(mindev)));
            distval = 1+floor(bins*(dist-(mindist))/(maxdist-
(mindist)));
            if devval>bins
                devval=bins;
            end
            if devval<1
                devval=1;
            end
            if distval>bins
```

```
                distval=bins;
            end
            if distval<1
                distval=1;
            end
            if mask(I,J) == 0

eigNoFaceHist(devval,distval)=eigNoFaceHist(devval,distval)+1;
            end
            if mask(I,J) == 1

eigFaceHist(devval,distval)=eigFaceHist(devval,distval)+1;
            end
%             if dev>maxa
%                 maxa=dev;
%             end
%             if dev<mina
%                 mina=dev;
%             end
%             if dist>maxb
%                 maxb=dist;
%             end
%             if dist<minb
%                 minb=dist;
%             end
        end
    end
end
x = linspace(mindev,maxdev,bins);
y = linspace(mindist,maxdist,bins);
figure(1);imagesc(x,y,log(1+eigFaceHist));colorbar;title('Face
Histogram std vs dist');
figure(2);imagesc(x,y,log(1+eigNoFaceHist));colorbar;title('Not-Face
Histogram std vs dist');
p = eigFaceHist ./ (1 + eigFaceHist + eigNoFaceHist);
figure(3);imagesc(x,y,p);colorbar;title('Probability of Face using
eigenFaces');
% [maxdev mindev maxdist mindist]
% [mina maxa minb maxb]
```

```
function [colorfacemap] = colorFaceness(im,p2);
%INPUT
%   im - color image received directly from just imread
%

amin=-60;bmin=-60;
amax=80;bmax=80;
bins=45; sigma = 4;

h = fspecial('gaussian',31,3.5);
im = imfilter(im,h);


[m,n,dontcare] = size(im);
colorfacemap = zeros(m,n);
[L,a,b] = RGB2Lab(im);
for I=1:m
    for J=1:n
        aval = 1+floor(bins*(a(I,J)-(amin))/(amax-(amin)));
        bval = 1+floor(bins*(b(I,J)-(bmin))/(bmax-(bmin)));
        %aval = a(I,J); bval = b(I,J);
        %prob = sqrt((aval-11.5556)^2 + (bval - 8.4444)^2);
        prob = p2(aval,bval);
        colorfacemap(I,J) = prob;%exp(-prob.^2/(2*sigma^2));
    end
end
figure(1);imagesc(colorfacemap);axis image;truesize;title('Faceness of
image based on color');
```

```
function mymatrix = eigenFacenessProb(im,
eigFaces,meanFace,width,height,p);
%function mymatrix = eigenFacenessProb(im,
eigFaces,meanFace,width,height,p);
%INPUT
%    im : a grayscale image
%    eigFaces: the eigenFaces from PCA
%    meanFace: the meanFace from PCA
%    width,height: the width and height of the box to use
%OUTPUT
%    mymatrix : A matrix of the "faceness" of each pixel

[m,n] = size(im);
counter=0;counter2=1;

h = fspecial('gaussian',31,4);
im = imfilter(im,h); mymin = 9999;  mymax = -9999;

   counter=counter+1;
   topleft = [(height/2)+1 (width/2)+1];
   bottomright = [m-(height/2) n-(width/2)];
   matrix = zeros(m-height,n-width);

   for I= topleft(1):bottomright(1)
       for J= topleft(2):bottomright(2)
           RECT = [J-(width/2) I-(height/2) width-1 height-1];
           thispic = imcrop(im,RECT);
           [matrix(I-(height/2),J-(width/2)),recreatedFace] =
faceVal(thispic,eigFaces,meanFace,width,height,counter,I,J,p);
       end
       I
   end

   if counter == 1
       mymatrix = matrix;
   end
   writepic=1-normalize(matrix);

   %********Uncomment the next line if you're making a movie
   %movie2avi(M,['facemovie.avi'],'fps',10);

   imwrite(normalize(writepic),['faceness' int2str(counter)
'.jpg'],'jpeg');
   fid = fopen('faceness.htm','a');
   fprintf(fid, '<img src="');fprintf(fid,'%s',['faceness'
int2str(counter) '.jpg']);
   fprintf(fid,'"><br>');
   fclose(fid);
   im=imresize(im,size(im)/sqrt(2),'bilinear');
   [m,n]=size(im);
   %end


function [val,recreatedFace] =
faceVal(im,eigFaces,meanFace,width,height,counter,I,J,p);
%
%
```

```
%
mindev = 0;  maxdev = .4;
mindist = .04;  maxdist = .7; bins = 45;
imbefore = im;
im  = im / sum(abs(im(:)));
im = reshape(im,1,width*height);

%subtract the mean face
im2 = double(im) - meanFace;
%Take the dot product to determine the weights
myWeights = im2 * eigFaces';
%Now recreate the face by multiplying the weights and eigenfaces and
adding
%to the meanFace
recreatedFace = meanFace + myWeights*eigFaces;
dev = std(imbefore(:));
dist = sum(abs(im - recreatedFace));
devval = 1+floor(bins*(dev-(mindev))/(maxdev-(mindev)));
distval = 1+floor(bins*(dist-(mindist))/(maxdist-(mindist)));
val = p(devval,distval);
```

```
function [facemap,binFaces,numFaces] = getFaceness(im,colProb,eigProb);
%function facemap = getFaceness(im,colProb,eigProb);
%
%INPUT
%     im - An image read directly from imread
%     colProb - The probability the pixel is a face based on color
(Saved as p2)
%     eigProb - The probability the box is a face based on STD/dist to
face
%               space (saved as p3)

colorFacemap = colorFaceness(im,colProb);

im = rgb2gray(double(im)/255);
A = faceMatrix(60,90);
[eigFaces, faceSpace, meanFace] = PCA(15,A);
eigFacemap = eigenFacenessProb(im,eigFaces,meanFace,60,90,eigProb);
h = fspecial('gaussian',31,2);
eigFacemap = imfilter(eigFacemap,h);

facemap = combineFaceness(eigFacemap,colorFacemap);

[binFaces,numFaces] = findFaces(facemap);


figure(1);imagesc(facemap);axis image;truesize;title('Total Combined
Faceness');
figure(2);imagesc(colorFacemap);axis image;truesize;title('Faceness
based on Color');
figure(3);imagesc(eigFacemap);axis image;truesize;title('Faceness based
on Eigenface Reconstruction');
figure(4);imagesc(binFaces);axis image;truesize;title('Location of the
faces');
```

```
function [faceness] = combineFaceness(eigFaceness,colFaceness)
%
%
%

[m,n] = size(eigFaceness);
[m2,n2] = size(colFaceness);

eigFaceness = padarray(eigFaceness,[(m2-m)/2 (n2-n)/2],0,'both');

faceness = (eigFaceness) .* (.5*colFaceness);
figure(4);imagesc(faceness);axis image;truesize;title('Combination of
color and eigenface faceness');
```

```
function [eigenfaces,facespace,meanFace] = PCA(numeigs,A);
%function [eigenfaces,facespace,meanFace] = PCA(numeigs,A);
%INPUTS
%   numeigs : The number of eigenfaces to return
%   A : An m x n matrix where the rows correspond to one face image and
n
%   is the number of faces
%OUTPUTS
%   eigenfaces: the eigenFaces returned from svd
%   facespace: the weights for each of the faces
%   meanFace: the average face


meanFace = mean(A);
[m,n] = size(A);
%figure(1);clf;imagesc(reshape(meanFace',120,80));axis
image;truesize;colormap gray;
% norm = meanFace - min ( meanFace(:) ); normmean = norm / max (
norm(:) );
% imwrite(normmean,'meanFace.jpg','jpeg');
%
%
% fid = fopen('eigfaceindex.html','a'); fprintf(fid,'<img
src="meanFace.jpg">'); fclose(fid);

for I=1:m
    A(I,:) = A(I,:) - meanFace;
end

%Find the eigenvectors of A
[u,s,v] = svd(A',0);
[a,b] = size(u);

%************The following commented out code creates jpg files of the
%eigenfaces*****************************
% for I=1:b
%     eigface = reshape(u(:,I),120,80);
%     norm = eigface - min ( eigface(:) ); normface = norm / max (
norm(:) );
%     figure(2);clf;imagesc(eigface);axis image;truesize;colormap gray;
%     eigfacename = [ 'eigface' , int2str(I) , '.jpg' ];
%     imwrite(normface,eigfacename,'jpeg');
%     fid = fopen('eigfaceindex.html','a');
%     fprintf(fid, '<img
src="eigface');fprintf(fid,'%s',int2str(I));fprintf(fid,'.jpg">\n');
%     fclose(fid);
% end

%*********The following code creates a cumulative sum of the
eigenfaces,
%showing that using about 20 brings back most the data***************
% [x,y] = size(s);
% eigvals = s*s; sum = trace(eigvals);
% for I=1:x
%     eigvals(I,I) = eigvals(I,I) / sum;
```

```
% end
% diagonal = diag(eigvals);
% cumul = cumsum(diagonal);
% figure(1);hold on;
% for I=1:numeigs
%     plot(I,cumul(I),'rx');
% end

eigenfaces = zeros(numeigs,n);
for I=1:numeigs
    eigenfaces(I,:) = u(:,I)';
end

facespace = zeros(m,numeigs);
for I=1:m
    for J=1:numeigs
        facespace(I,J) = dot(A(I,:),eigenfaces(J,:));
    end
end
```

```
function [D] = distanceMatrix(faceSpace,eigFaces,meanFace,A)
%  function [D] = distanceMatrix(faceSpace,eigFaces,meanFace)
%   Returns D, a matrix of the distance between pictures
%

numFaces = size(A,1);
D = zeros(numFaces,numFaces);

% for J=1:numFaces
%     im = A(J,:);
%     for I=1:n
%         weights(J,I) = dot(im(1,:),eigFaces(I,:));
%     end
% end

for I=1:numFaces
    for J=1:numFaces
        D(I,J) = findSimilarity(faceSpace(I,:),faceSpace(J,:));
    end
end

function [dist] = findSimilarity(point1,point2)
%  function [dist] = findSimilarity(point1,point2)
%INPUTS
%    point1,point2: the two "points" (can be any dimension)
%OUTPUT
%    dist: distance between the two points



dist = sqrt(sum(point1 - point2).^2);
```

```
function [S] = similarityMatrix(D)
%  function [S] = similarityMatrix(faceSpace,eigFaces,meanFace)
%   INPUT :
%       D - Distance matrix received from distanceMatrix.m
%   OUTPUT:
%       S - D after it's pushed through a gaussian to distribute the
data

sigma = std(D(:));
S=exp(-D.^2/(2*(sigma^2)));
```

```
function [index,loc] = pairCluster(S,k,clusters)
%
%
%
%

[u,s,v] = svd(S);
[m,n] = size(u);
eigvect = zeros(k,m);
for I=1:k
    eigvect(I,:) = u(:,I)';
end


[index,loc] = kmeans(eigvect',clusters,'Replicates',20,
'EmptyAction','singleton','Display','iter');
```

```
function [] =
makeClusterPage(kmeaned,loc,eigFaces,meanFace,pagename,desc)
% function [] =
makeClusterPage(kmeaned,loc,faceSpace,meanFace,pagename,desc)
%   INPUT
%      kmeaned : the index received from kmeans
%      loc : the loc received from kmeans
%      pagename: the name of the output index page (without .htm)
%      desc : a string description of the page

pics = dir('*_*_*.jpg');
[m,n] = size(kmeaned);
fid = fopen([pagename '.htm'],'a');
fprintf(fid,'<font size="+3"<b>');fprintf(fid,
desc);fprintf(fid,'</font></b><p>');
fclose(fid);
for I=1:max(kmeaned)
      %recreateFaceIm(loc(I,:),pagename,I);
      recreateFaceK(loc(I,:),pagename,eigFaces,meanFace,I);
      for J=1:m
          if kmeaned(J)==I
              fid = fopen([pagename '.htm'],'a');
              fprintf(fid, '<img
src="');fprintf(fid,'%s',pics(J).name);fprintf(fid,'">\n');
              fclose(fid);
          end
      end
     fid = fopen([pagename '.htm'],'a');
      fprintf(fid,'<p>');
      fclose(fid);
  end
```

```
function [] = recreateFaceK(loc,pagename,eigFaces,meanFace,J)
%INPUTS:
%     loc : the loc returned by kmeans
%     pagename: the name of the html file
%     eigFaces: the eigenFaces from PCA
%     meanFace: the meanFace from PCA
%     J : the number that's put after the filename

recreatedA = meanFace;
[dontcare,numeigs] = size(loc);
for I=1:numeigs
   recreatedA = recreatedA + (loc(1,I) * eigFaces(I,:));
end
myim = reshape(recreatedA,120,80); myim = normalize(myim);
imwrite(myim,[pagename int2str(J) '.jpg'],'jpeg');

fid = fopen([pagename '.htm'],'a');
fprintf(fid, 'Cluster
');fprintf(fid,'%s',int2str(J));fprintf(fid,':\n');
fprintf(fid, '<img src="');fprintf(fid,'%s',[pagename int2str(J)
'.jpg']);fprintf(fid,'">\n');
fclose(fid);
```

```
function [] = recreateFaceIm(myim,pagename,I);
%  [] = function recreateFaceIm(loc,pagename);
%   Used to recreate a face from image pixels
%INPUT
%   myim : a 1x9600 matrix image
%   pagename: name of the index page made
%   I: number put after the filename

    myim = reshape(myim,120,80); myim = normalize(myim);
    imwrite(myim,[pagename int2str(I) '.jpg'],'jpeg');



fid = fopen([pagename '.htm'],'a');
fprintf(fid, 'Cluster
');fprintf(fid,'%s',int2str(I));fprintf(fid,':\n');
fprintf(fid, '<img src="');fprintf(fid,'%s',[pagename int2str(I)
'.jpg']);fprintf(fid,'">\n');
fclose(fid);
```