Face Detection and Recognition in Color Images using Image Segmentation and LPQ

Niklas Andersson, nikan278,
Joakim Deborg, joade361
Gabriel Baravdish, gabba873
Lars Bergman, larbe444
December 13, 2015

Abstract

In this report an algorithm is proposed to find and recognize faces in color images. The established and implemented algorithm is divided into three main phases; *color correction*, *face detection* and *face recognition*. For color correction, a method combining black and white balancing is used. The face detection part build upon image segmentation techniques, while the recognition part is based on a method called *Local Phase Quantization*, which involves various *Fourier transforms*. Throughout the report we describe the methods and the accompanying theory we use and base our design decisions on. The report also include a discussion and the presentation of the experimental results achieved.

Keywords: Black balancing, White balancing, Eye map, Mouth map, YCbCr color space, HSV color space, Image segmentation, Morphological operations, Circular Hough transform, Eigenfaces, Local phase quantization

Contents

1 Introduction

1.1 Aim

The aim of this project work is to implement an algorithm in *Matlab* that can detect and recognize human faces in different color images. The implementation should, from an input image of an unknown face, detect where in the image the face is located and make a decision if it belongs to a known person in the given database. Furthermore should the algorithm be tolerant for small rotations, translations and scaling inside the image plane as well as for varying tone values, lightning conditions, facial poses, background environments and degrees of blur. Other prerequisites for the project work were that the input image always include a front-faced portrait and that the images could be of varying size.

1.2 Pipeline

The pipeline of a standard frontal face authentication algorithm involves solving multiple sub problems, such as: *preprocessing*, *face detection* and *face recognition*. Each of these sub problems can be solved in various ways and below we list our solutions.

- **In the preprocessing phase** we use simple methods of color correction, namely black and white balancing.
- **The face detection phase** is built around finding regions of skin in the *YCbCr* color space and the combination of those regions with masks gathered through various segmentation steps.
- **For the face recognition part** we base our algorithm upon on the *Local Phase Quantization* (LPQ) algorithm.

2 Theory

This section presents the underlying theory behind some of the methods utilized.

2.1 Face detection

Here in this section are the computation of the eye map and mouth map as well as the theory behind the alignment of the faces explained. All of which are methods that are used when deriving the final output of the face detection phase.

2.1.1 Eye map

Eyes can be located by recognizing signature eye regions in images. One way to do this is through the utilization of the *YCbCr* color space [1]. This process involves the development of a so called *eye map*, which essentially is a gray scale image that is brighter where the locations of the eyes could be present. The eye map itself is derived from two independent components, the *chroma* component and the *luma* component. After these two components have been compiled, they are merged via an AND operation to establish the eye map.

The interesting parts about the YCbCr color space, which allows us to proceed this way, are that the C_b and C_r channels are observed to have high variation around the eyes. Additionally, it is also observed that eyes most often contain both dark and bright pixels in the Y channel. Because of these observations, as described in [1], we can construct the chroma component of the eye map through Equation 1 and the luma component by Equation 2, where $\tilde{C_r}$ is the inverted C_r channel and where $\oplus$ represents the dilation and $\ominus$ the erosion of the Y channel by a kernel K_n. The kernel K_n can be of various shapes, e.g. an ellipse. For this algorithm, a circular disk is used, where the kernel width is proportional to n.

$$eyeMapChroma = \frac{1}{3}\{C_b^2 + \tilde{C_r}^2 + \frac{C_b}{C_r}\} \tag{1}$$

$$eyeMapLuma = \sum_{n=1}^{10} \frac{Y \oplus K_n}{n + Y \ominus K_n} \tag{2}$$

2.1.2 Mouth map

As [1] also mentions, the mouth area contain much stronger C_r than C_b values compared to the other parts of a face. This is due to the fact that the mouth is more often red than blue. By studying combinations of the C_r and C_b channels, one can therefore realize that the mouth region receives a low response by $\frac{C_r}{C_b}$ and a high response for C_r^2. As of such, it is possible to extract an accurate *mouth map* through Equation 3 and 4.

$$mouthMap = C_r^2 \cdot (C_r^2 - \eta \cdot \frac{C_r}{C_b})^2 \tag{3}$$

$$\eta = 0.95 \cdot \frac{\frac{1}{n} \cdot \sum C_r^2}{\frac{1}{n} \cdot \sum \frac{C_r}{C_b}}. \tag{4}$$

The sums in Equation 4 are defined over all pixels in the image for each component.

All color components in Equation 1, 2, 3 and 4 are normalized to the range $[0, 255]$.

2.1.3 Circular Hough transform

A compensation for the image plane rotation of a face can be achieved by taking advantage of the symmetry of faces, namely that both eyes most often form a vector that is parallel to the horizontal axis. After the extraction of such a vector, one can easily find the rotation of the face.

The algorithm we use to locate the eyes is called the *Circular Hough Transform* (CHT) which is a feature extraction technique for finding circles. The algorithm is popular and often used because of its robustness in noisy areas and tolerance for occlusion and varying illumination [2]. However, it does not exists a specific definition of the algorithm steps based on CHT. A brief description of a customized version is included below.

The first step is to locate pixels at regions with high gradient variations. These pixels are called candidate pixels and they have the ability to cast *votes*, in a circular pattern around themselves. An accumulator array is then defined to store the votes. More votes on a pixel increases the likelihood of it being a center point for a circle. The vote-pattern is illustrated by Figure 1.

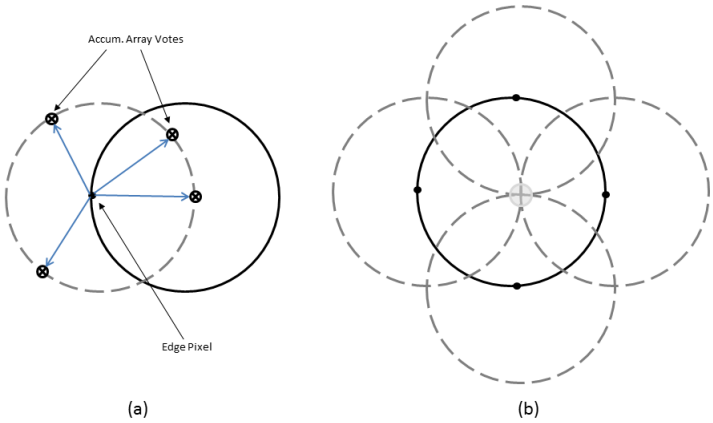


Figure 1: CHT voting pattern to estimate the center of the circle.

The second step is to estimate a center point. The position votes from the candidate pixels belonging to an image circle, tend to accumulate at the center of the circle. Therefore, the center of the circle is estimated by detecting the peaks in the accumulator array.

2.2 Face recognition

This section explains the theory behind different face recognition solutions. The methods explored where Eigenfaces and Local Phase Quantization.

2.2.1 Eigenfaces

Eigenfaces can be used as an approach for human face recognition and is the name given to a set of eigenvectors [3]. According to the authors of [4], these are the initialization operations to acquire the eigenfaces:

- Prepare a training set of face images.
- Calculate the eigenfaces from the training set. Only keep the M images that correspond to the highest eigenvalues. It is these images that defines the *face space*.
- Calculate the corresponding distribution in M-dimensional weight space for each known individual and projecting their face images onto the face space.

To calculate the eigenfaces, operations such as matrix multiplication and sums needs to be computed. Firstly we want to subtract the average image from each original image. Let the training set images be $\mathbf{\Gamma}_1, \mathbf{\Gamma}_2, \mathbf{\Gamma}_3 ... \mathbf{\Gamma}_M$. Also let the average face be $\mathbf{\Psi}$, then each face differs from the average by the vector $\mathbf{\Phi}_i = \mathbf{\Gamma}_i - \mathbf{\Psi}$. Secondly we need to calculate the eigenvectors and eigenvalues from the covariance matrix C. This can be done by Equation 5 below.

$$C = \frac{1}{M} \sum_{n=1}^{M} \mathbf{\Phi}_n \mathbf{\Phi}_n^T = AA^T \tag{5}$$

Where $A = [\mathbf{\Phi}_1 ... \mathbf{\Phi}_M]$. It is possible to determine the eigenvectors of C right away, but it is more efficient to compute the eigenvectors of the smaller matrix AA^T.

The eigenfaces $\mathbf{u}_l$ are given by Equation 6 below.

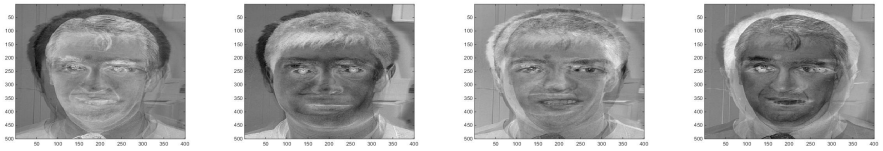
$$\mathbf{u}_l = \sum_{k=1}^{M} \mathbf{v}_{lk} \mathbf{\Phi}_k, l = 1...M \tag{6}$$

For the face recognition part, we need to project an image $\mathbf{\Gamma}$ to the face space $\mathbf{\Omega} = \widehat{U}^T(\mathbf{\Gamma} - \mathbf{\Psi})$ and determine which face class that image belongs to. Here $\widehat{U}$ is the set of eigenvectors and $\mathbf{\Omega}$ is the weight vector that represents the new face in face space. By minimizing the Euclidean distance, which is shown in Equation 7, we can determine which face class a certain image belongs to.

$$\epsilon_k = \|\mathbf{\Omega} - \mathbf{\Omega}_k\| \tag{7}$$

In Equation 7, $\mathbf{\Omega}_k$ is a weight vector that represents the kth face class. If the distance ϵ_k is lower than a certain threshold $\mathbf{\Theta}$, the image is considered to belong to the database.

In Figure 2 below some eigenfaces are shown with our implementation.



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

Figure 2: Example of some eigenfaces from our database.

2.2.2 Local phase quantization

Applying the method of local phase quantization (LPQ) to solve the problem of face recognition was introduced in [5]. This method is proposed especially to solve the problem of recognizing faces in blurred images, since the image descriptions provided by LPQ are considered to be blur invariant.

The LPQ method work under the assumption that the blur in an image can be represented as a function, called the point spread function (PSF). The final blurred image can then be obtained by applying a convolution between the clean image and the PSF. As seen in Equation 8, where i is the clean image, b is the PSF and f is the blurred image.

$$f = i * b \tag{8}$$

Applying the Fourier transform to Equation 8 will result in Equation 9, where I, B and F represent the Fourier transforms of i, b and f.

$$F = I \cdot B \tag{9}$$

Taking the argument of F in order to compute the phase will result in Equation 10, where $\angle F$, $\angle I$ and $\angle B$ are the phases of F, I and B respectively.

$$\angle F = \angle I \cdot \angle B \tag{10}$$

From Equation 10, a blur invariant descriptor of the image can be derived using two properties of the Fourier transform. The first property states that the Fourier transform of a symmetric function will be a real value function [6, p. 151]. By combining this with the knowledge from [7], that PSFs are

generally symmetric and that the phase of a real value function is either 0 or π, the conclusion can be reached that $\angle B$ can only assume the value 0 or π. Furthermore, [5] claims that regular PSFs are often close in shape to a Gaussian function or a sinc function, which as shown in [6, p. 144-148], has a positive Fourier transform for small enough frequencies. This realization leads to the conclusion that $\angle B = 0$. Using this in Equation 10 results in Equation 11, that indicates that $\angle F$ is a blur invariant descriptor of the image since it is not dependent on the PSF.

$$\angle F = \angle I \tag{11}$$

In order to compute the phase of the image, a short-time Fourier transform (STFT) is used. The STFT computes the Fourier transform for a given frequency interval that is chosen to be small enough for $B > 0$ to be valid. Four such intervals, seen in Equation 12, are chosen and computed per pixel.

$$u_1 = [a, 0], u_2 = [0, a], u_3 = [a, a], u_4 = [a, -a] \tag{12}$$

For every pixel the real and imaginary parts of each STFT computation is stored separately and transformed into a binary representation using Equation 13.

$$g = \begin{Bmatrix} 1, & if\ g \geq 0 \\ 0, & else \end{Bmatrix} \tag{13}$$

Combining the binary representations of the real and imaginary parts of all four STFT values results in an 8-bit binary value. This value can be transformed into a decimal number between 0 and 255. Finally the image descriptor can be derived by computing a histogram over the decimal numbers representing each pixel.

Once the image descriptor have been computed it can be compared with other descriptors to see if their respective images can be considered a match. The comparison is performed by calculating the average euclidean distance between the values in the descriptors. If the distance is smaller than a certain threshold, the images are considered a match.

3 Method

The constructed algorithm consists of three major phases, color correction, face detection and face recognition. The phases are linked together in a chain, meaning that the color corrected image given as output from the color correction phase will serve as input to the face detection phase and that the detected face is delivered as input to the face recognition phase. Finally the recognition phase will determine if the given face is stored in the database.

In this section we will go into detail describing the methods we use in each phase.

3.1 Color correction

We have implemented a way of color correcting the images given as input. It consists of two parts, where the first part is black balancing and the second part is white balancing. An example of what our algorithm achieve is pictured in Figure 3.

(a) (b)

Figure 3: Original image (a) and color corrected image (b).

3.1.1 Black balancing

This part aims to calibrate the black pixels of the image in order to make the most black pixels true black. We got the idea to do this when we saw that some of the faces in our dataset were not correctly estimated by our YCbCr color space thresholding. In particular there were some blue tinted images, such as Figure 3*a*, that did not register faces properly.

To solve this problem we extract a few percent (1.5) of the most black pixels in each channel of the image. Then, we compute the mean value of these before we subtract the computed values from each pixel of the image. The resulting image will then contain more pure black pixels but also more poor white pixels, which leads us on to the next part, namely white balancing.

3.1.2 White balancing

This part aims to calibrate the white pixels of the image in order to achieve better white pixels. It is essentially the same process as black balancing, however with one notable difference. Instead of utilizing subtraction by the mean values we apply scale factors that raise these mean white values of the brightest pixels to become true white. Here we take 0.1 percent of the most white pixels of each channel into consideration when computing the scale factors.

3.2 Face detection

This section will be explained through an example where the input is the color corrected image of Figure 4*a*.

In short, initially the input image is padded with zeros on all sides by a $4x4$ kernel in order for future morphological operations to work as intended. Several masks are then derived and used for image segmentation in order to narrow the search space for eyes and mouth. To be clear, we will call the mask that marks out where the estimated face is in the image for *face mask* and the result of this phase for *output*. The output is essentially an image derived from a narrowed face mask in form of a rectangle that precisely covers the eyes and mouth, see Figure 11*c* for an example. When the face mask is found we can search for the eyes and mouth. The eyes and mouth will then be used to compute the output.

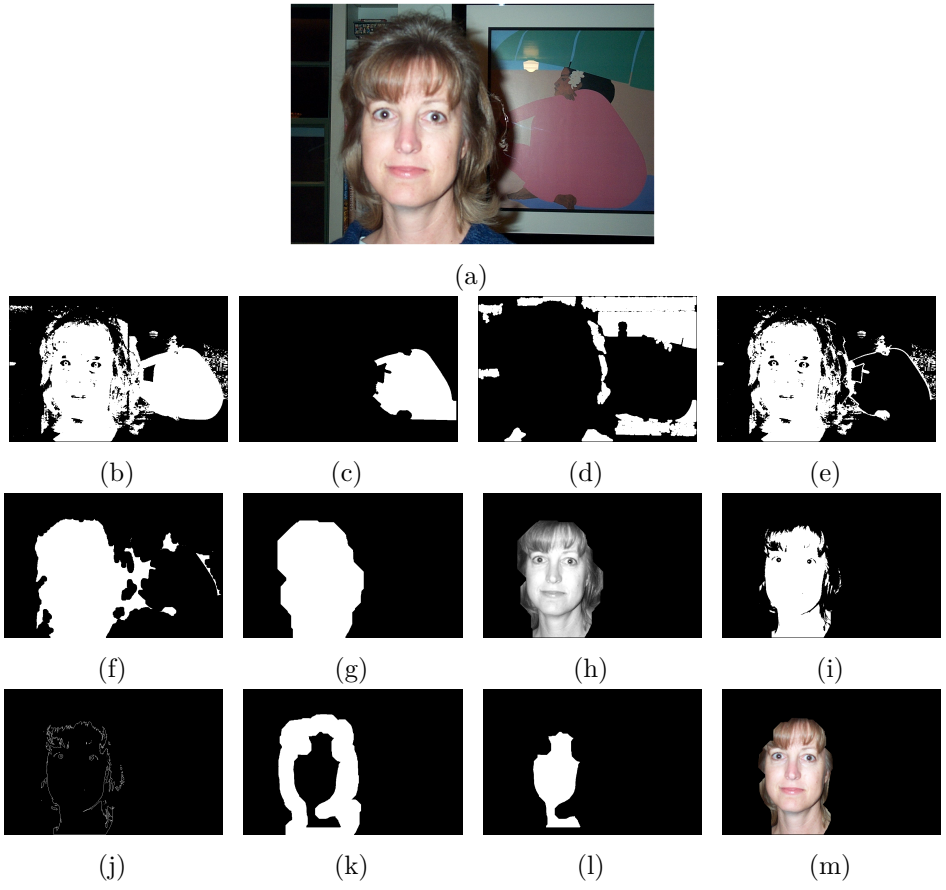


Figure 4: Masks *(b-l)* that are generated during the process of deriving the final face m from input image a.

3.2.1 Estimated skin mask

After padding the color corrected input image several masks are computed. At first, a mask for estimated skin regions is computed through thresholding inside the *YCbCr* color space. See Equation 14 for the specific values we use and Figure 4*b* for the estimated skin mask of this example.

$$\begin{aligned} estimatedSkinMask = and(and(C_b > 95,\ C_b < 145), \\ and(C_r > 132,\ C_r < 165)) \end{aligned} \tag{14}$$

3.2.2 Background mask

Next follows the computation of a mask for the background (Figure 4*c*), or in more particular, a mask containing large homogeneous regions of low frequency. This as we assume that a face contain a lot of high frequency parts while background regions, e.g. walls, often are homogeneous and mostly consist of low frequencies. To avoid that small regions in the face are selected for this mask, we remove all regions with areas containing less than 5 percent of the pixels of the image.

The function takes the gray version of the input image as argument. Furthermore, we first increase the contrast of the image by using *Matlab's imadjust* function. Then we use edge filtering on the image with the *Canny* kernel. This will highlight the edges of the image. We then perform some morphological operations that will implicitly threshold the image and deliver a binary mask containing blobs that represent each region, see Figure 5. As previously mentioned before continuing any computation, we remove small blobs.

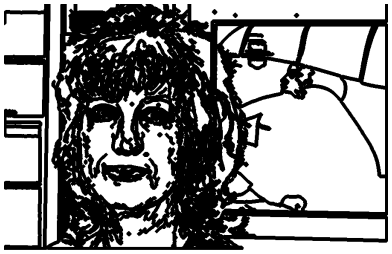


Figure 5: Segmented image for determining the background mask.

Then we derive a gray scale image representing the amount of high frequencies in each region by performing a median filtering of the input image followed by a filtering with the *Laplacian of Gaussian* kernel. The median filter will remove unwanted noise as we are only interested in those frequencies caused by the actual objects in our image. The LoG filter will

first smooth the image a bit more by the Gaussian kernel before the Laplacian kernel is applied in order to give us the more important high frequencies of the image. Given this computed image containing the high frequency information we can for each region blob calculate the percentage of high frequency. We conclude that if the high frequency part is less than 0.8 percent, it is likely that this region of the image is part of the background and that we therefore can add this blob to the background mask.

3.2.3 Non face regions mask

Next follows the computation of a mask containing regions of the image that we deem will not be part of the face, see Figure 4*d*. This through the utilization of the YCbCr as well as the HSV color space, as seen in Equation 15. The mask is before being used also subject of some morphological operations (opening and removal of small blobs), this in order to remove small parts, like eyes, which sometimes are included in this mask.

$$\begin{aligned} nonSkinMask1 &= C_b > C_r + 10 \\ nonSkinMask2 &= V < S - 0.2 \\ nonFaceMask &= or(nonSkinMask1,\ nonSkinMask2) \end{aligned} \tag{15}$$

3.2.4 Face mask

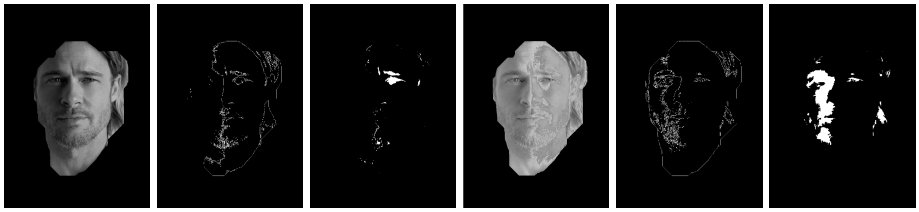
Now we can combine the estimated skin mask (Figure 4*b*) with the background mask (Figure 4*c*) as well as the mask containing non face regions (Figure 4*d*) in order to derive our first estimation for the face mask (Figure 4*e*). Worth noticing is that Figure 4*e* both contains large parts of the woman's face as well as parts of the pictured person's face in the background painting. The combining is essentially done by an AND operation;

AND(estimatedSkinMask, $\neg$ *backgroundMask,* $\neg$ *nonFaceMask).*

To improve this estimation we perform opening operations, this will give us Figure 4*f*. Then to further improve the estimation all the holes in the derived mask is filled before the largest blob is selected to be the new estimation. This new estimation will then be the subject of an iterative opening process that shrinks the mask as much as possible while making sure that there still is a blob left in the mask, see the result of this process in Figure 4*g*.

We now try to further improve the estimation while we in the process also try to derive eye candidate masks. To do this we first compute a gray image over the face, Figure 4*h*, by utilization of our face mask estimation. If the gray face image has one large bright part and one large dark part, which corresponds to the person being lit by a light from the side and that the

mean values of these regions are sufficiently diverse, we brighten the dark part of the face by adding 80 percent of the mean value difference to each pixel of the dark part. This reasoning is illustrated by Figure 6 and notice that both eyes are represented among the eye candidates in Figure 6*f* while only one eye is represented among the eye candidates in 6*c*. Furthermore, to compute the bright and dark part of the image we use the Y component of the YCbCr color space.



(a) (b) (c) (d) (e) (f)

Figure 6: Process of equalizing the dark and bright part of a face being lit from the side. (a) show the original gray face image while (b) show filtered face mask and (c) the resulting eye candidates for the same image. (d) display the improved gray face image while (e) display the filtered face mask and (f) the eye candidates of this improved image.

By thresholding the gray face image by 0.47 we receive the binary image seen in Figure 4*i*. This binary image is only computed to be used for deriving the face contours, see Figure 4*j*. This is done by the unconventional way of using filtering with the Laplacian kernel, however through experimentation we have found out that this method works better for this specific case than other edge detection algorithms. This filtered face mask is then influenced by a number of morphological and binary operations. We first extract the main contour of the face, this is done by filling in all the holes and XOR-ing this mask with an eroded version of this mask. It is done in this particular way as if the face contour instead was selected by the blob with largest area, we could get a blob inside the face instead, e.g. a contour of someones beard. If we on the other hand would select the blob that has the largest convex hull the face border could sometimes be connected to the borders of the eyes (e.g. slight side pose), eliminating one of the eyes from our mask, which is not what we want. By selecting regions that lie inside of the face contour we can extract different masks for eye candidates, combined they make up Figure 7*e*.

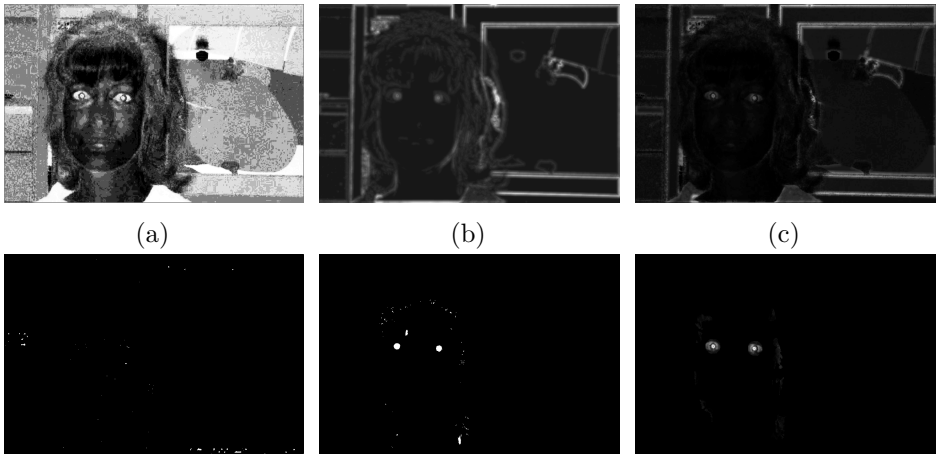
Now that we have computed our eye candidates that help us narrow the search space for the actual eyes, we can try to improve the estimated face mask even more. However observe that this last part was not included

in the final version of our algorithm as we found out that it only made a marginal difference and hence only constraint our algorithm without giving much benefit. Nevertheless it will be explained below.

The idea is established from the idea to minimize excessive hair and the neck from some faces. For example, removing the neck will also remove eventual necklaces that could interfere with the process of finding the eyes. The same applies to hair, as curls often form circular patterns that could be mistaken for an eye. Therefore we begin with dilating the contour of the face quite a lot. This as the hair line of our face contour often consist of a zig-zag pattern and that dilation will therefore smooth out these lines, see Figure 4*k*. Now by filling in the holes and XOR-ing with the previous state we could derive Figure 4*l*. Notice how the neck in this case is almost in an own blob, this is what we wanted, but this did not happen with this particular image. If the neck would be separated in an own blob, we would select the largest blob to be our new estimation of the face mask. To determine the final face mask used to derive the face image in Figure 4*m* we only need to perform some iterative dilation to our estimation. Notice how this method got rid of some of the hair and that it almost got rid of the neck in this case by comparing with Figure 4*h*.

3.2.5 Eye map

In order to deliver a precise output image we first need to locate the positions of the eyes and mouth. To extract the eyes we go by the method described in [1] and explained in the theory section above. By Equation 1 we compute the chromatic component of the eye map which can be seen in Figure 7*a*. More over is the luminance component of the eye map computed according to Equation 2 and displayed in Figure 7*b* while the resulting original eye map is seen in Figure 7*c*.

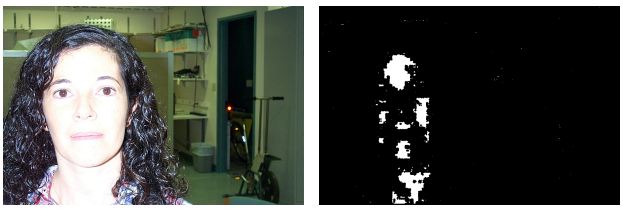


(a) (b) (c)

(d) (e) (f)

Figure 7: Process of computing the eyes locations. (a) is the chromatic component of the eye map, (b) the luminance component, (c) the original eye map, (d) the mask of over saturated regions, (e) the eye candidates while (f) is the final eye map.

To narrow the search space for the *Circular Hough Transform* that will try to find bright circles in the eye map, we want to detract as much as possible from it while leaving the eyes present. Through experimental testing we have found that oversaturated parts of some images have an negative influence on the outcome of the Circular Hough Transform when it is searching for the locations of the eyes. Therefore we attempt to minify their footprint by the construction and utilization of a mask containing those regions. In this particular image there are no prominent oversaturated regions as is easily understood by viewing the oversaturated mask in Figure 7*d*. However Figure 8 illustrates that there indeed is oversaturated parts of some images and that the oversaturation usually avoid the eye regions which let us detract this mask from the original eye map. The mask itself is computed in the same fashion as the process of determining background regions, i.e. by the use of filtering with the Laplacian kernel, thresholding and dilation.



(a) (b)

Figure 8: Original image (a) and the mask of oversaturated regions (b).

Another mask that we construct and detract from the original eye map is one derived from the mean colors of each channel between the pixels inside of the face mask. The negated version of the mask detracted in this example can be seen in Figure 9, however it does not provide much benefit in this case. There are though cases when this procedure will remove curls of hair hanging down on the forehead which is why we came to think about this solution. The solution is based on the assumption that we should be able to derive a better mask for the skin of the face after we have derived the face mask and that the eyes should lie inside of the filled version of that skin mask.

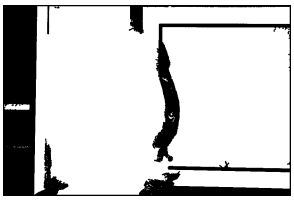


Figure 9: Mask containing the pixels of the original image that lie close to the mean color of the face mask. Here is pixels with a maximum difference of 20 percent from the mean value of each channel allowed. This as we have found that individual comparisons of the values of each channel works better than comparing the actual colors (RGB vectors).

Finally we also detract a tentative mouth mask and everything outside of the face mask while AND-ing that resulting mask with a dilated version of the mask containing the eye candidates seen in Figure 7*e*. This execution gives us the final eye map (depicted in Figure 7*f*) that we will hand over to the Circular Hough Transform that will try to find the true locations of the eyes.

3.2.6 Eyes extraction

We use the Circular Hough Transform implicitly by using Matlab's function *imfindcircles* when we determine the locations of the eyes in the final eye map. We have set a very high threshold to be used by the function making it deliver many possible bright circles that could be our eyes. As the returning output comes out in a sorted list with the best circle estimation at index zero, we assume that this circle is one of the eyes. Then we look at the next index to see if that circle is the other eye. If we find that the location of this circle is not further away than the combined radius of the first eye and this circle's times four, we deem this eye candidate being too close to the first eye in order to be accepted as being the second eye. Therefore we will

repeatedly move on to next circle until we find one that can be verified as being the second eye.

3.2.7 Mouth map

The mouth map seen in Figure 10*a* is computed by Equation 3 after the face has been aligned by compensating for the rotations in the image plane. By thresholding the mouth map with half of the maximum value inside of the rectangular mask seen in Figure 10*b*, we find blobs of mouth candidates. Figure 10*b* is derived from the eyes by measuring the distance between them while using this distance when estimating where the mouth should be located. Then by selecting the largest blob among the mouth candidates we determine the mouth mask seen in Figure 10*c* and by computing its centroid we also determine its location.

(a) (b) (c)

Figure 10: Process of computing the location of the mouth. (a) show the mouth map, (b) a mask for the region where the mouth should be located while (c) show the final mouth mask.

3.2.8 Output

When the final eye map, Figure 7*f*, has been computed we compensate for any rotations less than 90° inside of the image plane in order to align the face. We find the rotation of the face by determining the angle of the vector ranging from the left eye to the right eye. Figure 11*a* show the face image and the locations of the eyes before the compensating rotation has taken place while Figure 11*b* show the compensated image. Finally by measuring the horizontal distance between the eyes and the vertical distance between the eyes and mouth, we can derive the output of the face detection phase seen in Figure 11*c*.

(a) (b) (c)

Figure 11: Aligning the eyes to the horizontal axis. (a) show the face image before correction of the rotation in the image plane is performed, (b) show the rotated version and (c) the final output.

3.3 Face recognition

The problem of face recognition was solved using the LPQ method described in the Theory section.

The short-time Fourier transform was performed by using a two step convolution, as described in [5], between the image and two kernels, Equation 16 as well as its complex conjugate where h is the kernel size.

$$K(\mathbf{x}) = e^{\frac{-2\pi i \mathbf{x}}{h}} \tag{16}$$

The convolutions where performed four consecutive times to derive the STFT for the different frequency intervals specified by the kernel size. The imaginary and real parts of the STFT computations where transformed into a binary representation by encoding negative values to zero and positive values to one. By converting the binary values to decimal numbers and creating a histogram containing the amount of occurrences there where of each number an image descriptor could be derived.

The LPQ method was applied in two different steps when performing face recognition. The first step applied LPQ to all images in the database and stored the resulting descriptors in a file. This way no repeated computations had to be performed on the images in the database when performing recognition. The second step is the verification step where an image is attempted to be matched against one of the descriptors previously computed. First LPQ is applied to the image so that a descriptor is obtained. This descriptor is then compared to all descriptors from the images in the database and the differences between them is computed using the Euclidean distance. The descriptor with the smallest difference is chosen as the best match unless the difference is larger than a set threshold in which case the conclusion can be drawn that there is no matching image in the database.

4 Result

In this section we present how our algorithm performs concretely through benchmark tables as well as subjectively through images.

4.1 Face detection

The face detection part of our algorithm performs flawlessly on the original images in the given dataset which can be seen in Figure 12 as most outputs from the images in the given dataset is included in it. That it performs well on these original images is not a coincidence as the algorithm was developed and verified through testing on the given datasets. In Figure 15 we showcase how tolerant our algorithm can be for various conditions in certain images.

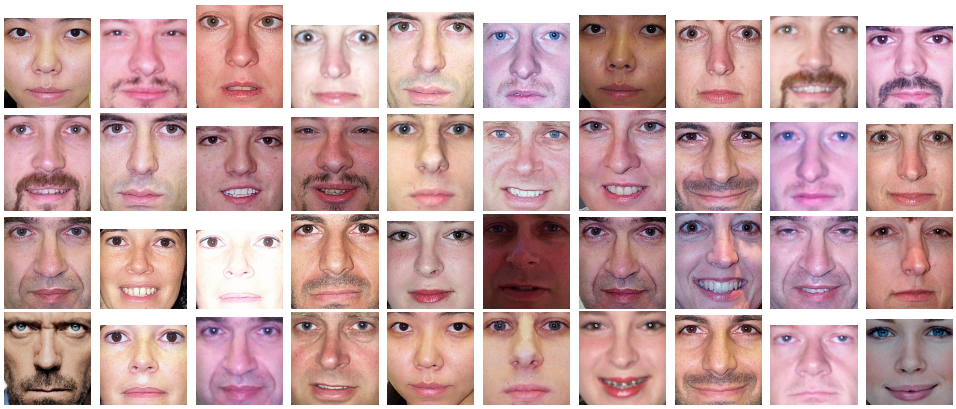


Figure 12: Successful experimental results of face detection.

However when our algorithm is faced with tougher conditions, for example blurry or very small images it somewhat breaks down. Examples of failed face detection can be seen in Figure 13 and 14. This is presumably due to hard coded thresholds and kernel widths for the morphological operations as well as for the methods themselves, especially the part were we derive the eye candidates. Detailed results where we have made the images harder to analyze through adding artificial artifacts can be looked up in Table 1.

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

Figure 13: A case where the proposed algorithm fails to detect the correct face due to an invalid extraction of the eyes from the eye map. (a) show the input image, (b) the correctly extracted face mask, (c) the proper eye map while (d) show the output image where the extraction of the eyes using the Circular Hough Transform fail by mixing up the left nostril with the right eye.

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

Figure 14: A case where the proposed algorithm fails to detect the correct face due to a lack of eye candidates for both eyes. (a) show the input image, (b) the filtered face mask that gives the insufficient eye candidates in (c) and lead to the invalid output (d).

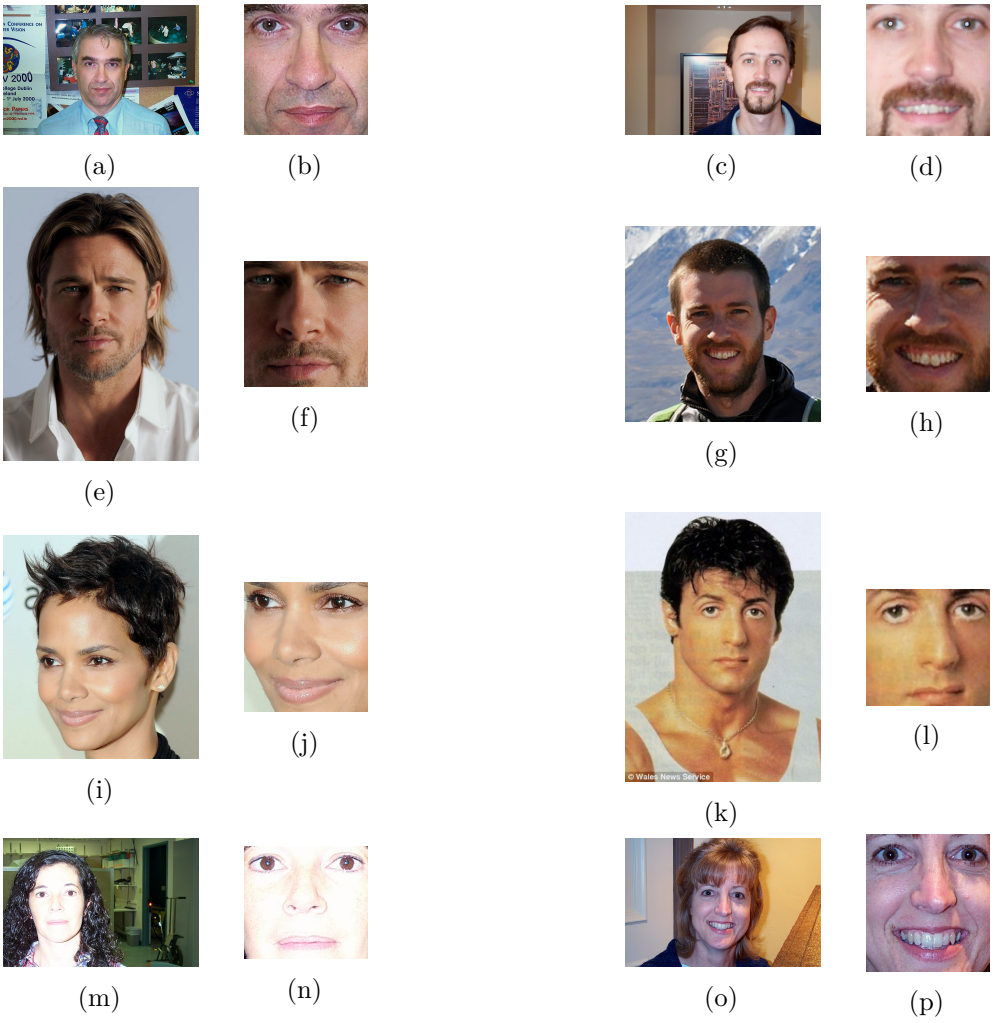


Figure 15: Results of the face detection phase depicting the algorithm's tolerance for varying environments *(a-d)*, lightning conditions *(e-h, m-p)*, poses *(i-j)* and amount of skin color in the image *(c-d, k-l)*.

Table 1: The resulting hit ratios over all images for different test scenarios.

Parameters	Access (db1)	No access (db0)	Hard (db2)	Test images
Blur, $\sigma = 0.5$	94%	75%	95%	50%
Blur, $\sigma = 1.0$	88%	50%	84%	50%
Blur, $\sigma = 2.5$	19%	25%	84%	33%
Tone, I = 0.5	94%	100%	95%	94%
Tone, I = 0.7	94%	100%	95%	89%
Tone, I = 1.3	75%	75%	87%	61%
Scale, s = 0.9	100%	50%	95%	94%
Scale, s = 1.1	100%	100%	97%	94%
Scale, s = 2	100%	50%	97%	50%
Rotation, $\theta = 5°$	94%	75%	97%	78%
Rotation, $\theta = 10°$	94%	75%	95%	78%
Rotation, $\theta = 20°$	88%	50%	95%	72%

A large test batch were performed to test different types of image problems and conditions. The tested conditions include varying illumination, scale, rotation and blur. All cases were individually reviewed in order to determine a hit, meaning that the face is extracted correctly. Furthermore the blur tests were done by using a Gaussian low pass filter while varying the standard deviation parameter σ. The tone level tests were done by decreasing or increasing the intensity value at each pixel with a global percentage parameter I. For example $I = 0.5$ corresponds to decreasing the intensity of the image with 50%. Moreover all images were rotated with an angle θ and scaled with a percentage parameter s.

Table 1 presents the hit ratio for the face detection with different types of test images and databases. The images in the database *Access* are persons with access permissions and with different types of background environments. *No access* images picture persons that should not be granted access. The original images have homogeneous intensities and backgrounds. The database *Hard* has images with highly varying backgrounds and illumination conditions. Finally, *Test images* is a collection of randomly selected images with various conditions.

4.2 Face recognition

In this section the results obtained by the face recognition algorithm is presented. The effects of performing face detection have been ignored in these results.

Table 2 show the results of applying the face recognition algorithm to the test data set.

Table 2: Results of performing face recognition on the test data set.

Number of images	False positives	False negatives	Recognition Rate
78	0	10	86%

Since the LPQ method used for face recognition is supposedly blur invariant, the effects of applying different types of blur to the input images were examined to verify this claim. Figure 16 shows how the recognition rate is effected by applying Gaussian blur to the input images while Figure 17 shows the effect caused by a linear motion blur filter.

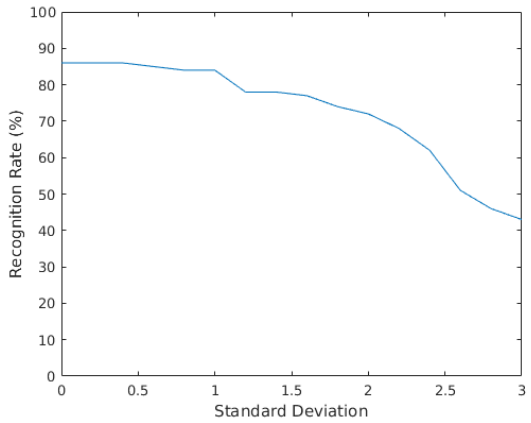


Figure 16: Shows how applying Gaussian blur effects the recognition rate.

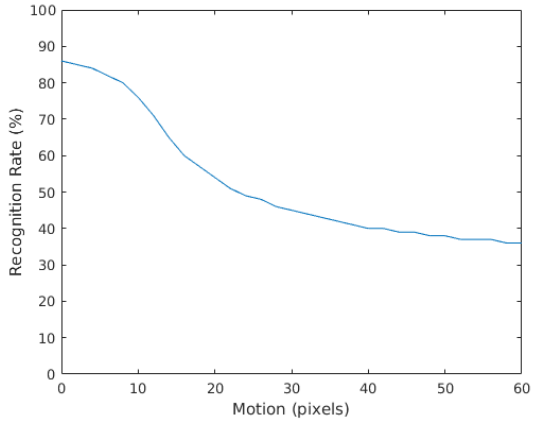


Figure 17: Shows how applying linear motion blur effects the recognition rate.

Figure 18 shows the result of decreasing the tone value and Figure 19 shows the result of increasing the tone value.

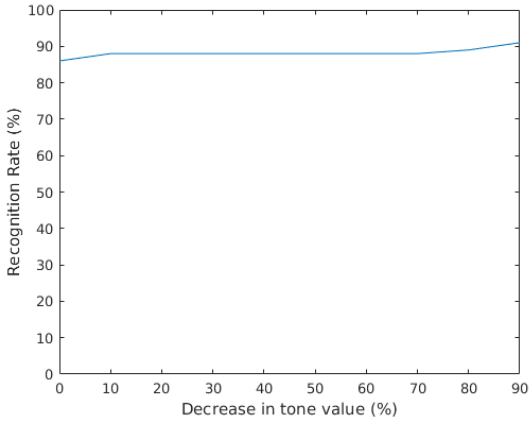


Figure 18: Shows how reducing the tone value effects the recognition rate.

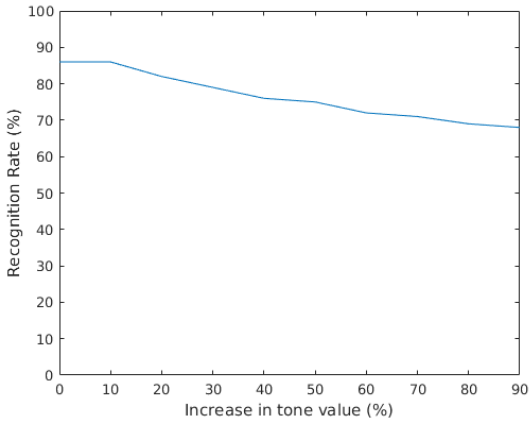


Figure 19: Shows how increasing the tone value effects the recognition rate.

4.3 Overview

Table 3 shows the result of running the entire pipeline including both face detection and face recognition.

Table 3: Results of performing both face detection and face recognition on the test data set.

Number of images	False positives	False negatives	Failed detections	Recognition Rate
78	0	10	4	82%

5 Discussion

We wanted to keep the preprocessing part of our algorithm simple as we found the other stages of the human face authentication pipeline more interesting. In hindsight though we recognize that this phase becomes the foundation determining the amount of work as well as the performance of the other phases. Therefore we should perhaps have put more effort into implementing an algorithm that works better for various lightning conditions and that also normalizes the colors better. An improved version could for example have led to that the proposed algorithm would have been able to correctly detect the face seen in Figure 20.

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

Figure 20: A case where the proposed algorithm fails to detect the correct face due to a too sparse estimated skin mask. (a) show the input face, (b) the estimated skin mask, (c) the resulting invalid face mask and (d) the output.

For face detection we decided to go by the article in [1] that is based on the extraction of eyes and mouth through the use of the relations between colors in faces. However we also looked into algorithms that require training and uses features, like the *Viola Jones* [8] algorithm.

The aim of the segmentation steps taken during the face detection phase were to reduce the search space for the Circular Hough Transform when looking for the eyes. We do though recognize that this process might be overly complicated as the single purpose for it is to make it easier to locate the eyes. Although we tried to extract the eyes directly from the original eye map (Figure 7*c*) in various ways, for example by thresholding while taking the mean value and the variance into consideration, we did not found one that was satisfying. We do however believe that a better preprocessing stage could have mitigated this problem.

Another method we looked into were to utilize the ability of Eigenfaces and its face space in order to recognize whether a blob found, in for example the mask of Figure 4*f*, were an actual face or not. However we left the implementation this method for a future study.

Furthermore do we believe that dynamic thresholds and kernel widths

would have made our implementation more robust. Especially in regard to faces and images of few pixels, like those in the given database *db0*. In order to make our implementation less sensitive and more tolerant we have for example been working with circular disks as kernel elements for dilation and erosion and thresholds with large margins. More over do we believe that the face detection part of the algorithm could be improved by utilizing the facial expression methods explored in [9]. This as we have found out through experimentation that the mouth plays a large role during the recognition phase. For example when we run tests only including the eyes and nose we received a larger recognition rate.

The chosen approach for performing face recognition became the LPQ method but before settling on that another method was examined. This other method was the popular Eigenfaces from [4] that was also described in the theory section. Eigenfaces proved easy to implement and initial tests where somewhat promising. However multiple sources including [10] claims that Eigenfaces performs poorly under certain conditions such as tone variations. Due to this other alternatives where examined and the LPQ method appeared as a promising candidate with promises of simplicity and blur invariance. While the theory behind LPQ is comprehensive it is certainly no more difficult than Eigenfaces and following the advice of the original authors the implementation could be made very efficient.

In order to verify the claim that LPQ is blur invariant tests where conducted where varying degrees of different types of blur was applied to the input image as seen in Figure 21. The results of these tests, as presented in the results section, shows that the algorithm performs well for all but the most extreme cases. While the blurring certainly effects the results in a negative manner this is to be expected since the claim that LPQ is blur invariant is merely a theoretical one. The blur invariance comes from the assumption that the phase of the Fourier transform of the blur function is zero which in practical applications will never be true. A more correct term would be to call the method blur resistant.

(a) (b) (c)

Figure 21: A comparison of different types of blur applied to the same image. (a) shows the original image. (b) shows the effect of applying Gaussian blur to (a) using a standard deviation of 3.0. (c) shows the result of a 60 pixels wide linear motion blur filter applied to (a).

To compare LPQ to Eigenfaces another set of tests where performed where the tone value of the input images varied. This was claimed to be an issue for Eigenfaces, but the tests performed showed that LPQ could handle it without reducing performance significantly. Some of the tests even suggested that decreasing the tone value could improve recognition rates by a few percent. This may be caused by some oversaturated images that are matched to the correct descriptor but whose distance value is above to allowed threshold. By reducing the tone value the distance between the descriptors are also effected so that these image now pass as they should.

An area of improvement for the face recognition algorithm is how the threshold that determines whether or not an image should be considered a math is chosen. Currently this value is chosen by conducting a number of tests with images that have no match and choosing a threshold so that no match is found for any of these images. If the Eigenfaces method had been chosen over LPQ a more accurate way of determining this would have been possible. Eigenfaces revolves around finding a subspace for all descriptors of the images in the database and therefore the boundary of this subspace can be used as a threshold. This is unfortunately not possible for LPQ.

6 Conclusion

To summarize, we deem that the proposed algorithm and implementation of the human face authentication pipeline achieved works fairly well for a large plethora of images.

7 References

1. Andrew Senior, Rein-Lien Hsu, Mohamed Abdel Mottaleb, and Anil K. Jain. Face detection in color images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):696–706, May 2002.
2. Inc. The MathWorks. Imfindcircles. `http://se.mathworks.com/help/images/ref/imfindcircles.html`, 2015.
3. L.Sirovich & M.Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America*, 1987.
4. M. Turk & A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 1991.
5. Timo Ahonen & Esa Rahtu & Ville Ojansivu & Janne Heikkil. Recognition of blurred faces using local phase quantization. *Pattern Recognition, 2008. ICPR 2008.*, pages 1–4, 2008.
6. R.J. Beerends & H.G. ter Morsche & J.C. van den Berg & E.M. van de Vrie. *Fourier and Laplace Transforms*. Cambridge University Press, 2003.
7. Wes Wallace & Lutz H. Schaefer & Jason R. Swedlow. Introduction to deconvolution. `http://www.olympusmicro.com/primer/digitalimaging/deconvolution/deconintro.html`, December 2015.
8. Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.
9. Narendra Patel and MukeshA. Zaveri. Automatic faps determination and expressions synthesis. In David C. Wyld, Jan Zizka, and Dhinaharan Nagamalai, editors, *Advances in Computer Science, Engineering Applications*, volume 166 of *Advances in Intelligent and Soft Computing*, pages 1–11. Springer Berlin Heidelberg, 2012.
10. Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.