Face Recognition

Cornelia Aksell, Hanna Johansson, Madeleine Rapp, Sandra Pettersson

TNM034 - Advanced Image Processing
Linköping University, Norrköping
Media Technology

corak923@student.liu.se hanjo306@student.liu.se
madra484@student.liu.se sanpe282@student.liu.se

December 10, 2017

Abstract

Face recognition and face detection are large and active research areas, constantly improved and new softwares developed. More and more people use it and probably more often than they think. Tagging your friends in images on Facebook is just one application of using face recognition.

This paper targets the task of detecting a face in an image and determining whether the face belongs to a known individual or not. The project carried out was to implement a software in Matlab *for face recognition based on lectures given in the course and related research papers.*

Keywords: Face recognition;* Matlab *software

1. Introduction

This is a report in the course *TNM034 - Advanced Image Processing* at Linköping University. The report presents the project of doing Face Recognition in Matlab and its aim, the method, implementation and results. A discussion in the end will cover improvements that could be done to the software as well as difficulties during the implementation of it.

1.1. Aim

The aim of the project was to implement a software for face recognition in Matlab. The implementation of the software would also require to solve several sub problems, such as preprocessing, face detection and feature extraction. In the end the goal with the project was to give the implemented software an image of an unknown face and the program should detect the face and determine whether it belongs to a known person. If the face do belong to a known person the program should return the information of which person it belongs to.

2. Background and requirements

To be able to implement a face recognition software a known database of faces is needed. A database called 'db1', consisting of 16 faces of 16 different individuals, were therefore given at the beginning of the project. The requirement for the program was to at the end be able to detect if the input image belongs to any of the individuals in 'db1'. If this is the case the identity number of the individual in the database should be returned. If the input face does not belong to any of the known individuals the number '0' should be returned. The database 'db1' can be seen in Figure 1 [1].

Further requirements for the program was to handle certain modifications of the faces

belonging to the individuals in the database. Modifications that had to be taken into account when implementing the software were the following:

- translation of the face in the image,
- rotation (max +/- 5 degrees),
- scaling (max +/- 10 percent) and
- change of tone values in the image (max +/- 30 percent).

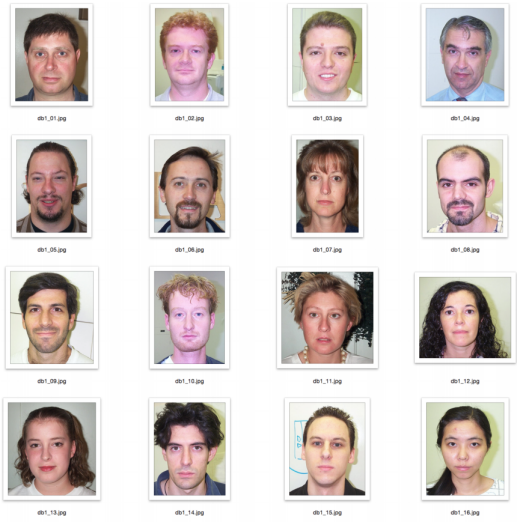

Figure 1: *The database called 'db1'.*

Additional restrictions were that the program had to be implemented in MATLAB and already existing functions in the MATLAB environment could be used, but no other program libraries were allowed.

3. METHOD

The software was implemented mainly by following the face recognition pipeline, see Figure 2 [2]. The different parts of face detection, face alignment, appearance normalization, feature description, feature extraction and matching were split up within the group. Mainly we worked in pairs, two and two, using pair programming and discussing the process within the pair and between the pairs throughout the project.

One pair were responsible for the face detection and the three last steps in the pipeline, while the other pair focused on the face alignment and appearance normalization. We realized that the face alignment and the problem of finding the eyes and mouth were more difficult than expected and we had problems with these parts, which will be discussed in Chapter 7.

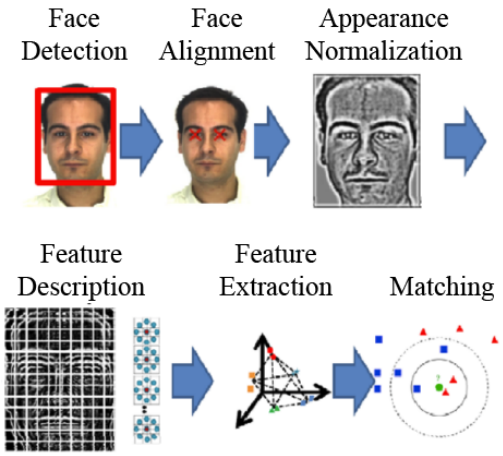

Figure 2: *The face recognition pipeline.*

4. IMPLEMENTATION

This chapter covers the implementation of the different parts of the software and follows the order of the face recognition pipeline.

4.1. Image editing

The very first step was to edit the images used. Both the images in the database and the images matched can be affected by the light source used and the image acquisition. Therefore the images may vary in color temperature, lighting and so forth. This will affect the color channels in the image which then will affect the difficulty of detecting the face. Because of this the differences between the picture and reality were edited.

First the image was converted into gray, using `rgb2gray` to get the mean luminance. After this `imadjust` was used to adjust for the brightness or lack of brightness in the image. All the different color channels in the image was then extracted, where the mean value was adjusted to make the images more alike. The channels was then put together again to create a new, edited, image.

The result of the function *editImages* can be seen in figure 3 and Figure 4.

Figure 3: *The original image compared to the edited image.*

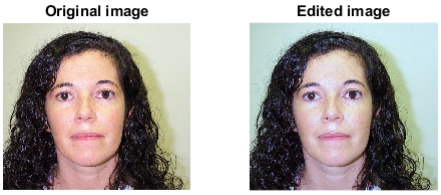

Figure 4: *The original image compared to the edited image.*

4.2. Face detection

When the images had been adjusted they were individually separated into subchannels of the *YCbCr* color space with `rgb2ycbcr`. The colorspace *YCbCr* is made of *Y* which is the images luminance channel, *Cb* is the chroma blue channel and *Cr* is the chroma red channel. The chroma red channel was the channel which had the best contrast between the face and the background in the images, which can be seen in Figure 5. The face mask should be a binary image, but before the image could be turned into a binary a threshold had to be found. The threshold was found by using `graythresh` on the color channel with the best contrast, which in this case was the chroma red channel. When the threshold was found the chroma red image was turned into a binary image with `im2bw` using the threshold. When this was done there were some images that had holes in them or dots outside of the face, which preferably should not be there. By using different morphological operations such as `imdilate` the face mask was improved on several images. Lastly the original image and the face mask were combined by multiplying them.

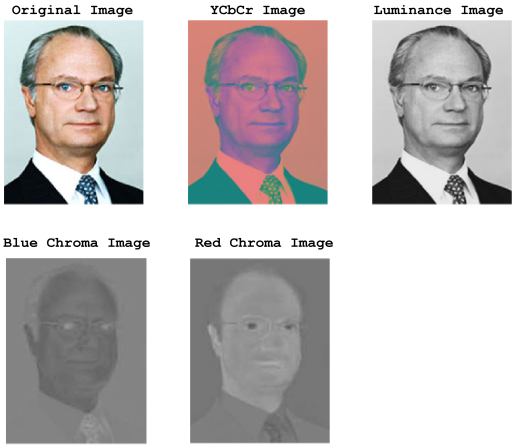

Figure 5: *The different color channels in YCbCr.*

4.3. Face alignment

This part of the program includes creating an eye map and a mouth map to detect eyes and mouth in the face to be classified. The eyes and mouth are facial features commonly used for face recognition since they have special features that quite easily separates them from the rest of the face. They also look pretty similar for all humans, regardless of other facial features present.

For this function of the program we use the face mask created in the previous step and also the edited image. From the edited image both the luma and the chroma is used to create the eye map and mouth map.

4.3.1 Eye map

The first map that we created was the eye map derived from the chrominance components of the input image, our *eyeMapC*. The basic idea of this map is that usually around the eyes a high value of Cb and a low value of Cr is found. The equation for this eye map can be seen in Equation 1. All components C_b^2, C_r^2, C_b/C_r were normalized to the range [0, 255]. For the normalization we created a separate function which is able to normalize an input matrix between two values x and y. [3]

$$EyeMapC = \frac{1}{3}\{(C_b^2) + (\tilde{C_r})^2 + (C_b/C_r)\} \quad (1)$$

The second step was to create yet another eye map, but this one is built from the luminance component. This map is called the *eyeMapL* and the equation for it can be seen in Equation 2 [3]. Here dilation and erosion operations are done by MATLAB's predefined functions `imdilate` and `imerode`. Before performing those operations the input image is however changed to gray-scale by `rgb2gray` and the chroma is enhanced by doing histogram equalization with `histeq`.

$$EyeMapL = \frac{Y(x,y) \oplus g_\alpha(x,y)}{Y(x,y) \ominus g_\alpha(x,y) + 1} \quad (2)$$

Further on a combination of the two eye maps are created to create one single, final eye map. The two maps are combined through a multiplication operation, but other options such as MATLAB's `imfuse` was also tested. To refine the final eye map a set of additional operations are performed, such as erosion, normalization and also the face mask is applied again.

The last step before returning the final eye mask to the next step in the pipeline was to remove some unnecessary parts of it. We made some assumptions that the eyes would not be at the very edges of the image, neither on the sides or at the top or bottom. These pixels were therefore set to black and afterwards the function `bwareafilt` was used to find the two biggest objects in the mask. We then considered those two objects to be the eyes of the face in the image.

4.3.2 Mouth map

The mouth map was created in a separate function where the input image first was converted to the *YCbCr* color space by the function `iycbcr`. Thereafter it was split into the different components of Equation 3 and Equation 4. When the calculations had been made by the equations the mouth map was normalized to lie in between 0 and 1. [3]

$$MouthMap = C_r^2 \cdot (C_r^2 - \eta \cdot C_r/C_b)^2 \quad (3)$$

$$\eta = 0.05 \cdot \frac{\sum_{(x,y)\in\mathcal{FG}} C_r(x,y)^2}{\frac{1}{n}\sum_{(x,y)\in\mathcal{FG}} C_r(x,y)/C_b(x,y)} \quad (4)$$

Description of elements in the equations above:
$\mathcal{FG}$ = face mask
η = estimated as a ratio of the average C_r^2 to the average C_r/C_b
$\oplus$ = dilation
$\ominus$ = erosion
[3]

Additional refinements were done by performing dilation and erosion with `imdilate` and `imerode`. Further on the image was converted to binary by `im2bw`, using the function `graythresh` to find an appropriate threshold level.

Two last steps were then performed before the final mouth map was returned as the return

argument from out mouth map function. The first step was to remove unnecessary pixels. We made the assumption that the mouth would not be found at the very edges of the image (within 10 percent of the image's width from the left or right edge of the image) or at the upper half of the image. The second step was to use the function `bwareafilt` to extract the two biggest areas in the mouth map.

We decided to use the two biggest areas rather than the one single biggest area to be more flexible with the mouth map. We realized that the biggest area was not necessarily always the mouth, but some other object in the face, so therefore we made the decision to keep the two biggest areas, which in the end gave a better result.

One example of an eye map and a mouth map along with the face mask can be seen in Figure 6

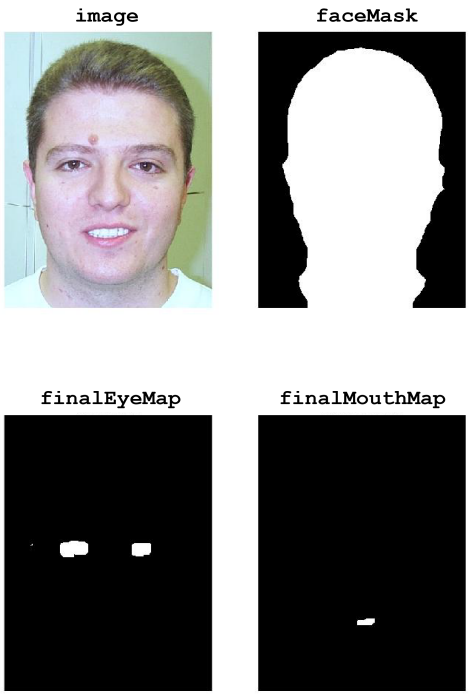

Figure 6: *The original image, face mask, eye map and mouth map of the image 'db1_03'.*

4.4. Appearance normalization

The first step in appearance normalization was to rotate the images if needed. This was done by using `imrotate`, which takes an image and a clockwise rotation angle as input arguments. Because of the fact that the eyes should be aligned, the rotation angle could be found by taking the angle between the left eye and the right eye. This was done by using the trigonometric function *rotationAngle*, see Equation 5.

$$rotationAngle = taninv \cdot \frac{opposite}{adjacent} \quad (5)$$

To be able to compare images with the dataset they all had to be the same size with the same dimensions. The face of the person in the images should also have similar proportions compared with the image size. This had to be done before cropping the images so that the face was in the middle of the image and did not end up outside of the cropped image. To scale `imresize` was used. The command used an image and a scale factor as input arguments. The scale factor was found by dividing a wanted length between the eyes with the actual length. After scaling the image the new eye coordinates were found by applying the scale factor to the old ones.

For the matching to work all images had to be of the same size, otherwise the matrix dimensions would be different. To achieve that the `imcrop` was used. To find the starting point for where to start cropping the coordinates for the left eye were subtracted with fixed numbers so that the crop would not start from the left eye. Subtraction was used because origo was placed in the top left corner on the images and not in the bottom left as it normally is. The end coordinates for the crop were picked so that the crop size would be of the desired size. The rotated, scaled and cropped image can be seen in Figure 7.

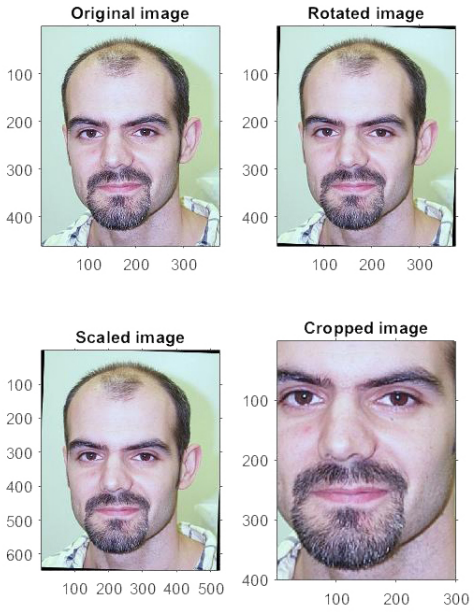

Figure 7: *The original image, the rotated image, the scaled image and the cropped image.*

4.5. Feature description and extraction

To be able to compare and match a query image with the database, the eigenfaces needed to be calculated for all the images used.

4.5.1 Dimensionality reduction

Reducing of the number of features is done by Principal Component Analysis, PCA, and means that only the principal components is analysed [4]. The principal components are determined by the eigenvectors that capture the most variance from the original image, when projected onto the subspace created by themselves. This allows a dataset with n dimensions to be represented by a set of k dimensions, which is also known as a dimensionality reduction algorithm.

4.5.2 Eigenfaces

Before all the calculations it was necessary to define some variables that would be used throughout the functions. The facial image dimension used is calculated with Equation 6. [5]

$$n = rows \cdot columns \tag{6}$$

Since the normalized image was cropped to the size of 300x400 pixels, this gave a dimension of 120 000. The number of images used in the training set is called M and was set to 16 because only 'db1' was used. This was because the first database only contains 16 images in total. The number of eigenfaces used, K, could be chosen but was not allowed to exceed the number of images in the training set. In this case K was set to 16.

The input images have been normalized for rotation, scaling and tone values and the first step was to read all M images, $I_1, I_2, ..., I_m$, that the database would consist of. These all had the same size as stated in the previous paragraph. Each image, I_i, was then represented as a n-vector x_i, as seen in Equation 7. [5]

$$I_i = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1c} \\ a_{21} & a_{22} & ... & a_{2c} \\ ... & & & \\ a_{r1} & a_{r2} & ... & a_{rc} \end{bmatrix} x_i = \begin{bmatrix} a_{11} \\ a_{21} \\ ... \\ a_{r1} \\ ... \\ a_{1c} \\ a_{2c} \\ ... \\ a_{rc} \end{bmatrix} \tag{7}$$

After this, all images could be treated as point vectors in a high-dimensional space. The next step was to find the average face vector, μ, that represent the mean face for the training set. This was done with Equation 8.

$$\mu = \frac{1}{M} \cdot \sum_{i=1}^{M} x_i \tag{8}$$

Then the mean face, μ, was subtracted from each face vector, x_i, see Equation 9. Each vector φ_i represents the difference between the mean face and each face vector. [5]

$$\phi_i = x_i - \mu \tag{9}$$

The covariance matrix, C, is calculated with Equation 10 where $A = \begin{bmatrix} \phi_1 & \phi_2 & ... & \phi_M \end{bmatrix}$.

$$C = A \cdot A^T \tag{10}$$

C have the size $(n \cdot n)$ and A have the size $(n \cdot M)$. The problem that arose was that the matrix C was too large to calculate in MATLAB. The way around the problem was to calculate the matrix C in another way, see Equation 11, that would get the size $(M \cdot M)$ instead of $(n \cdot n)$.

$$C = A^T \cdot A \tag{11}$$

The eigenvectors, u_i, was computed with MATLAB's predefined functions $[\texttt{V}, \texttt{D}] = \texttt{eig}(\texttt{C})$ and was afterwards multiplied with A. See Equation 12.

$$u_i = A \cdot V \tag{12}$$

This gave the M best eigenvectors, $u_1, u_2, ..., u_m$. When the eigenvectors u_i had been calculated they were reshaped into matrices that gave the eigenfaces. The last step was to get the weights. They were given by equation 13.

$$w_i = u_i^T \cdot \phi_i \tag{13}$$

Each normalized image would now be represented as a k-vector, the weights, see Equation 14.

$$\Omega_i = \begin{bmatrix} w_1 \\ w_2 \\ ... \\ w_K \end{bmatrix} \tag{14}$$

This was first done to the M images in 'db1' where three of the most important variables, the mean face, μ, the eigenvectors, u_i, and the weights, w_i, were saved to be used to match a query image to the database. [5]

4.6. Matching

The first step towards matching the images is to do all the above for the image that should be matched. The query image's weight, Ω, was then compared to the weights of each image in the database, Ω_i by computing the distance measure between the weights for the query image to each weight vector for the M images in the database. The distance is computed with Equation 15.

$$e_i = \left\| \Omega - \Omega_i \right\| \tag{15}$$

When the distances, e_i, was calculated the closest match was found by MATLAB's predefined function `min`. To know if the closest match actually was the same person a threshold, Θ, needed to be added.

$$\Theta = 16.5 \tag{16}$$

The threshold was changed multiple times to work with different types of images but was later set to 16.5, see Equation 16. It was decided that it was better to have a higher False Reject Rate, FRR, than a high False Acceptance Rate, FAR. [6] This also did so that some query images did not pass but still had the correct image as the closest. But since others had a smallest e_i that was around 20 it was better to get a few rejections then to accept the wrong image. The exact result of the tests that were a contributing factor to the threshold can also be seen in Appendix A, Appendix B and Appendix C.

5. Testing and results

By the end of the project it was decided to perform some tests on the software by sending different images as input to it.

The tests were done with the image database 'db1', see Figure 1, some additional test images that we created ourselves, see Appendix D, and last we decided to also test the database 'db2'. The database 'db2' contains a set of images of some of the persons in 'db1', but captured under different conditions. The results of the testing is presented in this chapter.

5.1. Database 'db1'

The first test that we did was, intuitively, to send all images from 'db1' to the software as input to see how the program corresponded.

As expected, and desired, the software returns the correct ID for each input image. This comes as no surprise since the faces in the database 'db1' was used to create the program's database. The score is 0 for the best match for each image since the face is exactly the same as the one in the database - in other words, a perfect match.

5.2. Test images

The second test was done using test images that we had created ourselves. The group decided to use two people from 'db1', quite randomly chosen, but one male ('db1_09') and one female ('db1_13'). According to the requirements listed in Chapter 2 the software should be able to handle translation, rotation, scaling and change of tone values in the image. Therefore a set of images were created from the faces of the two people mentioned above meeting those requirements. The images can be seen in Appendix D.

The test images were sent in to the program as input and the resulting output was noted as well as the score of this best match, which can be seen in Appendix C. All images except for one returns the correct ID of the person sent in to the program. The one image that gets an incorrect output is the image of person 09 with an increased toning of 30 percent.

A graphical representation of the results of this test can be seen in Figure 8, where the image with the incorrect output is not included. Shown in the figure are the scores for each image respectively. The yellow bars represents the images derived from 'db1_13' and the blue bars corresponds to 'db1_09'. As can be seen the lowest scores are found for scaling -10 degrees and translation. The program handles rotation quite well too, for both the male and the female face and in both positive and negative direction. The higher scores are found for toning and it is also here we have the one incorrect result for the image 'db1_09_tone_+30' (included as a score of 0 in the diagram and therefore not visible).

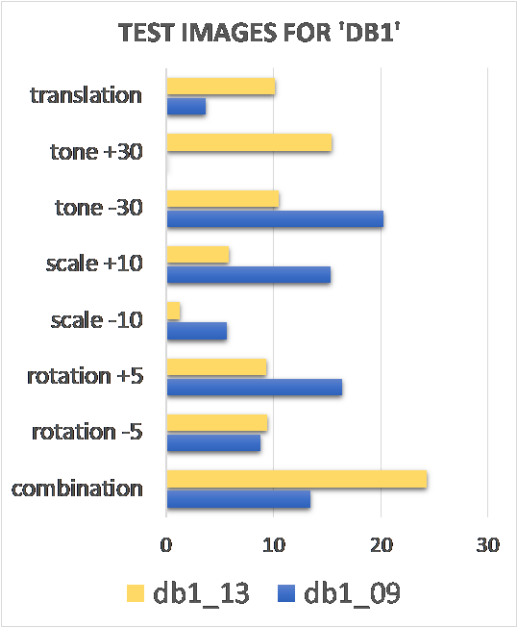

Figure 8: *Results after testing the test images that were created from the two images 'db1_09' and 'db1_13' from 'db1'.*

The conclusion of this particular test is that the program works for all the requirements given in the beginning of the project work, except for one case out of 16. Translation is handled well and so is scaling and rotation too. Toning works correctly, but requires a higher threshold, and this could also be the case for certain images with a combination of the different modifications. This does however depend on the combination itself, as also can be noted in the figure (the yellow bar indicates a much higher score for 'db1_13_combination' compared with 'db1_09_combination'.

5.3. Database 'db2'

The last test that we performed was on 'db2', which contains a set of images of some of the persons in 'db1' captured under different conditions. These conditions were not really addressed during our work with developing the software, but we were however interested in testing how well our program would do with

more difficult images as input and decided to run a test.

The database 'db2' contains images classified in four different categories. The categories are 'BL' (unfocused, blurred face), 'CL' (cluttered background), 'EX' (different facial expression) and 'IL' (faces captured under different illumination properties). The results of giving these images as input to the program can be seen in Figure 9.

As can be seen in the results the program handles blurry faces pretty well, with almost 60 percent of the blurry images being classified correctly. Closer to 45 percent of the images with cluttered backgrounds are also correctly classified, while facial expressions and different illumination properties causes incorrect results.

In this case it should however be mentioned that by 'correct' we mean that the best match for the input image is the correct person, but the output is not necessarily the correct person (the program returns '0' for several of the images). This is because the threshold used during the testing, 16.5, is quite low compared to the scores of the best matches. The scores of the correct best matches varies from 12.07 to 28.53, which can be seen in Appendix B. In other words, if the threshold would be set to 29 all these images would return the correct ID of the person, but on the other hand several other images would probably be misclassified.

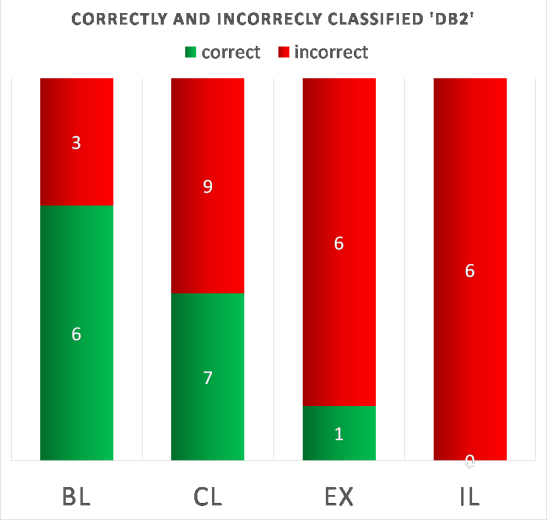

Figure 9: *Results after testing the software with db2.*

6. Conclusion

The general conclusion of the results is that the developed software accepts the correct faces and rejects the faces that are not present in the chosen database.The program can handle translation, rotation and scaling, with the requirements given in Chapter 2.

When it comes to the images that have mixed tone values there is a problem. When the image is slightly darker, there is no problem, but the when the tone value is increased the face becomes too bright. This causes the program to give a face mask that is incorrect, which then causes it to misplace the eyes. This mistake of course contributes to the program giving an incorrect ID as output.

7. Discussion

In this chapter improvements and ideas for future development of the program is discussed.

7.1. Method

As was mentioned in Chapter 3 we had a few problems during the development of the software and the method of how we worked.

In the beginning the plan was to divide the development of the software in two parts between the two pairs of developers in the group - one pair doing the first half of the face recognition pipeline and the other pair doing the second half of the pipeline. We realized, however, pretty fast that this was not an optimal distribution of work within the group.

The work on the face alignment and the problem of finding the eyes and mouth were more difficult than expected and took a lot longer time than we first planned. Now by the end of the project we see that we should maybe have read more literature or asked for some input ideas from other groups in the class, because we spent a lot of time trying things that in the end didn't work well and that we therefore had to redo several times.

Another thing that we could have done differently, or more of, is the testing of the program when we considered it done. For example we realized very late that we could have downloaded the full dataset "Faces 1999" and used some images from that to test our program [1]. The full dataset contains 450 facial images of 27 unique people and 'db1' is built up by 16 of those 27 people. The remaining 11 people in the dataset "Faces 1999" could in other words have been used instead of (or as a compliment to) the test images that we created ourselves.

Apart from the above our method of working with the project worked well and the decision of working in pairs had advantages too. We could discuss things as we were working on them, we helped each other to spot mistakes in the code and it was always two people in the group that had full control of each individual part of the program. We could of course had split the work between the four of us individually, but we found this to work well throughout the project.

7.2. Improvements

Different tone values is something that the program could be better at handling, especially in the case were the tone value is high and the image becomes very bright. This is something that could be implemented better with a bit more time on our hands. There might also be another solution completely that is connected to the way we find the face mask. Right now we do not look at different skin color samples in the color space that we use, which could be something that eventually affects the tone values.

The program could be better at handling clutter in the background. This would be improved if we implemented the face detection a bit better. As previously mentioned we do not currently use something to tell if something is in the same range as skin color in the color space. This would help detecting if something should be included or not in the face mask.

The program could also be implemented in a way that makes it faster, right now it runs very slowly when it goes through the database.

7.3. Future development

If the program were to be developed further a thing to implement would be the ability to recognize a person even thought the person has different facial expressions. This would make the program more usable since a person rarely has the same facial expression on multiple images.

Another idea would be for the program to be able to detect multiple faces in the same image and identifying the persons.

References

[1] Computational Vision at Caltech. *Faces 1999 (Front)* (March 17, 2005) Available at: ¡`http://www.vision.caltech.edu/html-files/`¿

[2] D. Nyström, *TNM034 Advanced Image Processing - Lecture 1: Course introduction* (October 2017). Le1_Course_introduction.pdf.

[3] R. Hsu, M. Abdel-Mottaleb, A. K. Jain. *Face Detection in Color Images* (2002). IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 5.

[4] J. Lever, M. Krzywinski, N. Altman. *Points of Significance: Principal component analy-*

sis (June 29, 2017) Available at: ¡https://www.nature.com/articles/nmeth.4346¿

[5] S. Trivedi. *Face Recognition using Eigenfaces and Distance Classifiers: A Tutorial* (February 11, 2009) Available at: ¡https://onionesquereality.wordpress.com/2009/02/11/face-recognition-using-eigenfaces-and-distance-classifiers-a-tutorial/¿

[6] D. Thakkar. *False Acceptance Rate (FAR) and False Recognition Rate (FRR) in Biometrics* Available at: ¡https://www.bayometric.com/false-acceptance-rate-far-false-recognition-rate-frr/¿

Appendices

A. Testresult for 'db0'

Image name	Output*	Lowest threshold	Comment about triangle
01	0	21,59	Correct
02	0	31,73	Correct mouth. One slightly misplaced eye. One wrong eye
03	11	16,12	Correct
04	0	22,48	Two slightly misplaced eyes. Correct mouth

*Output with threshold = 16.5. Output gives zero if the face does not exict in the database. If the face is found in the databse (with a threshold lower than 16.5) the output is the most similar face.

Columns marked green are the correct output.

B. Testresult for 'db2'

Image name*	Best match	Threshold	Second match	Comment about triangle
bl_01	01	18,12	11	Correct
bl_02	02	21,42	13	Correct
bl_04	03	23,59	2	Correct
bl_05	05	19,70	11	Correct
bl_06	09	20,87	11	Wrong mouth
bl_07	13	29,13	01	Correct
bl_10	01	21,91		Correct
bl_13	13	16,68	11	Correct
bl_14	14	25,51	01	Correct
cl_01	01	14,36	11	Wrong mouth. Slightly misplaced eyes
cl_02	05	37,97	3	Wrong triangle
cl_03	11	18,20	09	Correct
cl_04	04	19,90	09	Wrong mouth. Slightly misplaced eyes
cl_05	05	28,53	03	Mirrored eye
cl_06	**			
cl_07	07	15,08	11	Correct
cl_08	09	37,55	05	Wrong triangle
cl_09	01	19,92	09	Slightly misplaced eyes
cl_10	05	32,86	03	Wrong mouth. One correct eye
cl_11	11	12,07	01	Correct
cl_12	05	51,58	09	Right mouth, wrong eyes found
cl_13	13	14,57	11	Slightly misplaced eyes.Mouth wrong
cl_14	06	25,66	03	Wrong mouth. One correct eye
cl_15	01	23,7	13	Correct
cl_16	16	16,85	09	Correct
ex_01	13	19.18	01	Correct
ex_03	09	17,12	02	Correct
ex_04	09	25,22	16	Wrong mouth. One Slightly misplaced eye
ex_07	09	33,27	01	Correct
ex_09	09	20,31	11	Correct
ex_11	05	10,61	09	One wrong eye
ex_12	07	25,01	12	Correct
il_01	05	40.70	03	Wrong triangle
il_07	04	18,69	02	Wrong triangle
il_08	05	67.56	07	Wrong mouth. One slightly misplaced eye. One wrong eye xxxxxxxxxxx
il_09	05	41,02	07	Wrong mouth, one wrong eye
il_12	05	42,22	07	One slightly misplaced eye.
il_16	05	33,53	03	Slightly misplaced eyes.

* The categories are 'BL' (unfocused, blurred face), 'CL' (cluttered background), 'EX' (different facial expression) and 'IL' (faces captured under different illumination properties)
**Error in cropImage using imrotate.

Columns marked green are the correct output.

C. Testresult with selfmade images with two persons from 'db1'

Image name	**Output***	**Threshold**	**Second threshold**	**Comment about triangle**
09_combination	9	13,47	26,73	Correct
09_rotation_-5	9	8.75	28.23	Correct
09_rotation_+5	9	16.42	25,70	Correct
09_scale_-10	9	5.67	28,86	Correct
09_scale_+10	9	15,33	25,28	Correct
09_tone_-30	0 (9)	20.23	23.81	Correct
09_tone_+30	0 (10)**	33.30	34.62	Incorrect. Wrong eyes
09_translation	9	3.62	30.39	Correct
13_combination	0 (13)	24,28	29,37	Correct
13_rotation_-5	13	9.43	18.73	Correct
13_rotation_+5	13	6,29	18.89	Correct
13_scale_-10	13	1.19	21.32	Correct
13_scale_+10	13	5.80	24.84	Correct
13_tone_-30	13	10.43	22.17	Correct
13_tone_+30	13	15.44	31.87	Correct
13_translation	13	10.07	24.67	Correct

*Output with threshold 16.5. Second best match in parentheses
**Correct image is the 10th match

D. Test images for 'db1'

Figure 10: *Original.*

Figure 11: *Scale -10%.*

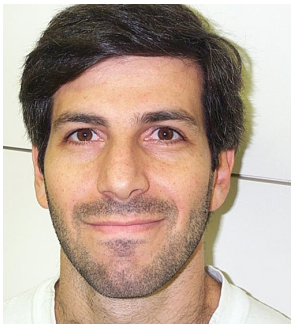

Figure 12: *Scale +10%.*

Figure 13: *Rotation -5 deg.*

Figure 14: *Rotation +5 deg.*

Figure 15: *Translated.*

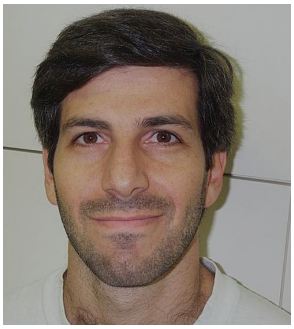

Figure 16: *Tone -30%*

Figure 17: *Tone +30%.*

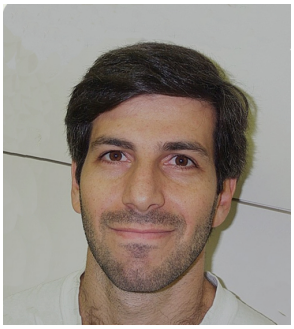

Figure 18: *Combination.*

Figure 19: *Original.*

Figure 20: *Scale -10%.*

Figure 21: *Scale +10%.*

Figure 22: *Rotation -5 deg.*

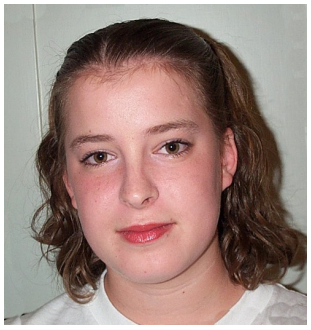

Figure 23: *Rotation +5 deg.*

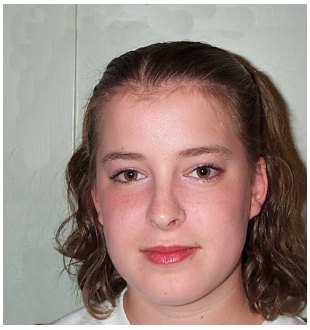

Figure 24: *Translated.*

Figure 25: *Tone -30%.*

Figure 26: *Tone +30%.*

Figure 27: *Combination.*