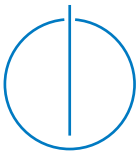
FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Evaluation of Feature Extraction Algorithms for Real-Time Face Recognition on Multiple Embedded Hardware Platforms

Amirali Amiri



FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# Evaluation of Feature Extraction Algorithms for Real-Time Face Recognition on Multiple Embedded Hardware Platforms

# Evaluierung von Algorithmen zur Merkmalsdetektion für Echtzeit-Gesichtserkennung auf Verschiedenen Embedded Hardware Platformen


| | |
|---|---|
| Author: | Amirali Amiri |
| Supervisor: | Dr. Claus Lenz |
| Advisor: | Prof. Dr.-Ing. habil. Alois Knoll |
| Submission Date: | 29.09.2015 |


I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 29.09.2015 Amirali Amiri

# Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Claus Lenz, for his support throughout the entire time I was working on this thesis. I also thank *open ideas GmbH* who gave me the opportunity to conduct my research in their company. Moreover, I wish to show my appreciation to the Autonomous Systems lab in the University of Texas at Arlington Research Institute (UTARI), which provided the embedded hardware platforms investigated in this thesis.

Most importantly, I thank my dearly loved mother and dedicate this work to her who has been a father and a mother to me as long as I can remember.

# Abstract

Face recognition is an interesting and challenging area of image processing. Human brain can recognize faces almost instantly and existence of variations in face such as expressions, pose, head rotation or illumination affects this capability only to some extent. For a computer, however, these variations can dramatically change the identification of the person. Scientists have been researching on improvement of this area for more than 20 years but reliable facial recognition systems were not used in real world applications until recently.

Moreover, face recognition requires huge amount of calculation. The process starts with detecting a face in an image, then reducing noise by cropping the picture to the face area, next extracting facial features, and finally compare the face to a set of provided face images called the training set. A computer with a lot of processing power is needed to do the whole calculation at a reasonable speed. With the recent advances in technology, embedded computers are getting smaller and more powerful; however, the question remains whether these small computers are powerful enough to be able to run the face recognition process in a real-time or near real-time fashion.

This thesis provides an evaluation of three open-source widely used facial feature extraction methods, eigenfaces, fisherfaces and local binary patterns histograms, on three embedded hardware platforms, Raspberry Pi 2, Intel Next Unit of Computing (NUC) and AMD G-series system on chip. Moreover, it takes six different cases in which variations in face images are investigated into consideration. In order to have thorough results, this study performs all experiments twice, one time with two people as known ,i.e. their pictures exist in the training set, and one as unknown and another time with three people as known and two as unknown.

Results are evaluated and discussed about for each case and each hardware platform separately, then an average of all experiments are reported as the overall performance for each method. Experiments show that eigenfaces had the highest performance among all, local binary patterns histograms had slightly lower accuracy of recognition and fisherfaces performed with the accuracy of 13,92 percent lower than eigenfaces.

# Contents

# 1. Introduction

## 1.1. Motivation

One of the most interesting tasks that the human brain is capable of doing is to recognize faces. Our brain distinguishes known faces almost momentarily despite of changes in facial features caused by

- **Expressions**
  As it can be seen in Figure 1.1 a familiar person is easily recognized regardless of the expression (s)he may have. In order to clarify, an emotion of happiness or sadness, for example, does not affect our ability [GÜR11].

- **Aging**
  Human brain can recognize a known face even without seeing it for several years. Changes may be considerable; however, the face is still recognizable.

- **Changes In the Appearance**
  Small changes in appearance like wearing a wig or eye-lenses, a different hairstyle or growing a beard can be easily identified.

- **Angles and Portions**
  A familiar face can be recognized from different angles even if a portion of it is visible [DC09].

- **Lighting and Background**
  Human brain is capable of recognizing faces under different lighting and backgrounds.

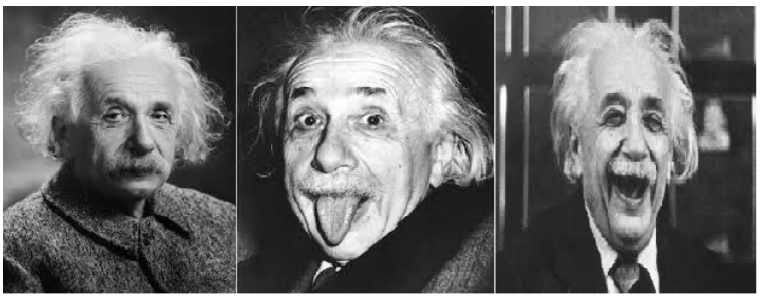


**Figure 1.1.:** A familiar face under different expressions

However, how this process is done in our brain is little known [Opea]. For several years, scientists attempted to give this ability to computers since a Face Recognition System (FRS) can be used in variety of applications, which are but not limited to

- **Security Systems**
  To give or prevent access of a known person. For this application a very high precision is required since a falsely recognized face can do harm to the system. Examples of usage can be in airport security or intelligent house where some people may not gain permission to pass the gateway.

- **Human Computer Interaction (HCI)**
  HCI is defined as "a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them" [Tri11]. Face recognition can be used in the HCI field for many applications, for example to unlock a smart-phone automatically instead of entering the pass-code.

- **Augmented Reality (AR)**
  "AR allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. Therefore, AR supplements reality, rather than completely replacing it" [Azu97]. A face recognition enabled augmented reality software can provide necessary information about a person, for example his or her preferences, in real time.

## 1.2. Industrial use

The Munich based start-up company *open ideas GmbH* provides software services to chilled delivery stations called emmasbox. Retailers, for example supermarkets, put their customers on-line bought products in these stations and customers pick them in a specific time frame. The range of software services offered for emmasbox starts from low-level embedded software, which deals with opening and closing boxes regarding the delivery of products by retailers and picking them by customers, to high-level computer vision using the integrated camera in the station.

**Figure 1.2.:** emmasbox station

The emmasbox system procedure is given in the following:

1. **On-line order**
   Costumers order on-line and select the emmasbox as delivery method.

2. **Off-line delivery**
   Retailers place the ordered product into the station.

3. **Notification**
   Costumers will be notified about the delivery and they will be given an access

code to retrieve their packages.

4. **Flexible collection**
   Costumers go to the station and collect their ordered packages [Emm].

Figure 1.3 shows the above procedure:

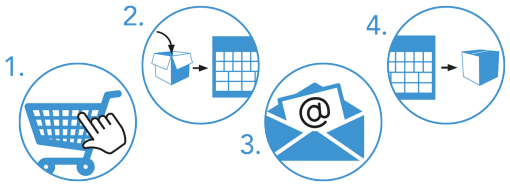



**Figure 1.3.:** procedure

An FRS can be chosen in the step three of the above procedure as the access code. In this case, when costumers go to the station to collect their packages, the integrated camera in the station will process the live stream of video taken from the costumer in front of the station and will automatically present their packages if the costumer is recognized.

## 1.3. Challenges

Research in the field of face recognition dates back to 1960s; however, only recently some face recognition systems have been used in real world applications. In order to use an FRS, very high degree of accuracy needs to be assured; otherwise, results may be disastrous. The problem in hand is hard to address not only from algorithmic approaches but also from the processing power point of view.

In order to clarify, a stream of high quality video taken by Full High Definition (Full HD) camera installed in an airport gives a resolution of 1080p which is 1080 horizontal lines of 1920 pixels each; therefore, each frame consists of 273600 pixels. A computer can represent a pixel with 24 bits which gives a total of 6566400 bits for each frame [Bal00].

Real-time face recognition, in this thesis, is defined as a system that can process captured frames from a camera continuously as they arrive without interrupting the stream of data. In case of an airport example provided above, the big amount of data

needs to be processed in a real-time manner which requires sophisticated algorithms and powerful hardwares.

The data processing performed for recognizing a person begins with detecting a face in a frame. Face detection is done by comparing all bits in a picture with a set of predefined information indicating whether or not a human face exits. Afterwards, the detected face will be compared with a database of known people to find a match. The whole process can be easily affected by many factors, for example the angle with which the person to be identified is looking at the camera or the lighting under which (s)he is being video-taped.

In a controlled test environment, the test subject may be looking directly at the camera; however, this does not happen in a real-world application in which face recognition is being done on-the-go. An example of such a system is in airport security when people entering the airport are being investigated to avoid a potential threat. In such a case, as it is shown in Figure 1.4, some parts of the face will not be shown in the picture and it will be harder to detect and recognize the face.

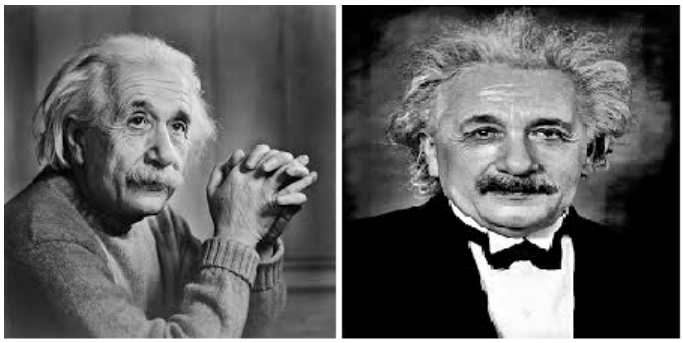


**Figure 1.4.:** Looking at the camera with an angle makes it harder to recognize the person

Furthermore, images from the same face under different lighting conditions can have higher variations than images from different people under the same lighting condition [YU94]. As a result, a person may be falsely recognized or might not be recognized at all when his/her picture stored in the database is under a different lighting direction.

## 1.4. Statement of the problem

As it was previously stated, given the complexity of the face recognition algorithms, a powerful hardware is required to perform face recognition algorithms on the receiving frames of a live camera feed in a real time manner. However, in the case of emmasbox, the powerful computer in the station neither can be fully allocated to the face recognition program since it will prevent the system to run the station software, nor can be shared between the two softwares since it will reduce the overall system performance.

As a result, a separate embedded hardware platform is desired to be only used for the face recognition software. This thesis attempts to figure out which of the most widely used facial feature extraction algorithms gives a better performance in terms of precision of the recognition under a real-world application. These algorithms include:

- **Eignefaces**
  Maps images into a face subspace using principle component analysis [TP92].

- **Fisherfaces**
  Uses the linear discriminant analysis to map to the lower subspace.

- **Local Binary Patterns Histograms**
  Segments the image into small grids and focuses on local features.

The theory behind these algorithms will be thoroughly investigated in the next chapter.

## 1.5. Outline of the thesis

The remaining of this thesis is organized as follows:

In chapter 2, the related research done in the field of face recognition will be presented in a historical way from the time face recognition was introduced. Moreover, this chapter presents the theoretical background needed to understand the thesis including how the face recognition is done from algorithmic approaches and discusses the mathematics behind the algorithms.

Chapter 3 describes the experimental setup of this thesis. The three investigated hardware platforms are introduced and software setup is illustrated including what

preprocessing steps are needed before face recognition can be preformed.

In chapter 4, experimental criteria will be defined and results will be presented reporting the performance of facial feature extraction methods for each of these criteria on all embedded platforms.

Finally, chapter 5 concludes the thesis by discussing the trend of data and giving the overall performances. In order to be complete, implementation of the algorithms are provided at the end of the thesis.

# 2. Literature survey and background

Face recognition is a two-step process done on an image or series of frames acquired from a video file or a camera feed. The first step is to detect a face in the picture and the second is to compare the detected face with a database of faces and identify the person. In this section, different methods available for each of these steps are investigated.

## 2.1. Face detection

There are many tools and techniques which are used in different methods of face detection. One is Principle Component Analysis (PCA) which is a mathematical approach that reveals simple structures amongst data in a complex dataset by reducing it to a lower dimension dataset [KL02]. Face is detected in [LY03] using the PCA to find eyes, and geometrical information to find mouth.

Linear Support Vector Machine (SVM) is used in [RY09] to detect a face region in a picture, then the location of the eyes and mouth are found according to the color differences within the region. In [WL02], a face region is detected according to the symmetry in the face, then SVM is applied to verify whether a face exists. In [CL06], SVM is combined with scanning techniques. Another approach was introduced by [HL04] which uses only pictures containing a human face to train the SVM as a one-class classifier since modelling images without a human face is not an easy task.

Template matching is a technique in which the whole image is scanned with a window size of 20x20 (or 30x30) pixels and compared to a template to find a face [GÜR11]. A template of half-face is used by [WW09] in order to add robustness against face orientations and reduce computational time. Abstract templates consists of parameters instead of a face image are used in [HJ10] in a two-step process. First step finds the region in which eyes reside and the second finds the exact location of each eye within this region.

Template matching is used in [WY08] to find face regions, then PCA is applied to the image to find features. In [CT06], PCA is used to find face regions, then faces are

given to neural networks to exclude the falsely detected faces. [DK99; Ana+02; XW10] segment the image according to the color of the human skin, then give the segmented image to the neural network to classify images with a face in them.

## 2.2. Face recognition

Face recognition dates back to 1960 when [LX06] produced the first facial recognition system. However, their machine was not fully automated and required a human to locate the facial features like eyes on pictures before they could be processed by the machine. Later, [AH06] used 21 features in a human face in order to automate the process of face recognition.

In 1973, the first machine who could perform face recognition in a fully automated fashion was introduced by [TJ05]. A set of 16 facial features were used by the proposed algorithm to identify a person and a precision rate of up to 75 percent was reported [Mar10]. In 1986, [SK87] used principle component analysis in face recognition. Their work was the foundation of many researches, and [TP92] introduced the eigenfaces algorithm in 1992 which became widely used in the following years resulting in the development of the first real-time face recognition system.

### 2.2.1. Eigenfaces

The following studies the theory and mathematics of the eigenfaces algorithms.

#### 2.2.1.1. Theory

[SK87] showed that a picture of a face can be approximately reconstructed by using principle component analysis meaning that a face image, which is a multidimensional data structure, can be mapped into a lower dimension space and then remapped into an image which is roughly the same as the original picture.

In other words, they only save the important features of an image. [TP92] stated that these features are not necessarily any facial features such as eyes, lips, eyebrows, nose or mouth but mathematically important features needed to reconstruct the image. They used this idea in face recognition in a way that they applied PCA to map a set of training face images into their feature space.

The face image required to be recognized is also mapped to its set of features, then it is compared to the training data in the lower dimension space and recognized as the most similar person. Moreover, they introduced a way to learn to recognize an unknown face automatically without having his/her pictures in the training set.

In their algorithm, a face is recognized with the following steps:

- Each of the training images is mapped into their features. These feature vectors then form the "face space".
- Mapping is done on the input image based on all of the mapped images in the training set, i.e., the face space.
- The feature set of the new image is compared to the face space. This verifies the existence of a face in the image, i.e., face detection.
- If face is detected, the recognition is performed based on the position of the mapped input image in the face space amongst all known images. If its closeness to a known person is higher than a threshold value, the algorithm will recognize him/her.
- If a face cannot be recognized but it has been seen several times, the mapped image will be added to the face space as a known person. This will allow the algorithm to expand its training set and learn to recognize new faces automatically.

#### 2.2.1.2. Mathematics

An image with the dimensions of $N \times N$ is a vector of length $N^2$; therefore, it can be represented as a point in a huge $N^2$-dimensional space [TP92]. Similar images, in this space, will be relatively close to each other and face images, due to having similar structures, will not be parsley distributed. Using PCA, a sub-space of face images can be obtained. PCA calculates the eigenvectors of the covariance matrix of all face images in the $N^2$-dimensional space, in other words, it defines a set of vectors showing the variation between face images.

The set of training images can be represented by Equation 2.1

$$X = \{x_1, \ldots, x_n\} \tag{2.1}$$

In order to compute the covariance, the mean of all images is required [Opea]

$$\mu = \sum_{i=1}^{n} x_i \tag{2.2}$$

Covariance is then calculated by the following formula

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \tag{2.3}$$

In which, $(x_i - \mu)$ is how much each image differs from the mean image. The eigenvectors and eigenvalues of $S$ can be found by solving Equation 2.4 [TP91]

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n \tag{2.4}$$

Then, the eigenvectors corresponding to the $k$ largest eigenvalues are used to project the training images into the lower dimensional space, as given by

$$y = W^T(x - \mu) \tag{2.5}$$

in which $W$ is the set of $k$ most significant eigenvectors

$$W = \{v_1, v_2, \ldots, v_k\} \tag{2.6}$$

When a new face image $I$ is given, first it will be projected into the subspace using Equation 2.7 [TP92]:

$$y_i = W^T(I - \mu) \tag{2.7}$$

Then, the recognition will be done using the nearest neighbor approach; in other words, $y_i$ is classified as a known person if its euclidean distance with the projected images of $X$ , i.e. people in the training set, is below a certain threshold.

### 2.2.2. Fisherfaces

The following studies the theory and mathematics of the fisherfaces algorithms.

#### 2.2.2.1. Theory

Principle component analysis, which is used by eigenfaces method, does not perform any class-specific dimensionality reduction. In order to clarify, all the images in the training set are mapped separately in the face space resulting in the maximum number of clusters, i.e, one cluster per each image.

[PK97] introduced fischerfaces which uses Linear Discriminant Analysis (LDA) to reduce the dimensions of train images and cluster them into separate classes, in the face space, in a way that mapped images in the same class are close to each other while different classes are far from one another. Figure 2.1 shows the result of PCA and LDA on a set of data [Wag].

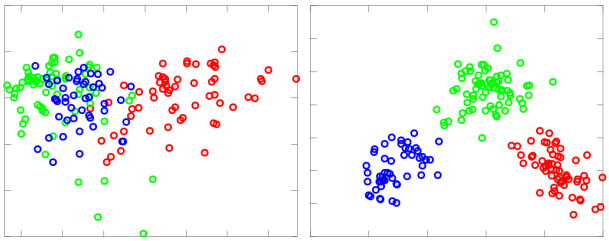


**Figure 2.1.:** PCA (left) and LDA (right) applied to the same dataset

Fisherfaces, as a result of using LDA, is more robust against variations in the data caused by an external source, e.g., light direction [PK97]. As it was previously stated and can be seen in Figure 2.2, light direction can make two pictures of the same person look more different than pictures of two different people [YU94].

**Figure 2.2.:** Effect of light direction on face images

#### 2.2.2.2. Mathematics

The set of training images can be represented by [Opea]

$$\begin{aligned} X = &\quad \{X_1, X_2, \ldots, X_C\} \\ X_i = &\quad \{x_1, x_2, \ldots, x_n\} \end{aligned} \tag{2.8}$$

Where $X$ is divided into $c$ classes, and each class $X_i$ contains a number of images.

As it was explained in the previous section, fisherfaces maps the train images into the subspace and clusters them into classes in a way that mapped images in the same class are close to each other while different classes are far from one another. In other words, fisherfaces finds the eigenvectors by maximizing the between-class to within-class scatter ratio [PK97].

The between-class scatter matrix $S_b$ and the within-class scatter matrix $S_w$ can be given as

$$S_B = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{2.9}$$

$$S_w = \sum_{i=1}^{c} \sum_{x_j \in X_i} (x_j - \mu)(x_j - \mu)^T \tag{2.10}$$

Where $N_i$ is the number of samples in each class. $M$ and $M_i$ are the mean of all images and the mean of images in each class, respectively

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.11}$$

$$\mu_i = \frac{1}{|X_i|}\sum_{x_j \in X_i} x_j \tag{2.12}$$

Fisherfaces finds an optimal projection that maximizes the ratio of $S_b$ to $S_w$

$$W_{opt} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|} \tag{2.13}$$

Then, *Wopt* can be given by solving the Equation 2.14

$$\begin{aligned} S_B v_i &= \lambda_i S_w v_i \\ S_w^{-1} S_B v_i &= \lambda_i v_i \end{aligned} \tag{2.14}$$

However, $S_w$ is a singular matrix because its rank is at most $(N - c)$ , where $N$ is the number of pictures and $c$ is the number of classes. As a fact, $N$ is almost always smaller than the number of pixels in an image; therefore, the within class scatter matrix becomes singular [PK97]. Fisherfaces, consequently, reduces the dimensionality of the image space in a two step-process

- PCA is used to project the training images into a sub-space with the dimension of $(N - c)$
- LDA is applied to the mapped images in the PCA sub-space to reduce the dimensionality of the input data to $(c - 1)$

The above procedure can be given as [Opea]

$$W_{opt}^T = W_{fld}^T W_{pca}^T \tag{2.15}$$

where

$$W_{pca}^{T} = \arg\max_{W} |W^{T} S_{T} W| \tag{2.16}$$

$$W_{fld}^{T} = \arg\max_{W} \frac{|W^{T} W_{pca}^{T} S_{B} W_{pca} W|}{|W^{T} W_{pca}^{T} S_{W} W_{pca} W|} \tag{2.17}$$

### 2.2.3. Local binary patterns histograms

The following studies the theory and mathematics of LBPH algorithm.

#### 2.2.3.1. Theory

Fisherfaces and eigenfaces use a top-down approach in face recognition, i.e. they map the whole image into a lower space and work on the mapped image. LBPH, however, encodes each pixel of the image and extracts features based on the encoded data [Opea].

The original Local Binary Patterns (LBP) approach moves a window of $3 \times 3$ pixels across the whole image and applies a threshold on each pixel of the window against the window's central pixel [TH13]. As it can be seen in Figure 2.3, if the current pixel has a greater or equal value than the central pixel, LBP gives the binary value of 1; otherwise, 0. Then, a binary value will be assigned by concatenating the 8 thresholded values [TP06].

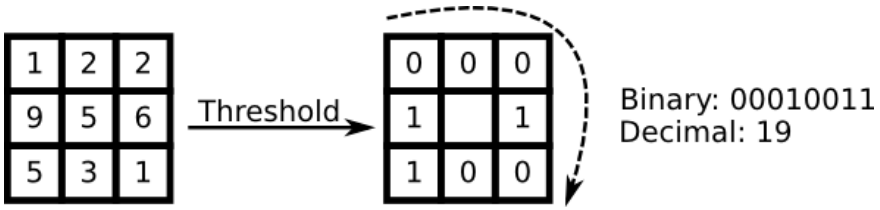



**Figure 2.3.:** The original LBP applied on a set of data

A histogram of these encoded pixel values, as shown in Figure 2.4, gives information about local micropatterns, e.g. edges and flat regions [RT96].

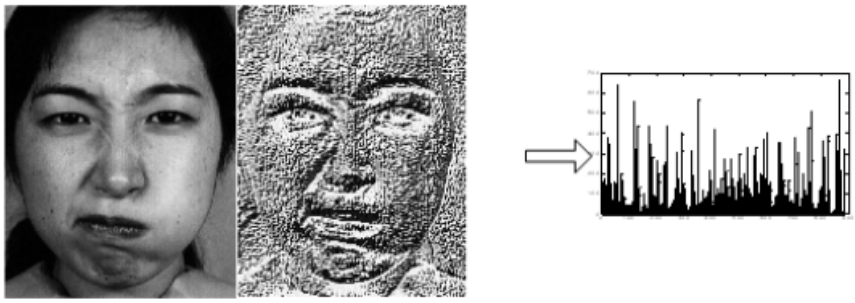


**Figure 2.4.:** Histogram of an LBP operator

A better representation of the facial image can be obtained by taking the spatial information of the picture into account. This is achieved with the following steps [TP06]

- Segmenting the image into small regions
- Encoding each pixel in the region by applying the LBP method
- The feature set of the new image is compared to the face space.
- Obtaining a histogram for each region
- Concatenating all of the histograms

Moreover, weights can be given to each region according to the importance of information they contain for face recognition, e.g. eyes are highly important. These weights will then be used to classify a new face with a nearest neighbor approach explained in the following section, which illustrates the mathematics behind the algorithm.

#### 2.2.3.2. Mathematics

The LBP operator can be represented as [Opea]

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c) \tag{2.18}$$

In which $P$ is the size of window, that is 9 in case of original LBPH since it uses a 3x3 window, $I_p$ is the density of the neighbor pixel, $I_c$ is the density of the central pixel located at $(x_c, y_c)$ and $s$ is the sign function given as

$$f(n) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

An extension to the LBP operator, called extended LBP (or circular LBP), was introduced by [TM02] in which the window is not of fixed size. Neighborhoods are defined with two extra variables, $P$ and $R$, which are the number of sampling points and the radius of the circle around the central pixel, respectively.

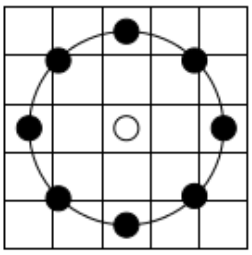


**Figure 2.5.:** Extended LBP with P=8 and R=2 pixels

Figure 2.5 illustrates the extended LBP [TP06]. As it can be seen, some points on the circular LBP may not correspond to an actual pixel in the image. In this case, a bilinear interpolation is used according to Equation 2.20 [Opea]

$$\mathrm{f}(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} \mathrm{f}(0,0) & \mathrm{f}(0,1) \\ \mathrm{f}(1,0) & \mathrm{f}(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \tag{2.20}$$

After the image is labeled with the extended LBP operator, a histogram of the encoded values can be given by

$$H_i = \sum_{x,y} I\{\mathrm{f}_l(x,y) = i\} \quad , \quad i = 0, \ldots, n-1 \tag{2.21}$$

In which $\mathrm{f}_l(x,y)$ is the labeled image, $n$ is the number of labels that the extended LBP

returns and $I$ is calculated according to Equation 2.22

$$I(A) = \begin{cases} 1 & \text{A is true} \\ 0 & \text{A is false} \end{cases} \tag{2.22}$$

The histogram can be further enhanced when taking the spatial information of a face image into account. In order to do so, the image is divided into $m$ regions and the enhanced histogram is given by [TP06]

$$H_{i,j} = \sum_{x,y} I\{\mathrm{f}_l(x,y) = i\}\ I\{(x,y) \in R_j\} \quad , \quad i = 0,\ldots,n-1 \quad j = 0,\ldots,m-1 \tag{2.23}$$

# 3. Experimental setup

This section presents the hardware and software setup to run the experiments.

## 3.1. Embedded Hardware Platforms

The primary goal of this thesis is to evaluate the algorithms on a Raspberry Pi 2 hardware platform; however for the sake of completeness, the implemented software is also run on an Intel Next Unit of Computing (NUC) and Advance Micro Devices (AMD) G-series System on Chip (SoC) which will be introduced in the following.

### 3.1.1. Raspberry Pi 2

Raspberry Pi (RPi) is a low cost, credit-card sized embedded computer capable of performing any task a normal computer would do. From browsing the Internet, to word-processing, playing videos, playing games and programming. RPi is produced to serve as an education assist to learn computing and programming for anyone regardless of their age and technical background [Foub]. To that end, RPi comes with a broad range of programming tools to make computing easy and understandable.

#### 3.1.1.1. History

At 2006, a group of scientists from the University of Cambridge considered a small-sized, cheap computer which could be used by beginner students to learn basic computer science concepts. Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft were concerned that students' interest in computer programming has specifically declined [Foua].

They believed there are number of reasons why this occurred, most importantly however, was because an affordable computer with which students can gain computer skills did not exist. Computers where so expensive that people were not eager to use

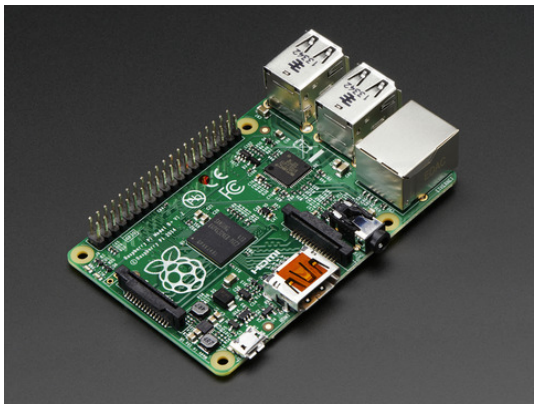


**Figure 3.1.:** Raspberry Pi

them for experimentations which could result in maintenance. The first prototypes of an affordable, credit-card sized computer were designed between 2006 and 2008.

By the year 2008, as a result of grown interest and competition in the cell-phone industry, processors became much more powerful and affordable. These scientists designed their embedded hardware platform which was finally introduced at 2012. Since then, Raspberry Pi has been used not only for educational purposes, but also for commercial applications. On February 2015, Raspberry Pi 2 was introduced which is the main hardware platform of interest for this thesis.

#### 3.1.1.2. Raspberry Pi 2 hardware specification

Raspberry Pi incorporates a Broadcam BCM2836 System on Chip (SoC) which includes a quad-core ARM Cortex-A7 complex and a VideoCore IV GPU. The ARM processor has a frequency of 900 MHz which can be over-clocked to 1 GHz. In the first module, Raspberry Pi came with a 256 MB which was upgraded to 1 GB in the newest version. The board can be connected to a monitor via an HDMI port and to a mouse and keyboard via available USB ports.

RPi is powered with a Micro USB port and connected to Internet via the Ethernet port or a USB wireless adapter. A Secure Data (SD) card must be plugged in to the RPi which acts as the main memory storage for the board containing the OS and user data. The audio output and the General Purpose Input Output (GPIO) pins serve as another means for the board to communicate with the outside world.

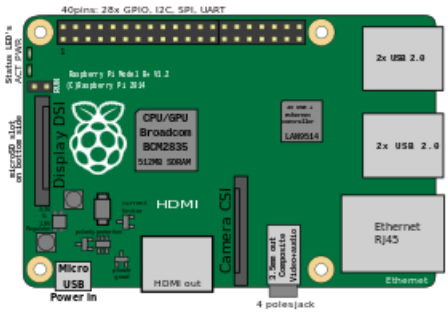



**Figure 3.2.:** Raspberry Pi hardware specification

### 3.1.2. Intel NUC

The face recognition software is also tested on an Intel NUC kit D54250WYKH which includes a core i5 4250U processor with the speed of 1.3 GHz with turbo capability which increases the frequency of the processor up to 2.6 GHz. The processor has 2 cores each of which including 2 threads and 3 MB of Intel smart cache.

**Figure 3.3.:** Intel NUC

### 3.1.3. AMD G-series SoC

As it was previously stated, for the stake of being complete, this thesis also cross validates the results on a Single Board Computer (SBC) with AMD Embedded G-Series

GX-420CA System on Chip (SoC) which incorporates a quad core processor operating at a 2.0 Ghz frequency.

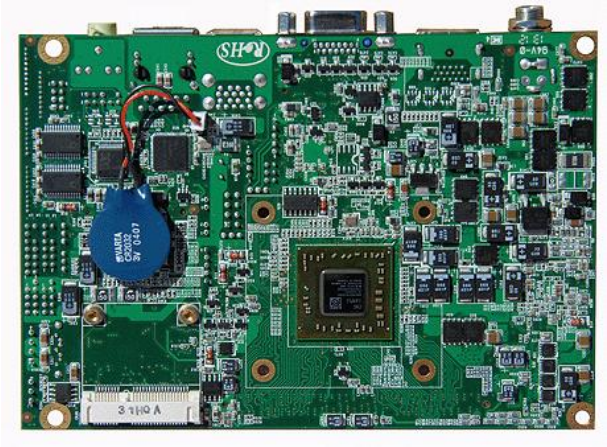


**Figure 3.4.:** AMD based embedded board

This board also includes up to 4 GB of DDR3 ram memory, 120 GB of SSD hard drive, two USB 3.0 and four USB 2.0 ports and a Gb ethernet. Figure 3.5 shows the hardware specification of the board [Sem].

| | |
|---|---|
| **System** | |
| **Form Factor** | 3.5" Single Board Computer |
| **Processor** | Onboard AMD® Embedded G-Series SoC |
| **Chipset** | Integrated |
| **Memory** | 1 x DDR 3 1600MHZ / So-DIMM up to 4GB |
| **BIOS** | AMI SPI BIOS |
| **Watchdog Timer** | 1 ~ 255 Sec. |
| **Ethernet** | 1 x Intel® i211 AT Gigabit Ethernet |
| **SSD** | 1 x Half-size Mini-PCIe support mSATA |
| **Storage** | 1 x SATA 6.0 Gb/s |
| **Digital I/O** | 8-bit programable |
| **LPC** | 1 x LPC support TPM module |
| **Expansion Interface** | 1 x Full-Size Mini-PCIe (w/USB, PCIe x 1 Single)<br>1 x Half-Size Mini-PCIe (w/USB, PCIe and SATA Single) |
| **Power Requirements** | DC 8V ~32V |
| **Board Size** | 146mm x 101mm |
| **Operating Temperature** | 0°C~60°C (32°F~140°F) |
| **Stroage Temperature** | -20°C~80°C (-4°F~176°F) |
| **Relative Humidity** | 10%~90% (non-condensing) |
| **Display** | |
| **Chipset** | Integrated |
| **Display Interface** | 1 x VGA<br>1 x dual channel 24-bit LVDS<br>1 x HDMI with Optional Chrontel CEC Support |
| **I/O** | |
| **Serial Port** | 1 x RS232/422/485 & 3 x RS232 |
| **USB** | 2 x USB 3.0 + 4 x USB 2.0 |
| **Audio** | HD Audio |
| **LPC** | 1 x LPC header for optional TPM module |

**Figure 3.5.:** AMD board hardware specification

### 3.1.4. Camera

For this experiment an Apple iSight camera is used which has a focus of 8-megapixel and takes 1080p High Definition (HD) videos. In order to be able to compare the performance of all three hardware platforms, all training videos have been taken with the same camera and given to the algorithms to extract training images and save the results. Then these results have been fed to the software on the embedded hardwares to have the exact same set of input across all three platforms.

## 3.2. Software setup

This section talks about software side setup for the thesis including the operating system, computer vision library and the operation done in the face recognition program to prepare for the experiments.

### 3.2.1. Operating system

The official widely supported operating system for RPi is raspbian which is a free linux distribution based on debian. There are a few other operating systems which serve different purposes for example raspbmc, a linux distribution with extensive media playback capabilities. However, with the introduction of Raspberry Pi 2, it is possible to run ubuntu on the board.

In order to have a fair comparison between three embedded hardware platforms, the latest stable ubuntu distribution, 14.04, was used. At the time of the thesis a later version, 15.04, was also available; however, the community support for 14.04 was a lot better and this version was massively used and tested.

### 3.2.2. Computer vision library

For this thesis, an Open source Computer Vision (OpenCV) is used to program the face recognition software. OpenCV is a programming library that includes the implementation of many computer vision algorithms with focus on real-time applications. One of these applications is face recognition for which implementation of the investigated algorithms is provided.

The latest stable version of OpenCV, 2.4.11, was used. At the time the experiments were being conducted, a newer version of OpenCV, 3.0.0, was also available but the stable version was chosen for the same reasons as those for choosing the version of operating systems.

## 3.3. Face recognition preparation

As it was previously stated, there are a lot of steps before the face recognition algorithms can come into place. When an image is acquired, first face needs to be detected, then the picture will be cropped to the face area to lose the irrelevant information, afterwards a lot of preprocessing will be done on the face picture to reduce the computation load and make it feasible for the system to be real-time. Figure 3.6 shows a picture used in the training set of this experiment. The outcome of these preprocessing steps on this image will be presented in the following.

**Figure 3.6.:** An original training image

### 3.3.1. Face detection

OpenCV comes with machine learning based classifiers which can be trained from a lot of positive and negative pictures. These classifiers can be used to detect many different objects of choice; however, some pre-trained classifiers are already available to detect face, eyes and different facial features. Figure 3.7 shows the result of face detection on the test image.

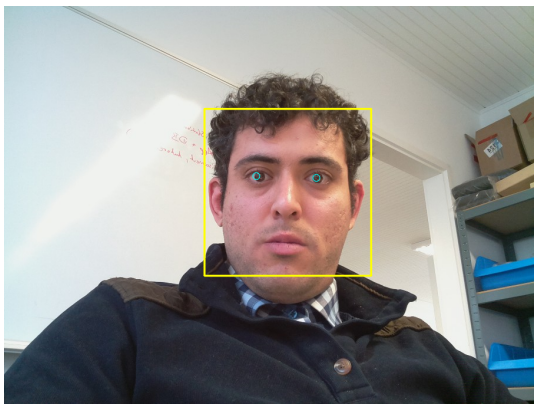


**Figure 3.7.:** Face and eyes detection

### 3.3.2. Preprocessing

After a face is detected, the picture will be cropped to face area in order to only keep the relevant information and reduce the computation complexity. Figure 3.8 shows this process.

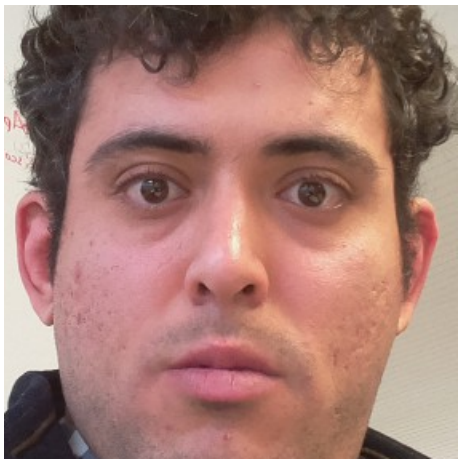


**Figure 3.8.:** The cropped image

Then, the cropped picture will be converted to gray style which is needed for the investigated algorithms. This is illustrated in Figure 3.9.

At this point, the image will be rotated to have the face in the middle. Another reason for that is for all pictures in the database to have almost the same face rotation regardless of the angle the subject is facing the camera. This will increase the precision

**Figure 3.9.:** Gray style image

accuracy of the face recognition when the subject to be recognized is looking at the camera from different angles. Moreover at this step, the image will also be resized to a smaller picture to reduce the computation complexity; however in order for the images to be viewable, pictures will be shown in their original size.

#### 3.3.2.1. Histogram equalization

The next step is histogram equalization which improves the contrast of an image. This will greatly affect the picture in case it is too light or dark. A histogram gives the pixel value distributions of an image [Opeb]. Figure 3.10 shows the histogram of the colored face image.

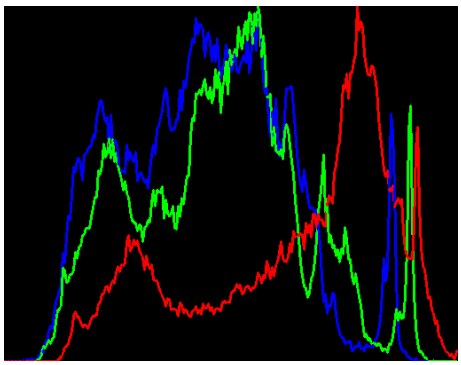


**Figure 3.10.:** Histogram of the colored image

As it can be seen, there are three histograms, one for each channel, which show the density of the pixels in the entire image. Figure 3.11 shows the histogram of the converted gray picture which has only one channel.

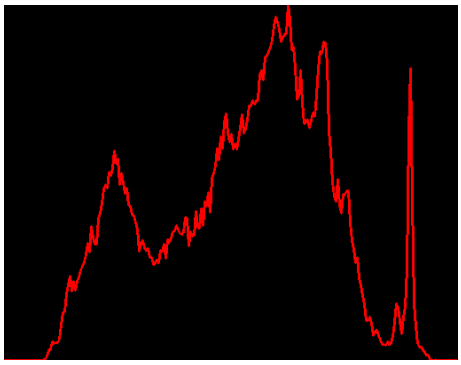


**Figure 3.11.:** Histogram of the gray image

Histogram equalization stretches the histogram ranges to enhance the contrast of the given image. Figure 3.12 presents the histogram of the enhanced photo.

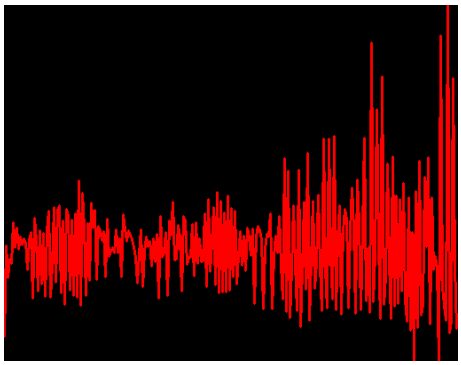


**Figure 3.12.:** Histogram of the enhanced image

Figure 3.13 shows the result of histogram equalization on the test image.

#### 3.3.2.2. Bilateral filtering

One of the most frequently used image processing techniques is filtering which smooths or blurs the image in order to reduce noise [Opec]. OpenCV comes with a range of filters, e.g. Gaussian, median and normalized box filters; however, bilateral is the only one that keeps the edges sharp while smoothing the picture. In the next step, a bilateral

**Figure 3.13.:** Histogram equalized image

filter will be used on the picture. Figure 3.14 illustrates the effect of bilateral filtering.

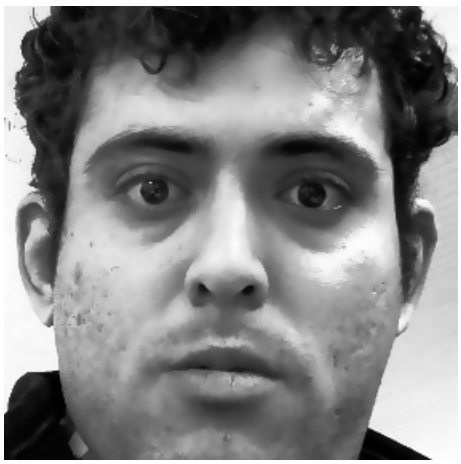


**Figure 3.14.:** Bilateral filtered image

Then, a mask will be created in the shape of an oval which clears the corners of the picture that do not belong to the face and are irrelevant information. Figure 3.15 shows the mask.

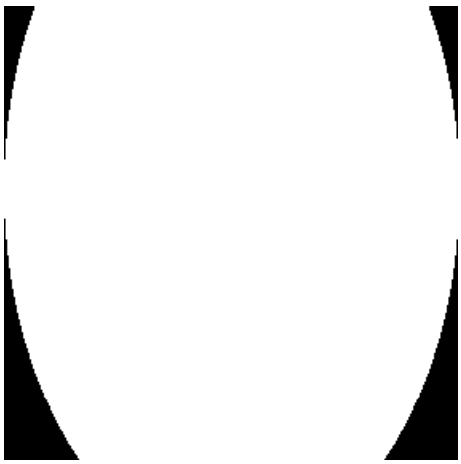


**Figure 3.15.:** Mask to clear the corners of the image

Finally, the masked image will be passed to the face recognition algorithms as shown in Figure 3.16.

**Figure 3.16.:** The masked image

The whole process will be done on each captured frame from the camera as well as on every train photos provided to the system. In order for the system to be real-time, all this computation must be done before the next frame arrives; as a result, a powerful processor must be in hand to execute these steps as fast as possible. However, using

the limited resources of an embedded hardware such as Raspberry Pi, a near real time system can also be acceptable.

# 4. Experimental results

This section reports the results obtained from the experiments.

## 4.1. Criteria

In order to correctly evaluate the performance of the algorithms, many different conditions must be examined. The facial recognition softwares may work outstandingly well when the test subject is looking directly at the camera and there is no lighting directions, head rotation or expressions; however, results may noticeably deteriorate with the existence of these conditions.

The other factor that affects the results is the number of pictures per person in the train set. When more pictures are provided for the face recognition software that contain all of the above mentioned conditions, the performance will considerably improve in comparison to when only small set of pictures are given.

Experimental criteria for this thesis are defined as the following. A picture showing each of these conditions are provided in the next sections where results for each criterion is reported.

- **Straight under normal lighting conditions**
  Subject looks straight at the camera under normal lighting condition.

- **Head rotation under normal lighting conditions**
  Performance of the algorithms will be evaluated when subject rotates his head.

- **Expressions under normal lighting conditions**
  Expressions like happy, sad and shocked are investigated whether they change the results.

- **Straight under heavy lighting conditions**
  Lighting conditions heavily impact the subject's face to the extreme that one half of the face is darker than the other. In this case, subject is looking directly at the camera.

- **Head rotation under Heavy lighting conditions**
  Head rotation are examined with light directions influencing the picture.

- **Expressions under heavy lighting conditions**
  Facial features extraction methods are evaluated when subject has expressions and is under lighting conditions.

- **Number of Images per Person (IpP)**
  Experiments are done using 20, 40 and 80 pictures per person.

  - 20 only includes straight and head rotation under normal lighting conditions in the training set.
  - 40 adds pictures of straight and head rotation under heavy lighting conditions to the set of images to be trained.
  - 80 takes all possible combination including expressions into consideration.

## 4.2. Definitions

In order to have the exact same set of input for all experiments, video files have been created each of which contains one of the investigated criteria. Face recognition will be done on every frame of the input video and the performance will improve if the test subject is correctly recognized or rightly classified as unknown. All of the following definitions are based on the number of frames that the algorithm works as expected out of the whole frames in the input video and are reported in percentages.

- **True Positive (TP)**
  TP indicates that the test subject is correctly recognized and classified as a known

person in the training set.

- **True Negative (TN)**
  TN specifies that the test subject is correctly classified as an unknown person.

- **False Positive (FP)**
  FP states that the subject who should be identified as unknown is falsely recognized as a known person in the training set.

- **False Negative (FN)**
  FN shows that the subject who should be identified as a known person in the training set is wrongly recognized as a unknown person or is identified as another test subject in the training set.

- **Accuracy**
  Accuracy is defined as the average of frames that the test subject is correctly recognized (TP) or rightly classified as unknown (TN) out of all frames and is reported in percentage. Accuracy can be given according to Equation 4.1.

$$Accuracy = \frac{TP + TN}{2} \times 100 \tag{4.1}$$

- **Error**
  Error rate is defined as the average of frames that the test subject who should be identified as unknown is falsely recognized as a known person (FP) or a test subject who should be classified as known is wrongly classified as unknown or another test subject (FN) out of all frames. Error is also reported in percentage and can be given according to Equation 4.2.

$$Error = \frac{FP + FN}{2} \times 100 \tag{4.2}$$

## 4.3. Threshold

As the theory of the algorithms were explained in the previous chapters, after the images in the training set and the image to be recognized are calculated on, a Nearest

Neighbor (NN) approach returns the most similar person as the result of recognition. Threshold will be applied to the similarity value reported by the NN algorithm which means the results are accepted only if the similarity is below a certain threshold and if above this value, the subject will be classified as unknown.

The threshold value affects the rate of TP and TN. Because NN will always return someone as the closest match, a high threshold will accept more unknown people as known; therefore, FP rate increases and TN rate decreases; however, the advantage of having a high threshold is that less known people will be filtered; as a result, the rate of TP increases.

On the other hand, a low threshold has the exact opposite effect. It will increase the rate of TN because more unknown people will be correctly recognized as unknown; however, its disadvantage is that the rate of TP decreases since more known people will be filtered out.

For these experiments, the value of threshold are chosen as low as possible since in a real-world application the penalty for a low rate of TN is much higher than a low rate of TP. In order to clarify, When a real-world face recognition system has a low TN rate (high FP rate) it means the system will accept more unknown people as known and it can lead to security leaks.

However, the penalty for a low TP rate is only poor performance which is not desirable; however, the software can ask costumers to use another log-in mechanism and there will be less security risks. It needs to be stated that for each criterion and each algorithm, the threshold values are adjusted separately then the results are compared which will be reported in the next sections.

## 4.4. Results

In this section, partial results will be presented. Partial in the sense that all defined values above will be reported for each criterion separately. It needs to be noted that all experiments have been done twice, one time with 2 people as known and 1 as unknown and another time with 3 people as known and 2 as unknown. results here are the average of both experiments. Moreover, the investigated hardware platforms, which were introduced in the previous chapter, are abbreviated as "rpi", "nuc" and "amd".

### 4.4.1. Case one: straight under normal lighting conditions

In this criterion, subject looks directly at the camera and the lightings are in normal condition which means the light shines from the above covering the whole face. Figure 4.1 shows an example image of this condition.

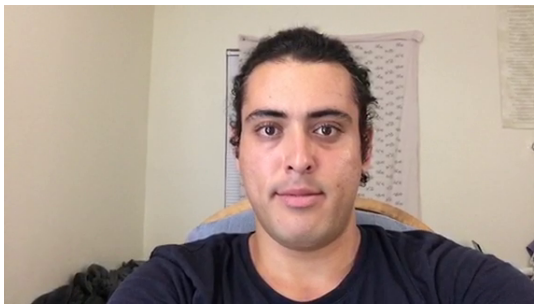


**Figure 4.1.:** Example picture illustrating case one

Results are first separately reported for each of the algorithms. Table 4.1 presents the results for eigenfaces.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 94.51 | 88.08 | 11.92 | 5.49 | 91.3 | 8.7 |
| | nuc | 95.37 | 87.15 | 12.85 | 4.63 | 91.26 | 8.74 |
| | amd | 95.37 | 87.15 | 12.85 | 4.63 | 91.26 | 8.74 |
| 40 | rpi | 98.81 | 85.82 | 14.18 | 1.19 | 92.32 | 7.68 |
| | nuc | 98.79 | 85.76 | 14.24 | 1.21 | 92.28 | 7.72 |
| | amd | 98.79 | 85.76 | 14.24 | 1.21 | 92.28 | 7.72 |
| 80 | rpi | 95.7 | 87.04 | 12.96 | 4.3 | 91.37 | 8.63 |
| | nuc | 96.2 | 86.46 | 13.54 | 3.8 | 91.33 | 8.67 |
| | amd | 96.2 | 86.46 | 13.54 | 3.8 | 91.33 | 8.67 |

**Table 4.1.:** Case one: eigenfaces results

Results for fisherfaces and LBPH are given in the next page in Table 4.2 and Table 4.3, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 92.09 | 100 | 0 | 7.91 | 96.04 | 3.96 |
| | nuc | 91.23 | 100 | 0 | 8.77 | 95.62 | 4.38 |
| | amd | 91.23 | 100 | 0 | 8.77 | 95.62 | 4.38 |
| 40 | rpi | 51.56 | 100 | 0 | 48.44 | 75.78 | 24.22 |
| | nuc | 54.32 | 100 | 0 | 45.68 | 77.16 | 22.84 |
| | amd | 54.32 | 100 | 0 | 45.68 | 77.16 | 22.84 |
| 80 | rpi | 94.85 | 100 | 0 | 5.15 | 97.42 | 2.58 |
| | nuc | 94.85 | 100 | 0 | 5.15 | 97.42 | 2.58 |
| | amd | 94.85 | 100 | 0 | 5.15 | 97.42 | 2.58 |

**Table 4.2.:** Case one: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 97.21 | 56.87 | 43.13 | 2.79 | 77.04 | 22.96 |
| | nuc | 96.44 | 75.4 | 24.6 | 3.56 | 85.92 | 14.08 |
| | amd | 97.79 | 55.83 | 44.17 | 2.21 | 76.81 | 23.19 |
| 40 | rpi | 99.42 | 82.39 | 17.61 | 0.58 | 90.91 | 9.09 |
| | nuc | 96.41 | 85.47 | 14.53 | 3.59 | 90.94 | 9.06 |
| | amd | 99.48 | 82.74 | 17.26 | 0.52 | 91.11 | 8.89 |
| 80 | rpi | 97.62 | 46.1 | 53.9 | 2.38 | 71.86 | 28.14 |
| | nuc | 96 | 72.27 | 27.73 | 4 | 84.14 | 15.86 |
| | amd | 97.58 | 45.93 | 54.07 | 2.42 | 71.75 | 28.25 |

**Table 4.3.:** Case one: LBPH results

Average of results for all hardware platforms are presented in Table 4.4 which shows the partial results for this case. Moreover, Figure 4.2 compares the accuracy of the algorithms according to the number of images per person. As it can be seen, fisherfaces has the highest performance with IpP of 20 and 80. Best results for this case were obtained when 80 pictures for each person were given to the fisherfaces algorithm to train on.

There is a drop of accuracy when changing from 20 to 40 images per person because 40 includes straight and head rotation under heavy lighting condition into the training set which are not examined in this case. However when 80 is used, which includes expressions under normal and lighting conditions, results improve considerably. When subjects have expressions, they look directly at the camera; as a result, performance improves for this case.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 95.08 | 87.46 | 12.54 | 4.92 | 91.27 | 8.73 |
| | Fisherfaces | 91.52 | 100 | 0 | 8.48 | 95.76 | 4.24 |
| | LBPH | 97.14 | 62.7 | 37.3 | 2.86 | 79.92 | 20.08 |
| 40 | Eigenfaces | 98.8 | 85.78 | 14.22 | 1.2 | 92.29 | 7.71 |
| | Fisherfaces | 53.4 | 100 | 0 | 46.6 | 76.7 | 23.3 |
| | LBPH | 98.44 | 83.53 | 16.47 | 1.56 | 90.98 | 9.02 |
| 80 | Eigenfaces | 96.03 | 86.65 | 13.35 | 3.97 | 91.34 | 8.66 |
| | Fisherfaces | 94.85 | 100 | 0 | 5.15 | 97.42 | 2.58 |
| | LBPH | 97.07 | 54.77 | 45.23 | 2.93 | 75.92 | 24.08 |

**Table 4.4.:** Case one: average results

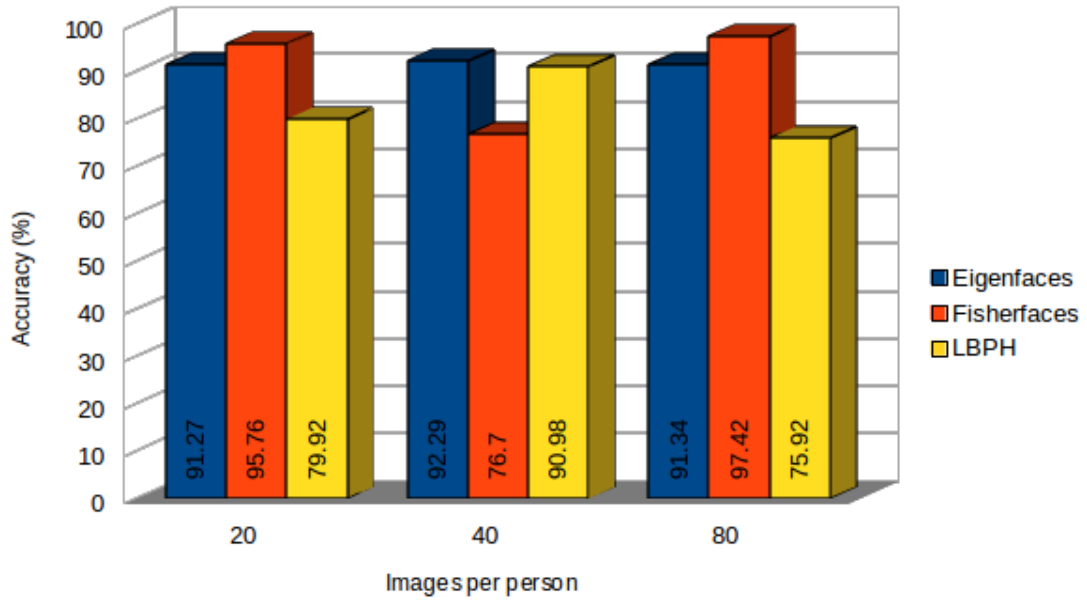



**Figure 4.2.:** Comparison of the accuracy of algorithms for case one

### 4.4.2. Case two: head rotation under normal lighting conditions

In this criterion, subject rotates his head to the sides, up and down under the normal lighting conditions. Figure 4.3 shows example images of this condition.

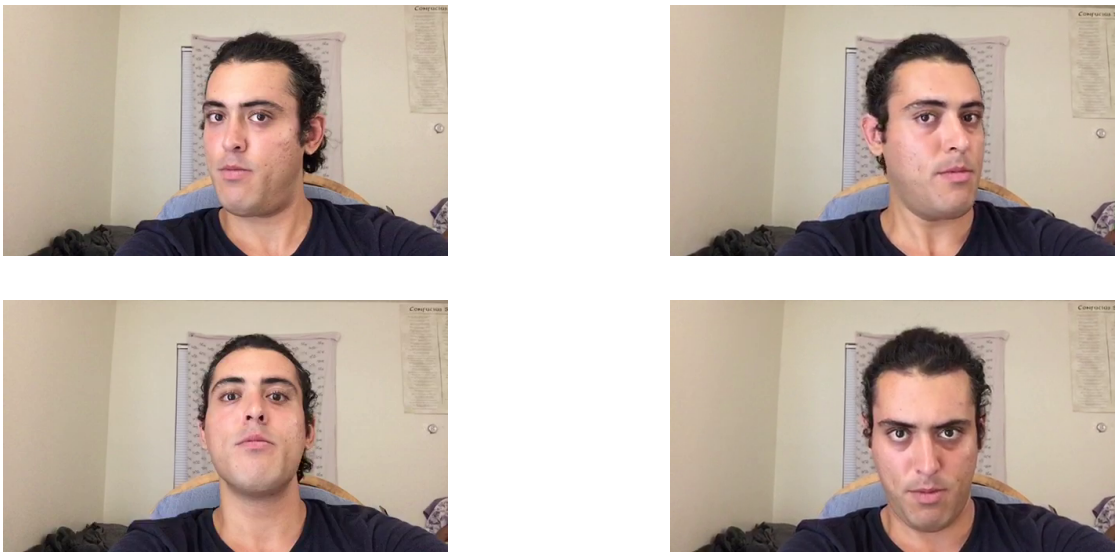


**Figure 4.3.:** Example pictures illustrating case two

Table 4.5 presents the results for eigenfaces.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 82.31 | 94.55 | 5.45 | 17.69 | 88.43 | 11.57 |
| | nuc | 81.95 | 95 | 5 | 18.05 | 88.47 | 11.53 |
| | amd | 81.95 | 95 | 5 | 18.05 | 88.47 | 11.53 |
| 40 | rpi | 91.36 | 81.02 | 18.98 | 8.65 | 86.19 | 13.81 |
| | nuc | 91.46 | 81.36 | 18.64 | 8.54 | 86.41 | 13.59 |
| | amd | 91.46 | 81.36 | 18.64 | 8.54 | 86.41 | 13.59 |
| 80 | rpi | 85.18 | 87.73 | 12.27 | 14.82 | 86.45 | 13.55 |
| | nuc | 85.29 | 87.73 | 12.27 | 14.71 | 86.51 | 13.49 |
| | amd | 85.29 | 87.73 | 12.27 | 14.71 | 86.51 | 13.49 |

**Table 4.5.:** Case two: eigenfaces results

Results for fisherfaces and LBPH are given in Table 4.6 and Table 4.7, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 41.4 | 96.36 | 3.64 | 58.6 | 68.88 | 31.12 |
| | nuc | 40.51 | 95.91 | 4.09 | 59.49 | 68.21 | 31.79 |
| | amd | 40.51 | 95.91 | 4.09 | 59.49 | 68.21 | 31.79 |
| 40 | rpi | 46 | 89.55 | 10.45 | 54 | 67.77 | 32.23 |
| | nuc | 44.67 | 88.18 | 11.82 | 55.33 | 66.43 | 33.57 |
| | amd | 44.67 | 88.18 | 11.82 | 55.33 | 66.43 | 33.57 |
| 80 | rpi | 48.27 | 76.36 | 23.64 | 51.73 | 62.32 | 37.68 |
| | nuc | 48.27 | 76.36 | 23.64 | 51.73 | 62.32 | 37.68 |
| | amd | 48.27 | 76.36 | 23.64 | 51.73 | 62.32 | 37.68 |

**Table 4.6.:** Case two: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 94.84 | 67.92 | 32.08 | 5.16 | 81.38 | 18.62 |
| | nuc | 87.86 | 81.48 | 18.52 | 12.14 | 84.67 | 15.33 |
| | amd | 94.4 | 68.93 | 31.07 | 5.6 | 81.67 | 18.34 |
| 40 | rpi | 97.15 | 79.17 | 20.83 | 2.85 | 88.16 | 11.84 |
| | nuc | 86.98 | 85.11 | 14.89 | 13.02 | 86.05 | 13.95 |
| | amd | 97.29 | 79.5 | 20.5 | 2.71 | 88.4 | 11.6 |
| 80 | rpi | 94.62 | 76.55 | 23.45 | 5.38 | 85.59 | 14.41 |
| | nuc | 83.82 | 85.34 | 14.66 | 16.18 | 84.58 | 15.42 |
| | amd | 94.61 | 76.43 | 23.57 | 5.39 | 85.52 | 14.48 |

**Table 4.7.:** Case two: LBPH results

As per last case, average results are shown in Table 4.8 and Figure 4.4 gives the comparison of the accuracy of algorithms. As it can be seen, eigenfaces has the highest performance, LBPH comes in the second place and fisherfaces the last. LBPH with 40 images per person performed even better than eigenfaces with the same number of IpP but not as good as eigenfaces in case there is 20 pictures per each person in the training set.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 82.07 | 94.85 | 5.15 | 17.93 | 88.46 | 11.54 |
| | Fisherfaces | 40.81 | 96.06 | 3.94 | 59.19 | 68.43 | 31.57 |
| | LBPH | 92.37 | 72.78 | 27.23 | 7.63 | 82.57 | 17.43 |
| 40 | Eigenfaces | 91.42 | 81.25 | 18.75 | 8.58 | 86.34 | 13.66 |
| | Fisherfaces | 45.11 | 88.64 | 11.36 | 54.89 | 66.88 | 33.12 |
| | LBPH | 93.81 | 81.26 | 18.74 | 6.19 | 87.53 | 12.47 |
| 80 | Eigenfaces | 85.25 | 87.73 | 12.27 | 14.75 | 86.49 | 13.51 |
| | Fisherfaces | 48.27 | 76.36 | 23.64 | 51.73 | 62.32 | 37.68 |
| | LBPH | 91.02 | 79.44 | 20.56 | 8.98 | 85.23 | 14.77 |

**Table 4.8.:** Case two: average results

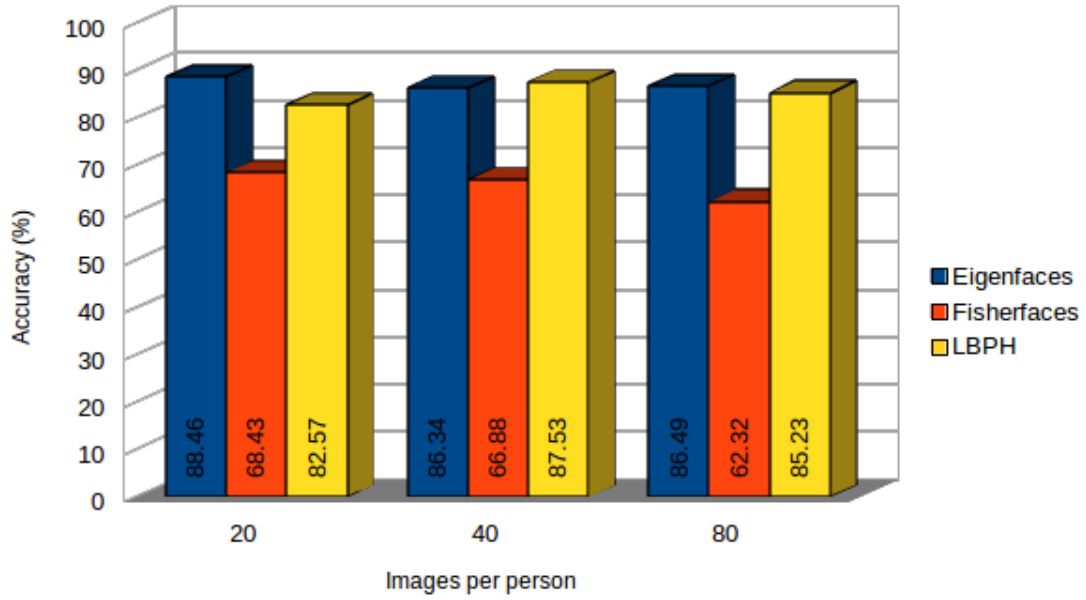



**Figure 4.4.:** Comparison of the accuracy of algorithms for case two

### 4.4.3. Case three: expressions under normal lighting conditions

The next case is when the subject has expressions and the light is in normal condition. Figure 4.5 shows example images of this condition.

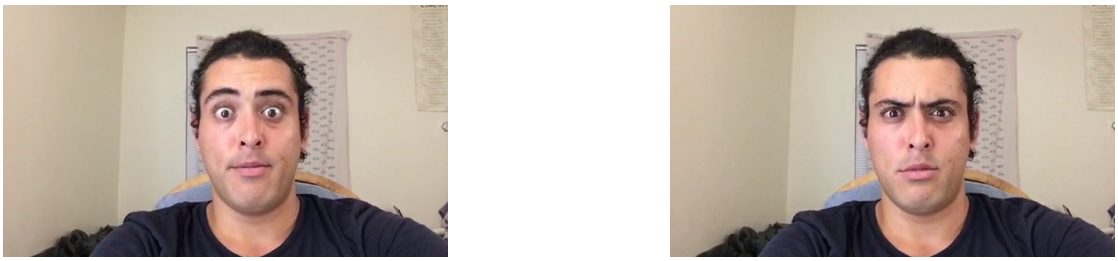


**Figure 4.5.:** Example pictures illustrating case three

Results for eigenfaces are reported in Table 4.9.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 59.38 | 98.98 | 1.02 | 40.62 | 79.18 | 20.82 |
| | nuc | 60.51 | 98.6 | 1.4 | 39.49 | 79.55 | 20.45 |
| | amd | 60.5 | 98.6 | 1.4 | 39.5 | 79.55 | 20.45 |
| 40 | rpi | 67.74 | 98.5 | 1.5 | 32.26 | 83.12 | 16.88 |
| | nuc | 68.46 | 98.6 | 1.4 | 31.54 | 83.53 | 16.47 |
| | amd | 68.41 | 98.6 | 1.4 | 31.59 | 83.5 | 16.5 |
| 80 | rpi | 78.12 | 98.86 | 1.14 | 21.88 | 88.49 | 11.51 |
| | nuc | 78.4 | 98.48 | 1.52 | 21.6 | 88.44 | 11.56 |
| | amd | 78.36 | 98.48 | 1.52 | 21.64 | 88.42 | 11.58 |

**Table 4.9.:** Case three: eigenfaces results

Results for fisherfaces and LBPH are presented in the next page in Table 4.10 and Table 4.11, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 56.1 | 98.08 | 1.92 | 43.9 | 77.09 | 22.91 |
| | nuc | 55.59 | 97.58 | 2.42 | 44.41 | 76.59 | 23.41 |
| | amd | 55.73 | 97.58 | 2.42 | 44.27 | 76.66 | 23.34 |
| 40 | rpi | 63.68 | 84.62 | 15.38 | 36.32 | 74.15 | 25.85 |
| | nuc | 62.89 | 84.54 | 15.46 | 37.11 | 73.72 | 26.28 |
| | amd | 62.89 | 84.54 | 15.46 | 37.11 | 73.72 | 26.28 |
| 80 | rpi | 73.5 | 69.08 | 30.92 | 26.5 | 71.29 | 28.71 |
| | nuc | 73.5 | 69.08 | 30.92 | 26.5 | 71.29 | 28.71 |
| | amd | 73.5 | 69.08 | 30.92 | 26.5 | 71.29 | 28.71 |

**Table 4.10.:** Case three: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 97.96 | 55.91 | 44.09 | 2.04 | 76.93 | 23.07 |
| | nuc | 93.42 | 74.92 | 25.08 | 6.58 | 84.17 | 15.83 |
| | amd | 98.04 | 56.58 | 43.42 | 1.96 | 77.31 | 22.69 |
| 40 | rpi | 96.18 | 73.98 | 26.02 | 3.82 | 85.08 | 14.92 |
| | nuc | 83.5 | 81.32 | 18.68 | 16.5 | 82.41 | 17.59 |
| | amd | 96.13 | 75.3 | 24.7 | 3.87 | 85.72 | 14.28 |
| 80 | rpi | 96.66 | 59.95 | 40.05 | 3.34 | 78.31 | 21.69 |
| | nuc | 86.76 | 75.28 | 24.72 | 13.24 | 81.02 | 18.98 |
| | amd | 96.79 | 60.33 | 39.67 | 3.21 | 78.56 | 21.44 |

**Table 4.11.:** Case three: LBPH results

As per previous cases, average results are shown in Table 4.12. Figure 4.6 gives the comparison of the accuracy of algorithms based on number of images per person. As it can be seen, LBPH performed slightly better than eigenfaces with the IpP of 20 and 40; however with 80 pictures per each person in the training set, eigenfaces outperformed all other algorithms noticeably.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 60.13 | 98.73 | 1.27 | 39.87 | 79.43 | 20.57 |
| | Fisherfaces | 55.81 | 97.75 | 2.25 | 44.19 | 76.78 | 23.22 |
| | LBPH | 96.47 | 62.47 | 37.53 | 3.53 | 79.47 | 20.53 |
| 40 | Eigenfaces | 68.2 | 98.57 | 1.43 | 31.8 | 83.39 | 16.61 |
| | Fisherfaces | 63.15 | 84.57 | 15.43 | 36.85 | 73.86 | 26.14 |
| | LBPH | 91.94 | 76.87 | 23.13 | 8.06 | 84.4 | 15.6 |
| 80 | Eigenfaces | 78.29 | 98.61 | 1.39 | 21.71 | 88.45 | 11.55 |
| | Fisherfaces | 73.5 | 69.08 | 30.92 | 26.5 | 71.29 | 28.71 |
| | LBPH | 93.4 | 65.19 | 34.81 | 6.6 | 79.29 | 20.71 |

**Table 4.12.:** Case three: average results

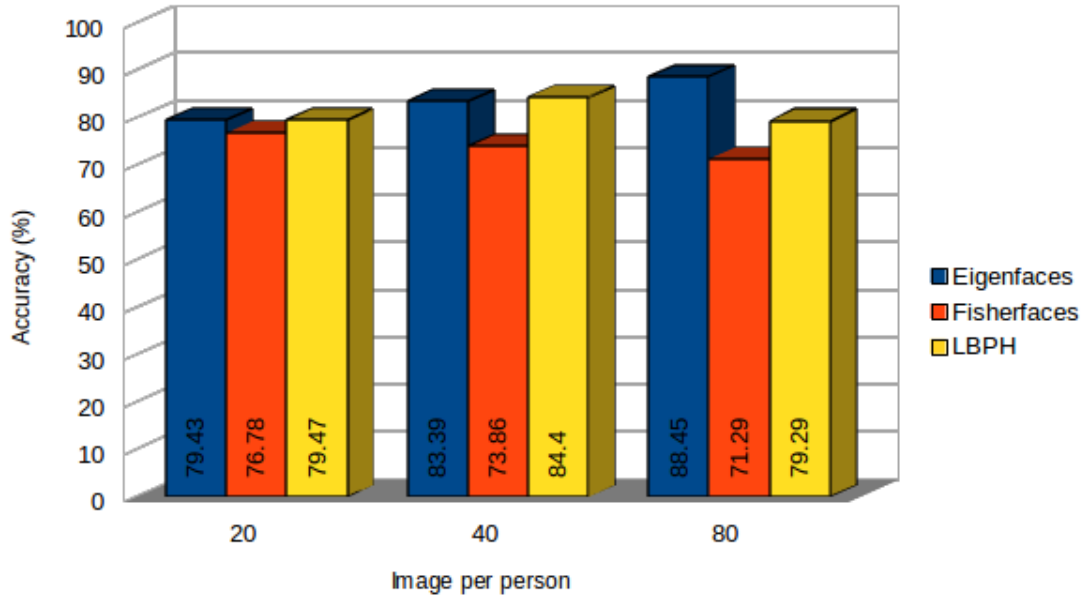



**Figure 4.6.:** Comparison of the accuracy of algorithms for case three

### 4.4.4. Case four: straight under heavy lighting conditions

In this case, subject is looking directly at the camera like case one; however, lighting conditions affect the face image as illustrated in Figure 4.7.

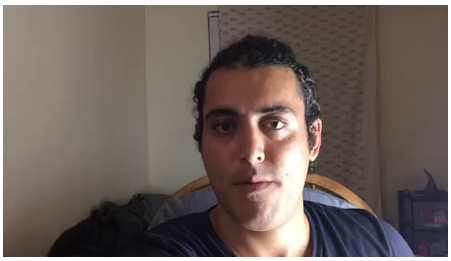
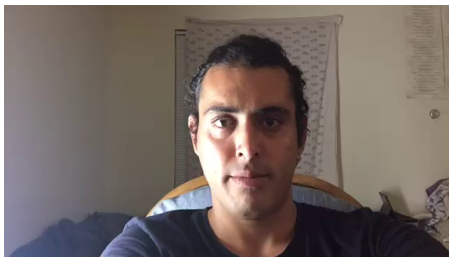



**Figure 4.7.:** Example pictures illustrating case four

Results for eigenfaces are reported in Table 4.13.

| pP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 79.44 | 87.16 | 12.84 | 20.56 | 83.3 | 16.7 |
| | nuc | 80.43 | 89.63 | 10.37 | 19.57 | 85.03 | 14.97 |
| | amd | 80.43 | 89.63 | 10.37 | 19.57 | 85.03 | 14.97 |
| 80 | rpi | 81.81 | 85.55 | 14.45 | 18.19 | 83.68 | 16.32 |
| | nuc | 82.37 | 85.55 | 14.45 | 17.63 | 83.96 | 16.04 |
| | amd | 82.37 | 85.55 | 14.45 | 17.63 | 83.96 | 16.04 |

**Table 4.13.:** Case four: eigenfaces results

Results for fisherfaces and LBPH are presented in the next page in Table 4.14 and Table 4.15, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 12.82 | 100 | 0 | 87.18 | 56.41 | 43.59 |
| | nuc | 13.43 | 100 | 0 | 86.57 | 56.72 | 43.28 |
| | amd | 13.43 | 100 | 0 | 86.57 | 56.72 | 43.28 |
| 80 | rpi | 12.82 | 100 | 0 | 87.18 | 56.41 | 43.59 |
| | nuc | 13.43 | 100 | 0 | 86.57 | 56.72 | 43.28 |
| | amd | 13.43 | 100 | 0 | 86.57 | 56.72 | 43.28 |

**Table 4.14.:** Case four: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 45.83 | 79.35 | 20.65 | 54.17 | 62.59 | 37.41 |
| | nuc | 28.26 | 90.78 | 9.22 | 71.74 | 59.52 | 40.48 |
| | amd | 44.47 | 78.68 | 21.32 | 55.53 | 61.57 | 38.43 |
| 40 | rpi | 93.53 | 60.03 | 39.97 | 6.47 | 76.78 | 23.22 |
| | nuc | 78.02 | 85.18 | 14.82 | 21.98 | 81.6 | 18.4 |
| | amd | 93.94 | 60.72 | 39.28 | 6.06 | 77.33 | 22.67 |
| 80 | rpi | 94.46 | 43.44 | 56.56 | 5.54 | 68.95 | 31.05 |
| | nuc | 83.25 | 72.41 | 27.59 | 16.75 | 77.83 | 22.17 |
| | amd | 95.67 | 42.91 | 57.09 | 4.33 | 69.29 | 30.71 |

**Table 4.15.:** Case four: LBPH results

Average results are shown in Table 4.16. Figure 4.8 gives the comparison of the accuracy of algorithms. As it can be seen for the case of 20 images per person, all algorithms performed poorly since the training set does not include any pictures with the lighting conditions. The training set in this case contains only pictures of subjects under normal lighting conditions.

However, LBPH performed slightly better than the other two with the IpP of 20. When more pictures for each person are added to the training set, which include lighting conditions, results are considerably better as it was expected. Overall, eigenfaces performed better than the other two investigated algorithms.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | Fisherfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | LBPH | 39.52 | 82.93 | 17.07 | 60.48 | 61.23 | 38.77 |
| 40 | Eigenfaces | 80.1 | 88.81 | 11.19 | 19.9 | 84.46 | 15.54 |
| | Fisherfaces | 13.23 | 100 | 0 | 86.77 | 56.61 | 43.39 |
| | LBPH | 88.5 | 68.65 | 31.35 | 11.5 | 78.57 | 21.43 |
| 80 | Eigenfaces | 82.19 | 85.55 | 14.45 | 17.81 | 83.87 | 16.13 |
| | Fisherfaces | 13.23 | 100 | 0 | 86.77 | 56.61 | 43.39 |
| | LBPH | 91.12 | 52.92 | 47.08 | 8.88 | 72.02 | 27.98 |

**Table 4.16.:** Case four: average results

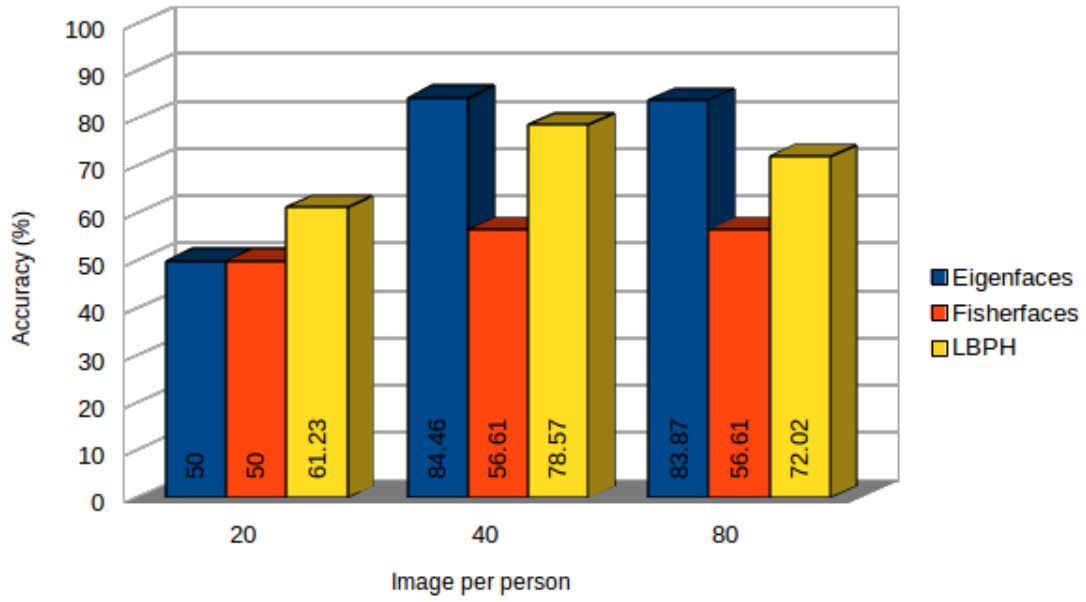



**Figure 4.8.:** Comparison of the accuracy of algorithms for case four

### 4.4.5. Case five: head rotation under heavy lighting conditions

Subject rotates his head the same as case two; however, lighting conditions influence the image. Figure 4.7 shows example images of this condition. It needs to be noted that, such as case two, this case also includes head rotation to the sides, up and down with lighting directions from left and right; however in order to avoid repetition, only two rotations for each lighting direction are shown.

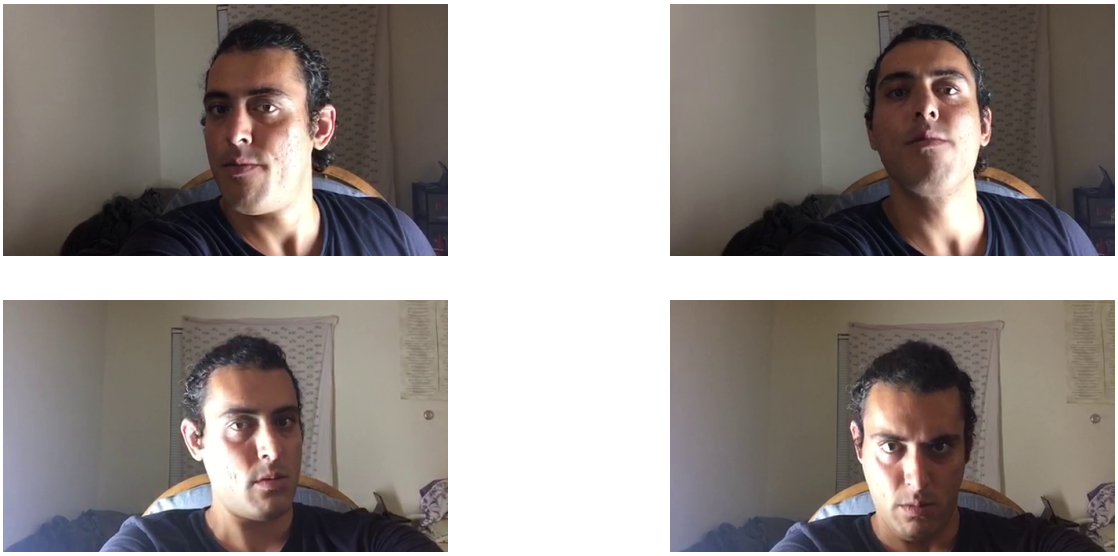


**Figure 4.9.:** Example pictures illustrating case five

Table 4.17 presents the results for eigenfaces.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 66.74 | 85.82 | 14.18 | 33.26 | 76.28 | 23.72 |
| | nuc | 64.9 | 88.09 | 11.91 | 35.1 | 76.49 | 23.51 |
| | amd | 64.87 | 88.09 | 11.91 | 35.13 | 76.48 | 23.52 |
| 80 | rpi | 68.06 | 84.37 | 15.63 | 31.94 | 76.21 | 23.79 |
| | nuc | 67.99 | 84.37 | 15.63 | 32.01 | 76.18 | 23.82 |
| | amd | 68.03 | 84.37 | 15.63 | 31.97 | 76.2 | 23.8 |

**Table 4.17.:** Case five: eigenfaces results

Results for fisherfaces and LBPH are given in Table 4.18 and Table 4.19, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 3.87 | 100 | 0 | 96.13 | 51.94 | 48.06 |
| | nuc | 3.99 | 100 | 0 | 96.01 | 51.99 | 48.01 |
| | amd | 3.91 | 100 | 0 | 96.09 | 51.95 | 48.05 |
| 80 | rpi | 3.71 | 100 | 0 | 96.29 | 51.86 | 48.14 |
| | nuc | 3.99 | 100 | 0 | 96.01 | 51.99 | 48.01 |
| | amd | 3.91 | 100 | 0 | 96.09 | 51.95 | 48.05 |

**Table 4.18.:** Case five: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 37.89 | 81.95 | 18.05 | 62.11 | 59.92 | 40.08 |
| | nuc | 23.68 | 93.77 | 6.23 | 76.32 | 58.73 | 41.27 |
| | amd | 37.82 | 83.34 | 16.66 | 62.18 | 60.58 | 39.42 |
| 40 | rpi | 83.96 | 69.47 | 30.53 | 16.04 | 76.71 | 23.29 |
| | nuc | 67.16 | 93.24 | 6.76 | 32.84 | 80.2 | 19.8 |
| | amd | 83.3 | 80.93 | 19.07 | 16.7 | 82.12 | 17.88 |
| 80 | rpi | 84.92 | 64.89 | 35.11 | 15.08 | 74.91 | 25.09 |
| | nuc | 69.91 | 84.71 | 15.29 | 30.09 | 77.31 | 22.69 |
| | amd | 84.3 | 64.86 | 35.14 | 15.7 | 74.58 | 25.42 |

**Table 4.19.:** Case five: LBPH results

Table 4.20 shows average results and Figure 4.10 gives the comparison of the accuracy of algorithms based on number of images per person. As it can be seen when there is only 20 pictures per each person, all algorithms performed poorly again. In this case, however, LBPH came in the first place, eigenfaces second and fisherfaces did not perform as good as the other two algorithms.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | Fisherfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | LBPH | 33.13 | 86.36 | 13.64 | 66.87 | 59.74 | 40.26 |
| 40 | Eigenfaces | 65.5 | 87.33 | 12.67 | 34.5 | 76.42 | 23.58 |
| | Fisherfaces | 3.92 | 100 | 0 | 96.08 | 51.96 | 48.04 |
| | LBPH | 78.14 | 81.22 | 18.78 | 21.86 | 79.68 | 20.32 |
| 80 | Eigenfaces | 68.03 | 84.37 | 15.63 | 31.97 | 76.2 | 23.8 |
| | Fisherfaces | 3.87 | 100 | 0 | 96.13 | 51.93 | 48.07 |
| | LBPH | 79.71 | 71.49 | 28.51 | 20.29 | 75.6 | 24.4 |

**Table 4.20.:** Case five: average results

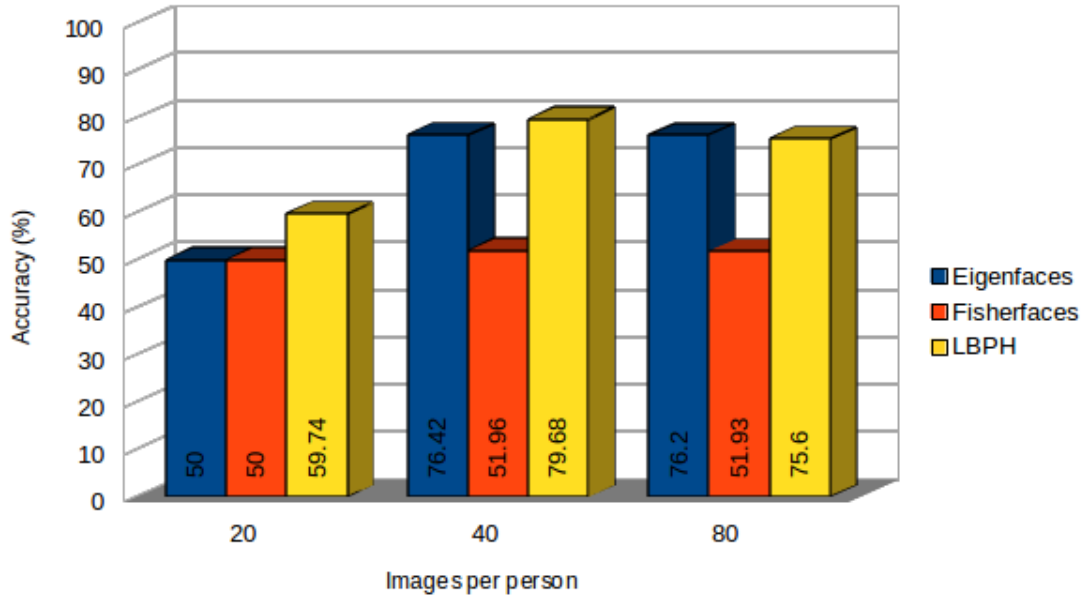



**Figure 4.10.:** Comparison of the accuracy of algorithms for case five

### 4.4.6. Case six: expressions under heavy lighting conditions

In this criterion, subject has expressions such as happy, sad and angry; moreover, lighting conditions also affect the image. Figure 4.11 shows example images of this condition.

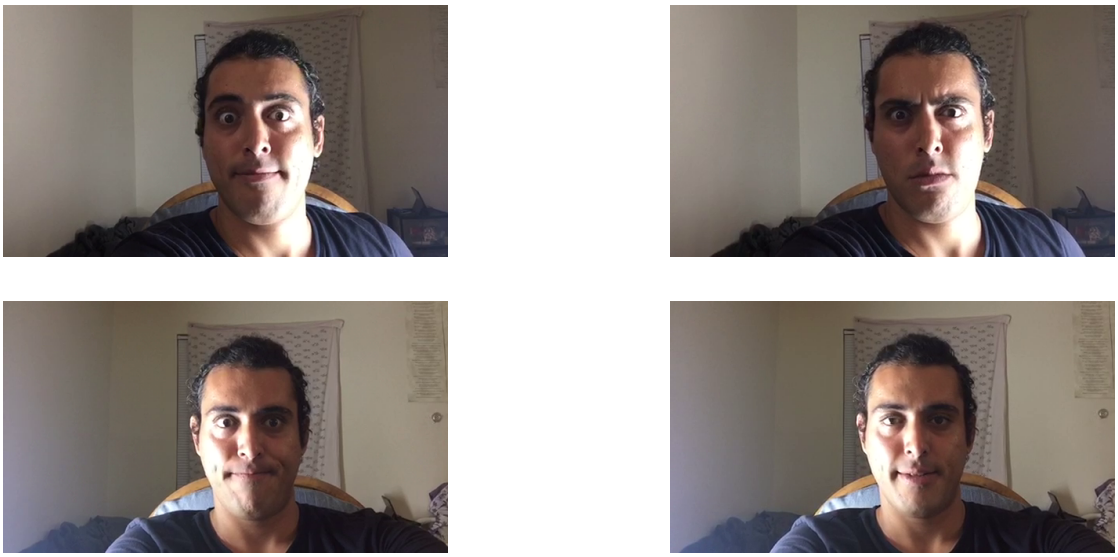


**Figure 4.11.:** Example pictures illustrating case six

Table 4.21 presents the results for eigenfaces.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 58.65 | 96.47 | 3.53 | 41.35 | 77.56 | 22.44 |
| | nuc | 59.02 | 96.62 | 3.38 | 40.98 | 77.82 | 22.18 |
| | amd | 58.87 | 96.62 | 3.38 | 41.13 | 77.75 | 22.25 |
| 80 | rpi | 68.02 | 91.88 | 8.12 | 31.98 | 79.95 | 20.05 |
| | nuc | 67.97 | 91.88 | 8.12 | 32.03 | 79.92 | 20.08 |
| | amd | 68.06 | 91.88 | 8.12 | 31.94 | 79.97 | 20.03 |

**Table 4.21.:** Case six: eigenfaces results

Results for fisherfaces and LBPH are reported in Table 4.22 and Table 4.23, respectively.

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 0 | 100 | 0 | 100 | 50 | 50 |
| | nuc | 0 | 100 | 0 | 100 | 50 | 50 |
| | amd | 0 | 100 | 0 | 100 | 50 | 50 |
| 40 | rpi | 8.75 | 100 | 0 | 91.25 | 54.38 | 45.62 |
| | nuc | 8.85 | 100 | 0 | 91.15 | 54.43 | 45.57 |
| | amd | 8.85 | 100 | 0 | 91.15 | 54.43 | 45.57 |
| 80 | rpi | 8.59 | 100 | 0 | 91.41 | 54.29 | 45.71 |
| | nuc | 8.85 | 100 | 0 | 91.15 | 54.43 | 45.57 |
| | amd | 8.85 | 100 | 0 | 91.15 | 54.43 | 45.57 |

**Table 4.22.:** Case six: fisherfaces results

| IpP | Platform | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | rpi | 42.37 | 71.06 | 28.94 | 57.63 | 56.72 | 43.28 |
| | nuc | 27.42 | 89.8 | 10.2 | 72.58 | 58.61 | 41.39 |
| | amd | 42.77 | 69.59 | 30.41 | 57.23 | 56.18 | 43.82 |
| 40 | rpi | 88.62 | 56.08 | 43.92 | 11.38 | 72.35 | 27.65 |
| | nuc | 71.64 | 90.17 | 9.83 | 28.36 | 80.9 | 19.1 |
| | amd | 87.77 | 61.93 | 38.07 | 12.23 | 74.85 | 25.15 |
| 80 | rpi | 90.83 | 46.56 | 53.44 | 9.17 | 68.69 | 31.31 |
| | nuc | 78.95 | 80.95 | 19.05 | 21.05 | 79.95 | 20.05 |
| | amd | 90.8 | 47.9 | 52.1 | 9.2 | 69.35 | 30.65 |

**Table 4.23.:** Case six: LBPH results

Average of results for all hardware platforms are presented in Table 4.24 which shows the partial results for this case. Moreover, Figure 4.12 compares the accuracy of the algorithms according to the number of images per person. As it can be seen, eigenfaces performed better than the other two algorithms and best results were obtained when 80 pictures for each person in the training set were given to this algorithm to train on. LBPH performed the second best and fisherfaces could not compete with the other two facial feature extraction methods.

| IpP | Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|---|
| 20 | Eigenfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | Fisherfaces | 0 | 100 | 0 | 100 | 50 | 50 |
| | LBPH | 37.52 | 76.82 | 23.18 | 62.48 | 57.17 | 42.83 |
| 40 | Eigenfaces | 58.85 | 96.57 | 3.43 | 41.15 | 77.71 | 22.29 |
| | Fisherfaces | 8.82 | 100 | 0 | 91.18 | 54.41 | 45.59 |
| | LBPH | 82.68 | 69.39 | 30.61 | 17.32 | 76.03 | 23.97 |
| 80 | Eigenfaces | 68.02 | 91.88 | 8.12 | 31.98 | 79.95 | 20.05 |
| | Fisherfaces | 8.76 | 100 | 0 | 91.24 | 54.38 | 45.62 |
| | LBPH | 86.86 | 58.47 | 41.53 | 13.14 | 72.67 | 27.33 |

**Table 4.24.:** Case six: average results

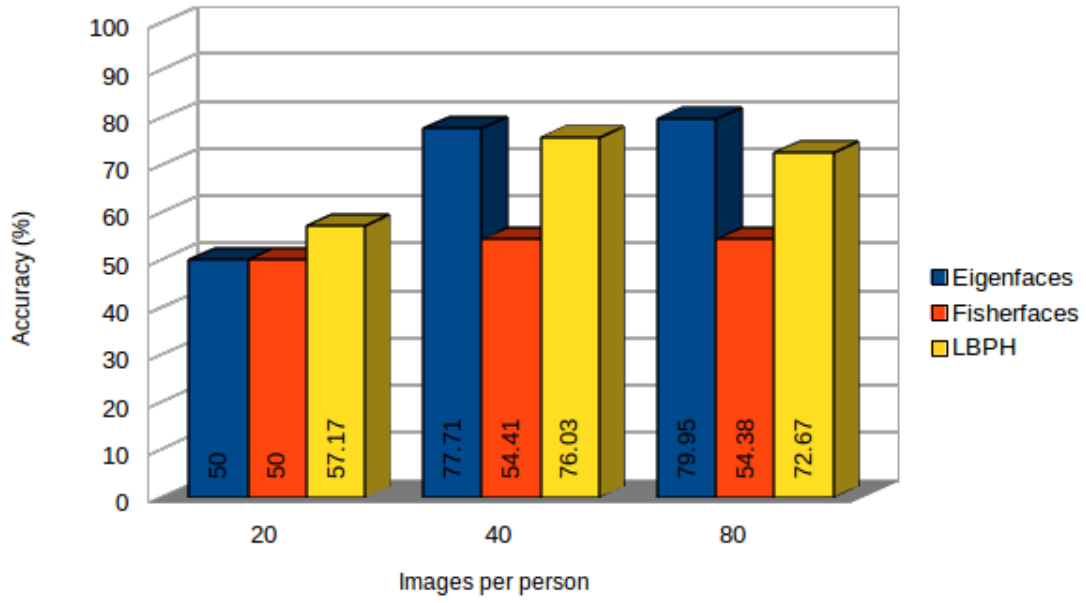



**Figure 4.12.:** Comparison of the accuracy of algorithms for case six

# 5. Conclusion

This chapter discusses the trend of data for all investigated algorithms and presents the overall results.

## 5.1. Trend of data

The trend of results will be discussed here. First, each algorithm will be investigated separately then, all three will be compared. It needs to be noted that the X-axis of the following graphs shows the six cases presented before, these cases are repeated here:

1. Straight under normal lighting conditions
2. Head rotation under normal lighting conditions
3. Expressions under normal lighting conditions
4. Straight under heavy lighting conditions
5. Head rotation under heavy lighting conditions
6. Expressions under heavy lighting conditions

It also needs to be reminded that 20 pictures per each person only include straight and head rotation under normal lighting condition in the training set. 40 adds straight and rotation under heavy lighting conditions to the the set of train images and 80 includes all possible combinations including expressions.

Figure 5.1 shows the eigenfaces trend. As it can be seen with the increase of training pictures, results improve in all cases except for case two in which head rotation under normal lighting condition is examined. In this case, 20 pictures per perosn in the training set matches exactly with the recognition video; in other words, there is no lighting effect in the training set or in the test video. Therefore, results are higher than IpP of 40 and 80 in which more pictures under lighting conditions are added to the

training set which are not examined in the recognition video and that led to a slightly lower accuracy.

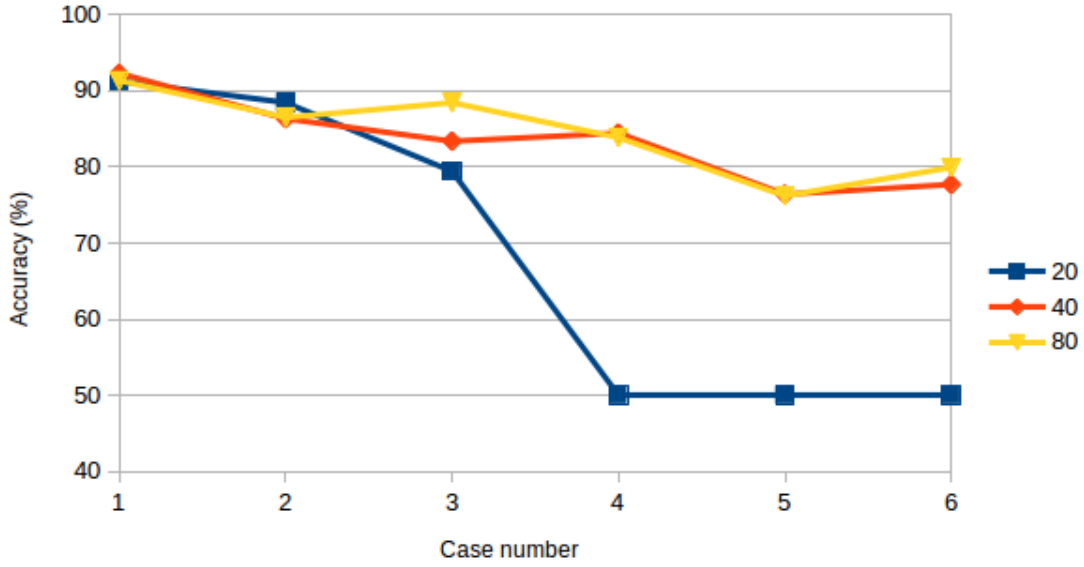



**Figure 5.1.:** Trend of data for eigenfaces

Moreover it can be seen that for 20 images per person, the accuracy of exactly 50 percent is received for cases four, five and six. That is justified in a way that the last three cases investigate the performance of the algorithms under heavy lighting conditions and IpP of 20 does not include any lighting condition in the training set. As a result, similarity of recognition decreases dramatically and with the idea of having a critically low threshold, which was explained in the previous chapter, even if the algorithm recognizes correctly, results will not be accepted and will be filtered.

In order to give a rationale for that assume a case in which there are two people in front of the camera to be recognized, one known and one unknown. The scale of similarity is 0-10 and threshold is 5 which means if the similarity is more than 5, the case will be accepted and below that will be classified as unknown. When the IpP is 20 for the last three cases, assume the algorithm returns a value of 4 for similarity of the known person and a value of 3 for unknown which are both below the threshold.

Although the algorithm recognized the known person correctly but the similarity is too low that it is almost the same as if an unknown person was misrecognized and returned as the known person, as in this scenario the unknown person has a similarity value

of very close to that of the known person. Therefore, the algorithm does not accept the known person and reports both of them as unknown which results in TN rate of 100 percent and TP rate of 0 percent. Accuracy is defined according to Equation 4.1 as an average of these two rates in percentages; therefore, the accuracy will be exactly 50 percent.

Moreover, it can be seen in Figure 5.1 that when there is 40 and 80 pictures per each person in the training set results are almost the same for all cases except for case three and six in which IpP of 80 performed better. In these two cases, the expressions will be examined under normal lighting conditions, case three, and under heavy lighting conditions, case six. IpP of 40 only includes straight and head rotation under normal and heavy lighting conditions in the training set. 80 also adds the expressions to the training set; therefore, results are better for these two cases.

Figure 5.2 shows the fisherfaces trend.

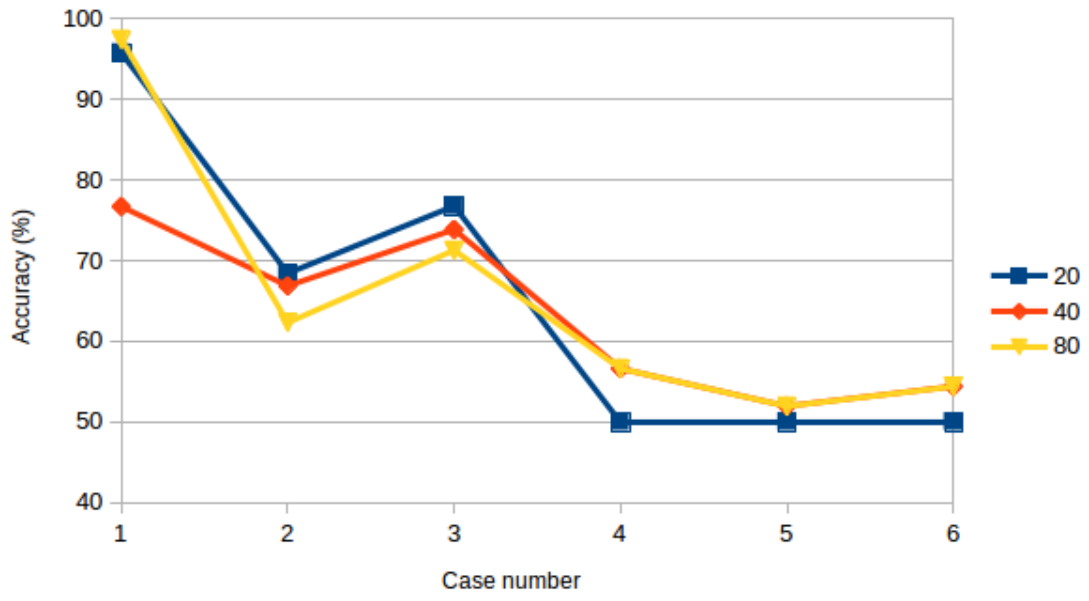



**Figure 5.2.:** Trend of data for fisherfaces

For fisherfaces, the same scenario with the exact 50 percent accuracy applies to the last three cases with the IpP of 20. As it can be seen, for fisherfaces results are mostly in the same range for 20, 40 and 80 pictures per person in the training set. It can be justified that increasing the IpP improved the TP rate but at the same time deteriorated the TN rate resulting in almost the same average of the two values which is defined as

accuracy; however, IpP of 80 performed slightly better on average of all six cases.

Figure 5.3 shows the LBPH trend.

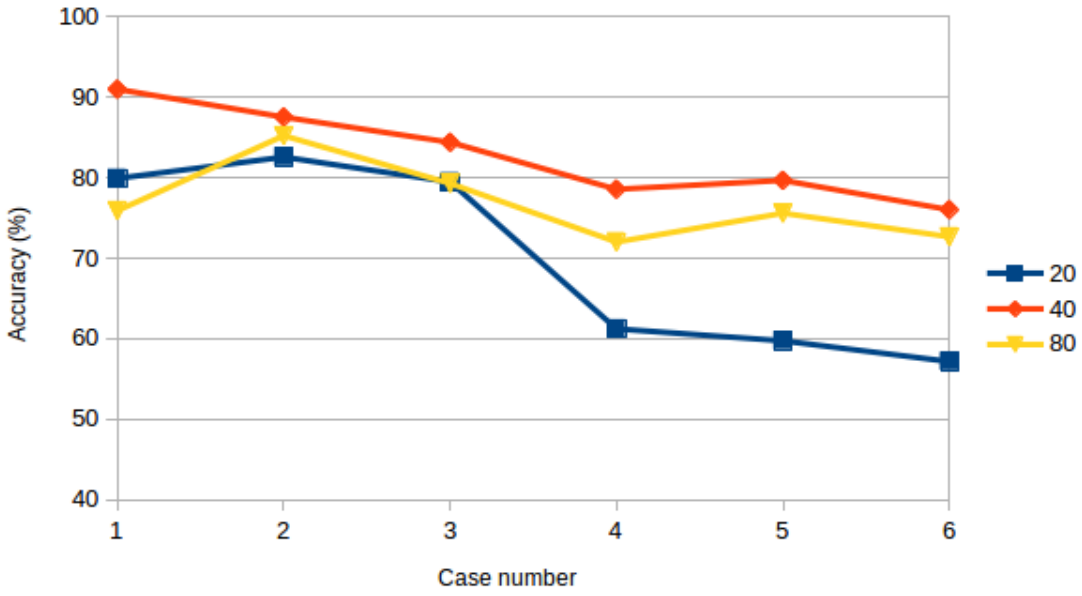



**Figure 5.3.:** Trend of data for LBPH

LBPH has a higher performance for the the last three cases with IpP of 20 than the other two algorithms. Unlike eigenfaces and fisherfaces, IpP of 40 performed the highest with an increase of 12.85 percent accuracy when 20 pictures per person is selected and 6.07 percent when 80 is chosen on average of all cases. Figure 5.4 shows the trend of data for the three investigated facial feature extraction methods in all six cases. It needs to be stated that these results are the average of 20, 40 and 80 pictures per person in each case.

As it can be seen, eigenfaces outperformed the other two algorithms in all cases except for case five in which LBPH had the highest performance. Fisherfaces had the lowest performance except for case one in which it outperformed LBPH. Eigenfaces and fisherfaces resulted in the highest accuracy for case one but LBPH had its best performance for case two.

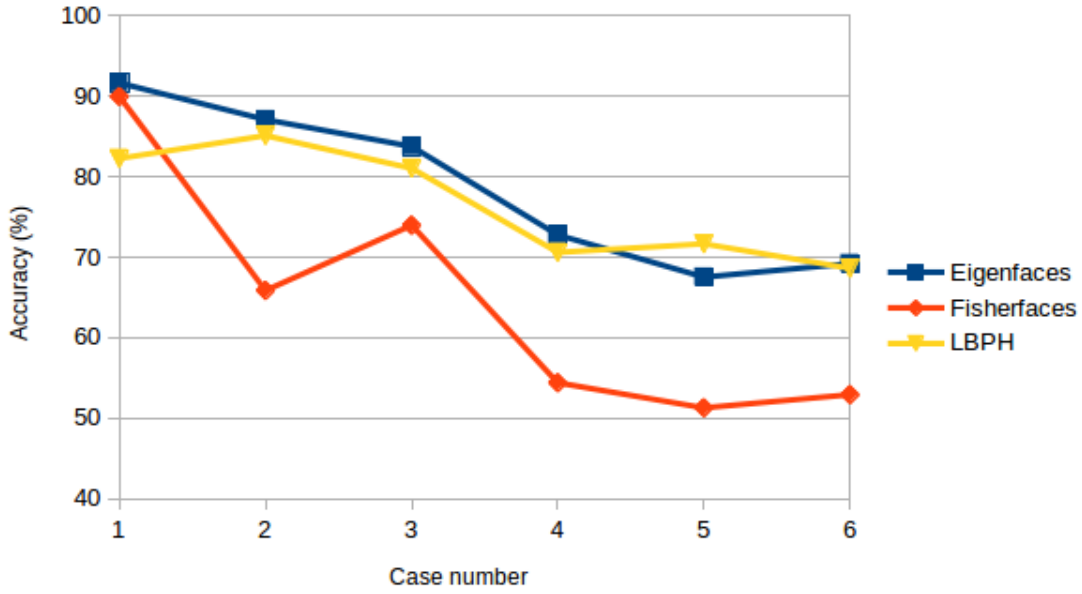



**Figure 5.4.:** Trend of data for all algorithms

## 5.2. Overall performance

Partial results for each case on each hardware platform were reported in the previous chapter. This section presents the overall performances of all algorithms. It needs to be stated that Frames Per Second (FPS) is defined as a rate which determines how fast the algorithms perform in the sense of number of frames they can process per each second. The investigated feature extraction methods are designed to be as fast as possible and the recognition software is tailored to have relatively high value of FPS.

In order to clarify, the resolution of each obtained frame are first reduced to $640 \times 480$ in which resolution, frames are small enough for the facial feature extraction methods to perform their algorithms in a timely manner, at the same time not to lose a huge amount of quality.

Moreover, no output picture is shown and results are only saved as a text file for each frame whether the algorithms recognized the person and what is the similarity of recognition. With this being done, the FPS rate almost doubled only because the software does not need to go through displaying the output picture. It needs to be noted that frames per second is calculated based on the time it takes for recognition to be done; in other words, frame query is excluded from this time.

| | Algorithm | TP | TN | FP | FN | Accuracy | Error | FPS |
|---|---|---|---|---|---|---|---|---|
| rpi | Eigenfaces | 65.32 | 91.77 | 8.23 | 34.68 | 78.55 | 21.45 | 4.03 |
| | Fisherfaces | 34.33 | 95.22 | 4.78 | 65.67 | 64.78 | 35.22 | |
| | LBPH | 85.23 | 65.09 | 34.91 | 14.77 | 75.16 | 24.84 | |
| nuc | Eigenfaces | 65.5 | 91.96 | 8.04 | 34.5 | 78.73 | 21.27 | 7.65 |
| | Fisherfaces | 34.36 | 95.09 | 4.91 | 65.64 | 64.72 | 35.28 | |
| | LBPH | 74.41 | 83.2 | 16.8 | 25.59 | 78.81 | 21.19 | |
| amd | Eigenfaces | 65.5 | 91.96 | 8.04 | 34.5 | 78.73 | 21.27 | 5.55 |
| | Fisherfaces | 34.35 | 95.09 | 4.91 | 65.65 | 64.72 | 35.28 | |
| | LBPH | 85.16 | 66.25 | 33.75 | 14.84 | 75.71 | 24.29 | |

**Table 5.1.:** Comparison of results on each hardware platform

As it can be seen, all hardware platforms performed very much in the same range with very small differences for eigenfaces and fisherfaces. In case of LBPH, Intel NUC outperformed Raspberry Pi and AMD hardware with 3,65 and 3,1 percent, respectively.

LBPH on Intel NUC had a lower TP rate but higher TN rate than the other two platforms resulting in a higher average defined as accuracy. Moreover, Intel NUC had a higher FPS value and much lower duration time than the other two embedded hardwares. The whole set of experiments took a little more than 4 hours on this hardware, while exactly the same calculation took more than 25 hours on Raspberry Pi 2.

Table 5.2 presents the overall performances of the three algorithms. It needs to be noted results in the following table are an average of all six cases for IpP of 20, 40 and 80 each of which is run on three hardware platforms. Moreover, all experiments have been done twice, one time with two people as known and one as unknown and another time with three people as known and two as unknown.

| Algorithm | TP | TN | FP | FN | Accuracy | Error |
|---|---|---|---|---|---|---|
| Eigenfaces | 65.44 | 91.89 | 8.10 | 34.55 | 78.66 | 21.33 |
| Fisherfaces | 34.34 | 95.13 | 4.86 | 65.65 | 64.74 | 35.25 |
| LBPH | 81.60 | 71.51 | 28.48 | 18.39 | 76.55 | 23.44 |

**Table 5.2.:** Overall results of the investigated algorithms

Eigenfaces outperformed all other facial feature extraction methods. LBPH had slightly lower accuracy of recognition and fisherfaces performed with the accuracy of 13,92 percent lower than eigenfaces. LBPH, however, had the highest TP rate and fisherfaces had the highest TN rate among all algorithms.

# Appendices

# A. Implementation of the algorithms

This section presents the implementation of the investigated extraction methods in OpenCV to better understand how previously explained mathematics can be used.

## A.1. Eigenfaces

Eigenfaces is implemented in a two-step process

- **Train**
  Accepts a set of training images and projects them into the PCA sub-space.
- **Predict**
  Finds the nearest neighbor of the new face image.

### A.1.1. Train

The entire train function will be explained part-by-part in the following. It needs to be noted that fisherfaces and LBPH methods share parts of the following code. These parts are only explained here and are not repeated in the next sections.

The train function accepts a set of training images and labels.

```
void cv::Eigenfaces::train(InputArray src, InputArray _lbls) {
```

In practice, as it was explained before, more than one image from a person is required in different poses, angles and lighting conditions. Labels are used in order to put the pictures of the same person in the same group; therefore, all images of the same subject have the same label.

The first part of the function verifies that the input training set has enough pictures; furthermore, it ensures that the labels are given as integer.

```
if(src.total() == 0) {

        string error_message = format("Empty training data was given. You'll need
            more than one sample to learn a model.");
        CV_Error(CV_StsUnsupportedFormat, error_message);

}

else if(_lbls.getMat().type() != CV_32SC1) {

        string error_message = format("Labels must be given as integer (CV_32SC1).
             Expected %d, but was %d.", CV_32SC1, _lbls.type());
        CV_Error(CV_StsUnsupportedFormat, error_message);
}
```

If there are enough pictures in the training set, the function makes sure they are equal in size since the implementation of eigenfaces only accepts input images of the same size.

```
if(src.total() > 1) {
        for(int i = 1; i < src.total(); i++) {

                if(src.getMat(i-1).total() != src.getMat(i).total()) {
                        string error_message = format("In the Eigenfaces method all
                             input samples (training images) must be of equal size!
                             Expected %d pixels, but was %d pixels.", src.getMat(i
                            -1).total(), src.getMat(i).total());
                        CV_Error(CV_StsUnsupportedFormat, error_message);

                } // end of if
        } // end of for
} // end of if
```

How many labels are used should match exactly with the number of people in the training set in order for the algorithm to know each picture belongs to whom in the database. In order to clarify, assuming there are 5 people in the training set with 10 images per person, a matrix of 5x10 will be created. In this case, 5 labels need to be provided each representing one subject in the database. This is ensured in the following.

```
        vector<int> labels = _lbls.getMat();
        Mat data = asRowMatrix(src, CV_64FC1);
        int n = data.rows;
        int d = data.cols;

        if(n != labels.size()) {
                        string error_message = format("The number of samples (src) must
                            equal the number of labels (labels). Was len(samples)=%d, len(
                            labels)=%d.", n,labels.size());
                        CV_Error(CV_StsBadArg, error_message);
        }
```

Variables appearing in the following are the private variables of the class Eigenfaces, which are used to store the inputs and outputs of the PCA function.

```
        if((_num_components <= 0) || (_num_components > n))
                _num_components = n;

        PCA pca(data, Mat(), CV_PCA_DATA_AS_ROW, _num_components);

        _mean = pca.mean.reshape(1,1);
        _eigenvalues = pca.eigenvalues.clone();
        _eigenvectors = transpose(pca.eigenvectors);
        _labels = labels;
```

Finally, the images in training database are projected to the sub-space which will be used by the predict function.

```
        for(int sampleIdx = 0; sampleIdx < data.rows; sampleIdx++) {
                Mat p = subspace::project(_eigenvectors, _mean, data.row(sampleIdx).clone
                    ());
                        this->_projections.push_back(p);

        } // end of for
} // end of function
```

### A.1.2. Predict

The predict function is explained in this section. Fisherfaces and LBPH methods share parts of the following code which are not repeated in the next chapters. The function accepts a new face image, a class and a minimum distance variable used for classification of the image.

```
void cv::Eigenfaces::predict(InputArray _src, int &minClass, double &minDist)
    const {
```

It is obvious that the train function must be called before the prediction can be done. Hence, it is firstly ensured that the projection matrix, which is filled by the train function, is not empty.

```
Mat src = _src.getMat();
       if(_projections.empty()) {
              string error_message = "This cv::Eigenfaces model is not computed
                  yet. Did you call cv::Eigenfaces::train?";
       CV_Error(CV_StsError, error_message);
}
```

Moreover, the function verifies that the test image has the same size as the images in the training set.

```
else if(_eigenvectors.rows != src.total()) {
              string error_message = format("Wrong input image size. Reason:
                  Training and Test images must be of equal size!      Expected
                  an image with %d elements, but got %d.", _eigenvectors.rows,
                  src.total());
              CV_Error(CV_StsError, error_message);
}
```

Then, the test image will be projected to the sub-space according to the eigenvectors calculated by PCA in the train function.

```
Mat q = subspace::project(_eigenvectors, _mean, src.reshape(1,1));
```

Finally, the function finds the nearest neighbor in the subspace by calculating the euclidean distance between the new image and all the projected images in the training database.

```
minDist = DBL_MAX;
minClass = -1;

for(int sampleIdx = 0; sampleIdx < _projections.size(); sampleIdx++) {
       double dist = norm(_projections[sampleIdx], q, NORM_L2);
```

The nearest neighbor will be chosen as the training image which has the lowest value of distance when it is smaller than the threshold. If this picture is found, its label will

be reported as the class the test image belongs to.

```
            if((dist < minDist) && (dist < _threshold))
            {
                    minDist = dist;
                    minClass = _labels[sampleIdx];

            } // end of if
        } // end of for
} // end of function
```

## A.2. Fisherfaces

Fisherfaces is also implemented in OpenCV in a similar two step-process, i.e. train and predict.

### A.2.1. Train

The first part of the function, which tests for errors in the input images and stores the variables in the local ones, is the same as previous train function and will not be explained in order to avoid repetition. In the following, first PCA is used to map the input data to a $(N - c)$-dimensional subspace, then LDA is applied on the projected data to reduce the dimensionality to $(c - 1)$.

```
if((_num_components <= 0) || (_num_components > (C-1)))
        _num_components = (C-1);

PCA pca(data, Mat(), CV_PCA_DATA_AS_ROW, (N-C));
subspace::LDA lda(pca.project(data), labels, _num_components);
```

In the next step, the eigenvalues of the LDA is stored and the projection matrix is calculated as the multiplication of PCA eigenvectors and LDA eigenvectors.

```
lda.eigenvalues().convertTo(_eigenvalues, CV_64FC1);
gemm(pca.eigenvectors, lda.eigenvectors(), 1.0, Mat(), 0.0, _eigenvectors,
    GEMM_1_T);
```

Finally, the images in training database are projected to the sub-space which will be used by the predict function.

```
lda.eigenvalues().convertTo(_eigenvalues, CV_64FC1);
gemm(pca.eigenvectors, lda.eigenvectors(), 1.0, Mat(), 0.0, _eigenvectors,
    GEMM_1_T);
```

### A.2.2. Predict

The predict function of fisherfaces is the same as that of eigenfaces. The only difference is that the input test picture will be projected to the LDA sub-space.

```
Mat q = subspace::project(_eigenvectors, _mean, src.reshape(1,1));
```

## A.3. Local binary patterns histogram

LBPH is also implemented as train and predict which will be explained in the following.

### A.3.1. Train

Once more, in order to avoid repetition, the first part of the function will not be explained. In the subsequent lines, for all input images in the training set, the LBP labels and the spatial histogram will be calculated. The histograms are then passed to the predict function to classify a new test image.

```
for(int sampleIdx = 0; sampleIdx < src.size(); sampleIdx++) {
              Mat lbp_image = elbp(src[sampleIdx], _radius, _neighbors);

              Mat p = spatial_histogram(
              lbp_image,
              static_cast<int>(std::pow(2.0,              static_cast<double>(
                  _neighbors))),
              _grid_x,
              _grid_y,
              true
       );
       _histograms.push_back(p);
}
```

### A.3.2. Predict

The predict function accepts a new face image, calculates the LBP labels and provides the spatial histogram of the labeled image.

```
Mat src = _src.getMat();
Mat lbp_image = elbp(src, _radius, _neighbors);

Mat query = spatial_histogram(
                      lbp_image,
                      static_cast<int>(std::pow(2.0,
                      static_cast<double>(_neighbors))),
                      _grid_x,
                      _grid_y,
                      true );
```

Then compares the new histogram with the ones provided by the train function to find the nearest neighbor, i.e. the one with the lowest euclidean distance if the distance is smaller than the given threshold.

```
minDist = DBL_MAX;
minClass = -1;

for(int sampleIdx = 0; sampleIdx < _histograms.size(); sampleIdx++) {

        double dist = compareHist(_histograms[sampleIdx], query, CV_COMP_CHISQR);
        if((dist < minDist) && (dist < _threshold)) {
                minDist = dist;
                minClass = _labels[sampleIdx];
        }
}
```

# List of Figures

# List of Tables

# Bibliography


[AH06] C. A.H.Boualleg and H.Tebbikh. "Automatic Face Recognition Using Neural Network-PCA." In: *Proc. 2nd Information and Communication Technologies* (2006), pp. 1920–1925.

[Ana+02] C. Anagnostopoulos, I. Anagnostopoulos, D. Vergados, I. Papaleonidopoulos, E. Kayafas, V. Loumos, and G. Stasinopoulos. "A Probabilistic Neural Network For Face Detection On Segmented Skin Areas Based On Fuzzy Rules." In: *Proc. IEEE MELECON 2002* (2002), pp. 493–497.

[Azu97] R. T. Azuma. "A Survey of Augmented Reality." In: *Presence: Teleoperators and Virtual Environments* (1997), pp. 355–385.

[Bal00] L. S. Balasuriya. "Frontal View Human Face Detection and Recognition." In: *Department of Statistics and Computer Science, University of Colombo* (2000).

[CL06] R. L. C. Shavers and G. Lebby. "An SVM-Based Approach To Face Detection." In: *Proc. 38th Southeastern Symposium on System Theory* (2006), pp. 362–366.

[CT06] J. T. C.C. Tsai W.C. Cheng and C. Tao. "Face Detection Using Eigenface And Neural Network." In: *Proc. 2006 IEEE International Conference on Systems, Man, and Cybernetics* (2006), pp. 4343–4347.

[DC09] W. R. Dong Hyun Jeong Caroline Ziemkiewicz and R. Chang. "Understanding Principal Component Analysis Using a Visual Analytics Tool." In: *Charlotte Visualization Center, UNC Charlotte* (2009).

[DK99] E. D. D. Anijiantis and G. Kukkinakis. "A Neural Network Method For Accurate Face Detection On Arbitrary Images." In: *Proc. The 6th IEEE International Conference on Electronics, Circuits and Systems* (1999), pp. 109–112.

[Emm] Emmasbox. *How It Works*. `http://www.emmasbox.de/`. Accessed: 2015-07-29.

[Foua] R. P. Foundation. *The Making of Pi*. `https://www.raspberrypi.org/about/`. Accessed: 2015-08-12.

[Foub] R. P. Foundation. *What is a Raspberry Pi?* `https://www.raspberrypi.org/help/what-is-a-raspberry-pi/`. Accessed: 2015-08-12.

[GÜR11] C. GÜREL. "Development of a Face Recognition System." In: *The gradure school of natural and applied sciences of Atilim university* (2011).

[HJ10] Y. Y. H. Guo and Q. Jia. "Face Detection With Abstract Template." In: *Proc. 2010 3rd International Congress on Image And Signal Processing* (2010), pp. 129–134.

[HL04] Q. L. H. Jin and H. Lu. "Face Detection Using One-class-based Support Vectors." In: *Proc. Sixth IEEE International Conference on Automatic Face and Gesture Recognition* (2004), pp. 457–462.

[KL02] C. O. K. Seo W. Kim and J. Lee. "Face Detection And Facial Feature Extraction Using Color Snake." In: *Proc. ISIE 2002 - 2002 IEEE International Symposium on Industrial Electronics* (2002), pp. 457–462.

[LX06] X. S. L. Zhao and X. Xu. "Face Detection Based On Facial Features." In: *Proc. ICSP2006* (2006).

[LY03] A. L. Zhi-fang Y. Zhi-sheng and W. Yun-qiong. "Face Detection And Facial Feature Extraction In Color Image." In: *Proc. The Fifth International Conference on Computational Intelligence and Multimedia Applications* (2003), pp. 126–130.

[Mar10] I. Marquès. "Face Recognition Algorithms." In: *Universidad del Paìs Vasco* (2010).

[Opea] OpenCV. *Face Recognition with OpenCV*. `http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html`. Accessed: 2015-08-16.

[Opeb] OpenCV. *Histogram Equalization*. `http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html`. Accessed: 2015-09-02.

[Opec] OpenCV. *Smoothing Images*. `http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.htmll`. Accessed: 2015-08-16.

[PK97] J. P. H. Peter N. Belhumeur and D. J. Kriegman. "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection." In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 19, NO. 7* (1997), pp. 711–720.

[RT96] N. C. Ramchand Hablani and S. Tanwani. "Recognition of Facial Expressions using Local Binary Patterns of Important Facial Parts." In: (1996), pp. 163–170.

[RY09] J. Ruan and J. Yin. "Face Detection Based On Facial Features And Linear Support Vector Machines." In: *Proc. 2009 International Conference on Communication Software and Networks* (2009), pp. 371–375.

[Sem] SemiconductorStore. *3.5" Embedded SBC w/ AMD G-Series SOC 25W quad-core GX-420CA*. `http://www.semiconductorstore.com/cart/pc/viewPrd.asp?idproduct=48684`. Accessed: 2015-09-02.

[SK87] L. Sirovich and M. Kirby. "Low-dimensional procedure for the characterization of human faces." In: *Journal of the Optical Society of America A - Optics, Image Science and Vision* (1987), pp. 519–524.

[TH13] M. P. T. Ojala and D. Harwood. "A Comparative Study of Texture Measures with Classification based on Feature Distributions." In: *International Journal of Image Processing (IJIP), (Volume:7, Issue:2)* (2013), pp. 51–59.

[TJ05] V. P. T. Sawangsri and S. Jitapunkul. "Face Segmentation Based On Hue-Cr Components And Morphological Technique." In: *Proc. IEEE International Symposium on Circuits and Systems* (2005), pp. 5401–5404.

[TM02] M. P. T. Ojala and T. Mäenpää. "Multiresolution Gray-scale and Rotation Invariant Texture Classification with Local Binary Patterns." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence 24* (2002), pp. 971–987.

[TP06] A. H. Timo Ahonen and M. Pietikäinen. "Face Recognition with Local Binary Patterns." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume:28, Issue: 12)* (2006), pp. 2037–2041.

[TP91] M. Turk and A. Pentland. "Eigenfaces for Recognition." In: *Journal of Cognitive Neurosicence* (1991), pp. 71–86.

[TP92] M. A. Turk and A. P. Pentland. "Face Recognition Using Eigenfaces." In: *Vision and Modelling Group, The Media Laboratory Massachusetts Institute of Technology* (1992).

[Tri11] K. P. Tripathi. "A Study of Interactivity in Human Computer Interaction." In: *International Journal of Computer Applications (0975 – 8887) (Volume 16, No.6)* (2011).

[Wag] P. Wagner. *Principal Component Analysis and Linear Discriminant Analysis with GNU Octave*. `http://www.bytefish.de/blog/pca_lda_with_gnu_octave/`. Accessed: 2015-08-09.

[WL02] S. C. H. W. Wang Y. Gao and M. K. Leung. "A Fast And Robust Algorithm For Face Detection And Localization." In: *Proc. 9 th International Conference on Neural Information Processing* (2002), pp. 2118–2121.

[WW09] X. Y. W. Chen T. Sun and L. Wang. "Face Detection Based On Half Face-Template." In: *Proc. The Ninth International Conference on Electronic Measurement and Instruments* (2009).

[WY08] J. Wang and H. Yang. "Face Detection Based On Template Matching And 2DPCA Algorithm." In: *Proc. 2008 Congress on Image and Signal Processing* (2008), pp. 575–579.

[XW10] G. G. X. Liu and X. Wang. "Automatically Face Detection Based On BP Neural Network And Bayesian Decision." In: *Proc. 2010 Sixth International Conference on Natural Computation* (2010), pp. 1590–1594.

[YU94] Y. A. Y. Moses and S. Ullman. "Face Recognition: The Problem of Compensating for Changes in Illumination Direction." In: *European Conf. Computer Vision* (1994), pp. 286–296.