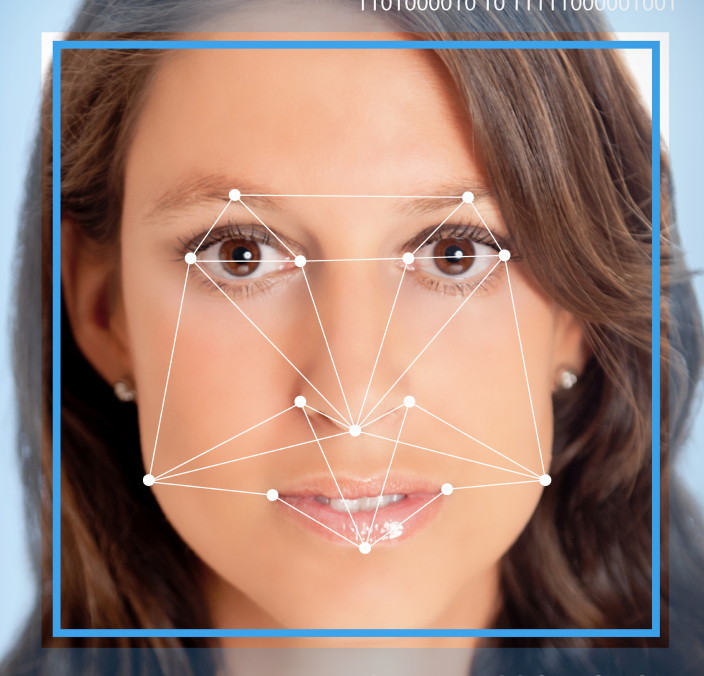
***Face recognition for medical data display***

*Techniques de programmation en C++*

14 /12/15

**Our Team:**

**Ines Frebault, Guillaume Laurent, Jonathan Couturier, Karim Mimouni**

Table des matières

[Introduction: 3](#_Toc437849723)

[1) Functional analysis 3](#_Toc437849724)

[2) OpenCV 3](#_Toc437849725)

[3) Architecture 3](#_Toc437849726)

[a) Layers 3](#_Toc437849727)

[b) Functions 4](#_Toc437849728)

[4) Business: Face recognition 5](#_Toc437849729)

[a) Principle 5](#_Toc437849730)

[b) Implementation 5](#_Toc437849731)

[c) Face Extraction 5](#_Toc437849732)

[d) Results 6](#_Toc437849733)

[5) BDD 7](#_Toc437849734)

[6) Interface User Machine 9](#_Toc437849735)

## Introduction:

As part of our engineering background to ISEP, the module "Techniques de programmation sous C++" allowed us to understand the computer language C ++ and the use of OpenCV which is a free library.

To do this, our group had to build a project relating to health, which enabled us to acquire and use the knowledge acquired during this module.

## Functional analysis

First we will analyze the project we choose by studying its specifications, identify all the concepts and introduce the technical solutions we’ve found.

In a second step we will explain these technical solutions and how we implemented them in our project.

Finally, we will conclude with the issues encountered, their solutions and the improvement points.

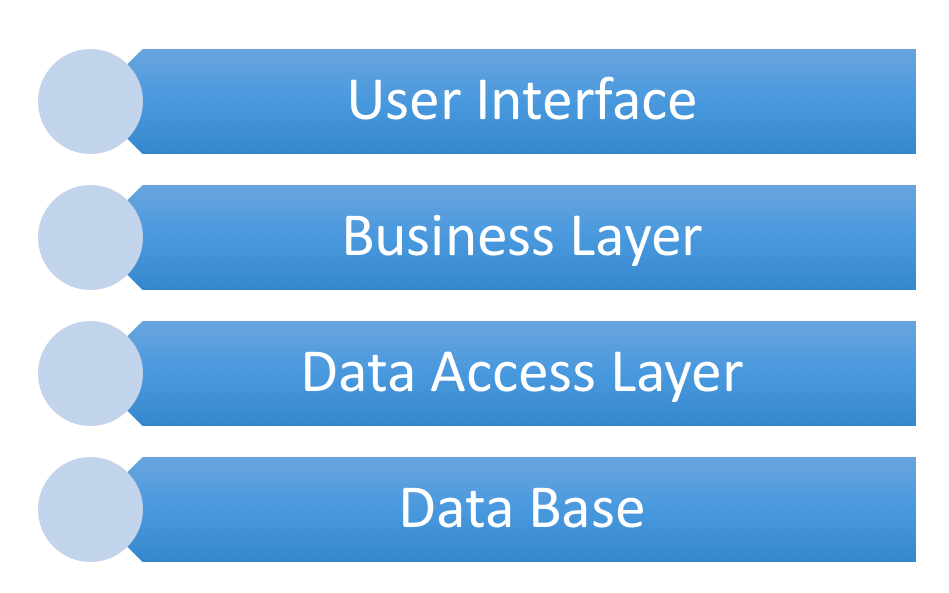
|  |  |  |
| --- | --- | --- |
| **Functionality :** | **Tools :** | **Developer :** |
| * Database | C++ | Guillaume |
| * Image processing and face recognition | C++ & openCV | Inès |
| * GUI | C++ & visual studio | Jonathan |
| * Interaction with the database | C++ | Karim |

## OpenCV

## Architecture

### Layers

The source code of our application will be organized according to the architecture we developed. Inspired by the MVC architecture, the GUI part will contain the management GUI; the Business will contain the image processing algorithms, and Com party will handle the communication with the database. To avoid crossing several times the same layers, the face recognition algorithm is executed at the Com layer. The following table lists the methods that we will implement:



### Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Nom** | **Entrée** | **Sortie** |
| GUI | getImage : Patient’s picture |  | Image |
| GUI | patientData : Display Data | Cluster |  |
| Business | faceDetection : detects the face and extracts the associated matrix | Image | Matrice |
| Com | getMatrix : create a request to export the matrix image from the base | id | Matrix, id |
| Com | matrixReader : Comparison algorithm | Matrice, Matrice, id | Booléen, id |
| Com | getDatas : create a request to export a patient’s datas | id | Cluster |
|  |  |  |  |

## Business: Face recognition

To recognise the patient on the input image, we chose the Eigen algorithm available in the “Face” contrib librairy of OpenCV : two other algorithm are also available but the Eigen algorithm was proven to be the best with our database.

### Principle

The Eigen algorithm use the Principal Component Analysis (PCA) algorithm. Given an n\*pmatrix, the PCA calculates pprincipal components to summerize most of the dataset variability. The first principal component is a subspace of the dataset that is representative of the direction in which the data vary the most; the second principal component is a linear combination of the variables uncorrelated with the first principal component and that have the largest variance, and the same principal applies for the other principal components. The principal components are names “eigenvectors” that are ordered by the eigenvalues; both are calculated as a linear combination with the Covariance Matrix.

The algorithm is then performed in two steps :

* the training samples and the input image are represented as a linear combination of  the eigenvectors
* the closest representation of the projection of the input image compared to the projections of the training samples is chosen as the result

### Implementation

Since the algorithm works with same-sized gray-scale images, we do the resizing in the PreTreatment file; we also have a histogram equalization option in the treatment() function to improve the results because the algorithm takes in account the illumination.

We will look a little bit more on the details of the face librairy :

The **FaceRecognizer** class is defined with **Algorithm** as a base class.

We have to start by creating the recognizer with the method **createEigenFaceRecognizer()**, where we can choose a number of eigenvectors to use (all in our case) and a confidence prediction value (no threshold in our case).

We used the **train()** function from the class FaceRecognizer where we specify the images and the corresponding labels to train the algorithm, as well as **predict()** where we specifiy the image to perform the algorithm on, the reference of its predicted label value and the reference of the confidence we want on the prediction.

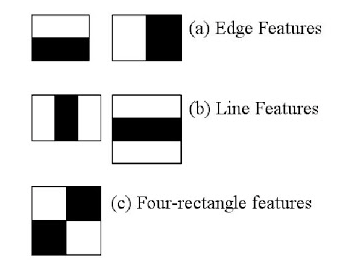
As a result, we get the corresponding label of the image recognised as the closest of our input image, or -1 if no image was identified as our patient.

There are a lot of functions we could have used in the class FaceRecognizer, like the function **save()** to save the state of our recognizer after the training, but we didn’t have the time to do so.

### Face Extraction

To improve the results, we decided to proceed on a face extraction before we launch our EigenFaceRecognizer to be more precise and avoid perturbations that could be on the picture.

The method we are using is the haar cascade classifier. The aim is to apply features  (like the one represented under) at each point of the image and to compute the sum of the pixels in the white areas and in the the black areas to define some rules to recognise the object.



In order to reduce computation time, the image is divided in regions and the features are grouped; if the region is defined as the object we want with the first group of features, we apply stricter groups features to avoid fake detections; if not, we skip the whole region.

In our case, our classifier has 38 groups of features: it is the CascadeClassifier that is contained in a .xlm file.

We have the **load()** method to use it, as well as **detectMultiScale()** method, that takes as input a reference to the image, a reference of the vector of objects detected from the image, the scale factor (how much the image is reduced at each iteration), the minimum number of neighbors, the value of the flags (not used), the min size of the object and the max size of the object.

This algorithm can detect as many faces in a picture as we want, but in our case we know there is only one face per picture so we return the last face found in the picture in our FaceExtraction class.

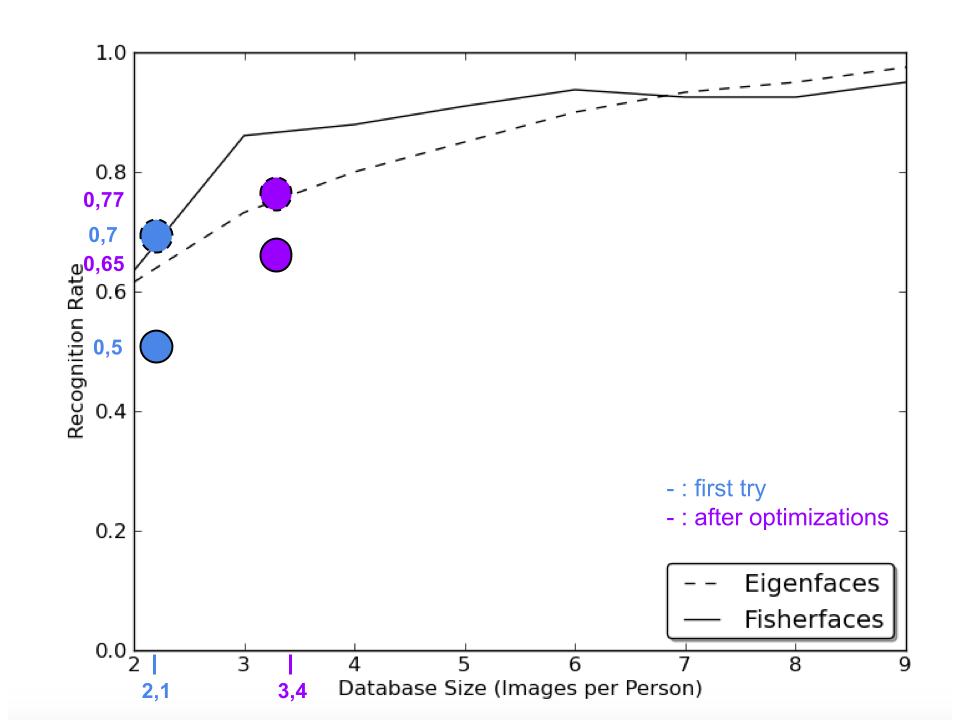
### Results

After an optimization phase, we have good results on the face recognition algorithm. The Face Extraction works perfectly well, with a 100% of face isolated from the pictures in the base. The scores for the recognition algorithm are showed on the next chart :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Eigenfaces | Fisherfaces | LBHF |
| First try : | 0,75 | 0,5 | 0,5 |
| After optimisations : | 0,764705882 | Not tested | 0,647058824 |

This chart shows the recognition rate we obtain with our picture database, composed of some pictures of classmates. Optimisations include face extraction, treatment() function and threshold.

We’ve found some theorical results for the recognition rate according to the number of picture per person, so we added our results on the graph :



We can see that our results are a little bit under what we should obtain, especially with the Fisherfaces algorithm.

The final results are as followed for a base where there is at least two picture of each person, for a total of 5 different persons :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Recognition | No Recognition | Errors |
| Final rates : | 0,764705882 | 0,058823529 | 0,176470588 |

“No Recognition” is when the picture of someone is too far from the other pictures of himself;  “Errors” is when someone is mistaken with someone else. We improved the error rate by putting a threshold confidence empirically, we could optimisate the system by changing it.

## BDD

Our database will consist of three tables:

• the main table: it has the primary key id which is incremented for each patient, it includes the personal information of each patient

• disease table: it has the primary key id and a foreign key id patient; this division allows a patient to have between 0 and n diseases.

• picture table: it has primary key id of each line and for the foreign key id also patient; So it allows each patient to have between 0 and n associated photos, which can be useful if you chose an algorithm that requires a learning phase.

Main table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | nom | prénom | sexe | date de naissance | n° sécu | adresse | n tel |
| int | texte | texte | char | MMDDYY | string | texte | string |

Table maladie :

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | id patient | maladie | traitement |
| int | int | texte | texte |

Table image :

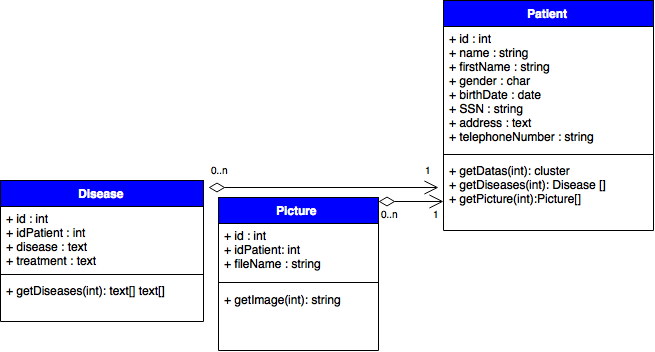
|  |  |  |
| --- | --- | --- |
| **id** | id patient | nom fichier |
| int | int | texte |

**objects:**

**• Patient: aggregates information on a patient; class creates in face recognition**

**• Processing: includes the processes associated with a patient**

**• Image: includes the necessary information about an image**



## Interface User Machine

This section contains the HMI we realized an indication, the first screen is the display of patient data, with key information on blue background and medical history at the bottom of the page.

The second screen is the screen indicates the image to identify a patient from the base and then offers to display its information; the third screen indicates that no patient has been recognized from the image and then offers to choose another image.

