

**Silesian University of Technology**

**Faculty of Automatic Control, Electronics   
and Computer Science**

##### Final Project

##### (choose appropriate)

Implementation of face detection algorithms in video sequences

Author: Wojciech Mazur

Supervisor: PhD Michał Staniszewski

Consultant: PhD Name Surname

Gliwice, January 2019Załącznik Nr 2 do Zarz. Nr 97/08/09

**Oświadczenie**

Wyrażam zgodę/nie wyrażam\* zgody na udostępnienie mojej pracy dyplomowej/rozprawy doktorskiej\*

…………….………., dnia …………………………

……………………………………..……………….……

*(podpis)*

……………………………………………………………

*(poświadczenie wiarygodności podpisu przez Dziekanat)*

*\* właściwe podkreślić*

**Oświadczenie promotora**

Oświadczam, że praca „Implementation of face detection algorithms in video sequences” spełnia wymagania formalne pracy dyplomowej inżynierskiej.

|  |  |
| --- | --- |
| Gliwice, dnia ……………………… | ………………..……………….……  *(podpis)* |

Contents

[1. Introduction 1](#_Toc535111802)

[1.1. Goal of the thesis 2](#_Toc535111803)

[1.2. Scope of work 2](#_Toc535111804)

[1.3. Description of chapters 3](#_Toc535111805)

[2. Problem analysis 4](#_Toc535111806)

[2.1. Face detection methods 4](#_Toc535111807)

[2.2. Face recognition methods 7](#_Toc535111808)

[2.2.1. Holistic approaches 8](#_Toc535111809)

[2.2.2. Hybrid approaches 9](#_Toc535111810)

[2.2.3. Feature-based 9](#_Toc535111811)

[2.2.4. Soft computing methods 10](#_Toc535111812)

[3. Requirements and tools 11](#_Toc535111813)

[3.1. Requirements 12](#_Toc535111814)

[3.1.1. Functional requirements 12](#_Toc535111815)

[3.1.2. Non-functional requirements 13](#_Toc535111816)

[3.2. Tools 13](#_Toc535111817)

[3.2.1. Programming language 13](#_Toc535111818)

[3.2.2. Libraries 15](#_Toc535111819)

[3.2.3. Methodology of design and implementation 17](#_Toc535111820)

[3.3. Use cases 17](#_Toc535111821)

[4. External specification 19](#_Toc535111822)

[4.1. Software requirements 19](#_Toc535111823)

[4.2. Installation process 19](#_Toc535111824)

[4.3. Example of usage 24](#_Toc535111825)

[4.4. Processing video sequence 25](#_Toc535111826)

[5. Internal specification 26](#_Toc535111827)

[5.1. Application implementation details 26](#_Toc535111828)

[5.1.1. Face image processing 26](#_Toc535111829)

[5.1.2. Training recognizer 30](#_Toc535111830)

[5.1.3. Video processing 31](#_Toc535111831)

[6. Verification and validation 32](#_Toc535111832)

[6.1. Verification procedure 32](#_Toc535111833)

[6.2. Verification results 33](#_Toc535111834)

[7. Conclusions 35](#_Toc535111835)

[Bibliography i](#_Toc535111836)

[List of abbreviations and symbols iv](#_Toc535111837)

[Contents of attached CD-ROM v](#_Toc535111838)

[List of Figures vi](#_Toc535111839)

[List of Tables vii](#_Toc535111840)

# Introduction

According to Maslow’s Hierarchy of needs, safety is one of our most fundamental needs. Without it, it is hard to think about friends, relationships, accomplishments, or self-fulfillment. Over the last few years it can be observed, that the biological identification is rising in its popularity. There are new ways found to utilize human biological footprints. What can be noticed, is that most of the use cases revolves around security, and for a good reason. Fingerprints are used to unlock locks for decades. Fingerprint scanners have also found a way to consumer electronics. World with securing our smartphone or a notebook with just a password can no longer be imagined. Fingerprint scanners have expended rapidly when it comes to smartphones and made our lives better.

But it’s not the only way people are trying to secure our privacy, including our increasingly valuable resource, which is data. There are ways developed to recognize people by other features like iris, or even from a face in general, using different kinds of sensors. Cameras, infrared light sensors to depth detectors that remember given points on a face. And everything happens within a blink of an eye.

What people want to do apart from protecting our data, is something that humanity crave to do since beginning of itself. To protect ourselves from external harm. Feel safe and comfortable, which is, in fact, required, as stated previously.

One of the main issues arising in recent years, is public safety and detection of dangerous people. Increasing activity of terrorism in a lot of European countries can be observed. The awareness of people is also increasing, it is being learned that prevention is better that treatment. National safety agencies recognize most of European countries at least on moderate threat of terrorism. A lot of countries are considered even as a high threat for regular travelers. This is something that needs to be addressed as quickly as possible, and it is needed to work on the technology, that can be effectively applicated for new solutions.

## Goal of the thesis

Main goal of this thesis is to implement face detection algorithms, both with the face recognition. This could provide us with useful tools to analyze and examine, for example video footages that are collected every day on thousands of city cameras around the globe. That existing architecture can be used, not only to look for people that have already broke the law, but also to prevent crime. Simple camera system doesn’t provide us with the possibility of detecting the face. It would be needed to do that manually which is a complicated and tedious work, that not every person is able to do. The main focus is to accomplish filtering dangerous, or at least for some reason crucial for safety people, implementing algorithms that detect and recognize their faces.

## Scope of work

Main scope of the work contains a few elements that needed to be combined to give full spectrum of the problem. First of all, I needed to choose programming language, that would be easy to use, yet contain all the elements needed to process images, like easily accessible and usable libraries that would greatly accelerate the progress of work. Then, analysis of current solutions, and choice of a few algorithms that comply with what this thesis is trying to achieve and comparison of their effectiveness. Next thing was the actual implementation of the solution in the programming language of choice, but also processing the testing material consisting of video sequences that were analyzed and graphically modified that the human can easily observe effects of the detection and recognition in real time.

## Description of chapters

Paper contains a few chapters that enclose the whole process of the thesis. First chapter is an introduction with goal of the thesis specified and scope of work described.

Second chapter focuses on the problem analysis, with face detection and recognition methods that were researched and their brief description.

Next chapter describes the requirements and tools that were specified for the project, research that was made, and argumentation of decisions made regarding technologies used.

Fourth chapter describes the software requirements that need to be fulfilled to use the project, and technical process of installing required tools.

In the fifth chapter there is a more in-depth description of the algorithms used and code written with explanations why given rules were applied.

Sixth chapter characterize the verification and validation of the project and what needed to be implemented, to measure accuracy of predictions.

Last chapter enclose the summary of the implementation process of the project and condensed description of problems and further work.

# Problem analysis

Problem analysis in this chapter revolves around the most important aspects of face detection and recognition methods. It introduces a few most popular methods both with a concise description of each and describes the results of the research that was performed.

## Face detection methods

There are a lot of ways to detect given features, including face features, that would allow us to recognize position of the face on a given media sample. Although one that got very popular since its release is based on machine learning approach, from the work of P. Viola and M. Jones [1] [2]. Their work is described as extremely rapid in terms of effectiveness, and this is something that is crucial for effective and real time face recognition. The original study was performed on Intel Pentium III clocked on 700MHz, which is a rather old processing unit considering performance of newest units. This would give over 15fps on 384x288 pixel images. Similar CPUs will be compared using tool UserBenchmark [3]. Comparing closest processor that this tool provides us, which is a successor to Pentium III, Pentium 4. Its performance can be taken as comparable, especially in units that were on similar clock speeds [4]. Unit that was used for calculations in this thesis was Intel Core i5 4690k, which is according to UserBenchmark, over ten times faster on average, often going up to twenty times faster and above [3]. This gives us the opportunity to take higher resolution of video samples and photos, and still achieve a very satisfying frames per second rate.

Idea is that if given element, item or other object has a unique shape, it can be assumed, that it also has unique properties when it comes to light that is projected onto that object. This gives us a tool to check whether chosen parts of an image have lighting that is changing in a way that was previously seen and is recognize for example as a face.

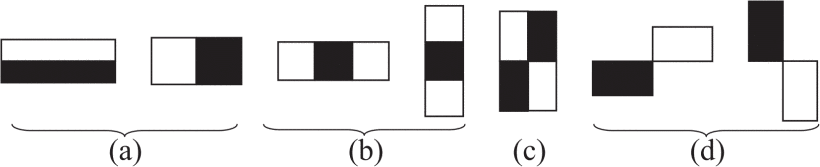


Figure 1Basic Haar-like features [1]

On the Figure 1 is can be seen what features are used to distinguish between light and dark spots on a picture. Although this can be misleading, what should be really be seen if understanding of the problem is a priority, is to see the image in a way shown in Figure 2.



Figure 2Haar-like feature on face [5]

The colors itself are not important, but the pixel values within. For example, if face is to be found, it can be safely assumed, that forehead pixels will be on average brighter than pixels that are within eyes region, due to the shadow dropping from the eyebrows. The same goes mostly for things like nose, where either vertical line, that is bright and has darker surroundings due to the shadows. But this is also something that needs to be taken with care, due to the changing light conditions. Compensation can be done, by looking for white line that has a darker region only on one side.

There were taken many approaches to improve initial Viola and Jones, including the Viola himself. One of the first successful improvements were made only one year later, by the R. Lienhart and J. Maydt [6]. They were dealing with two main problems of original approach, by adding 45 rotated features and adding new optimization procedures for improved performance.

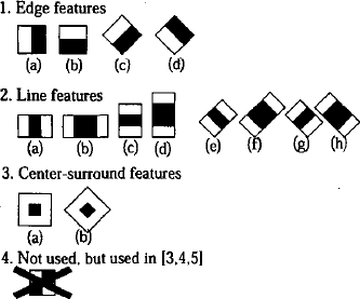


Figure 3Extended features proposed by Lienhart and Maydt [6]

Further attempts of improvements can be observed. T. Mita, T. Kaneko and O. Hori proposed and derived method of detecting co-occurrence of features. One of the things that are addressed by those authors, is that in original solution, after detecting one feature, detecting of proceeding ones comes with much higher error rate [7].



Figure 4 Joint Haar-Like feature

This approach allows to capture more of a structural similarity of faces. Experiment results give reduced error rates and improved performance, even when considering bigger number of features, than in Viola and Jones solution.

## Face recognition methods

Face recognition is a topic that is highly researched for decades, first by manual labor, where for example physical photos were projected onto photomultiplier matrix, then small motors were turned according to illumination [8], which is in fact a machine learning approach. Since then, great amounts of methods were developed to recognize patterns, that includes faces [9]. Numerous approaches that are holistic, hybrid, feature-based, artificial neural networks, fuzzy-based, generic-algorithms based and more can be found [9], where variance in implementation can mixing different approaches can also differ. Some ways of recognition would estimate face position, for instance, to be able to transform image in given space, to obtain more accurate results.

It is also required, for a reasonable efficiency to optimize and normalize dataset from which the algorithm will be built upon. The significant matter is to provide good quality images, for face recognition straight face is a suggested look, without rotation and with the same scale as the rest of the photos, also with an equal lighting [10]. But it is not always possible to provide such resources. Good practice would be to implement image processing that equalize the appearance of the given object.

Next general task in facial recognition is to extract features [9]. Dimensionality of the image is needed to be reduced, so that the data that needs to be processed is not that enormous, and yet the important features are conserved.

### Holistic approaches

Holistic ways of recognizing faces are dealing with the problem with comprehensive approach to processing facial verification, taking data from image as a whole block. They consider small number of features and try to envelop components of an image. Later, these components are used to verify similar shapes on other data [11].

#### Eigenfaces

When considering sets of images, it can be seen that great amount of data can easily arise. Reasonable and natural step would be to reduce its size. The advantage of this approach is that big dataset is not needed, although the bigger is usually the better. Eigenface is a method that is extracting features into vectors, and then represent then in a form of covariance matrix. Having those vectors, it can be calculated how distant they are [12] [11].

However, in this approach training data set is required to be processed, otherwise results will be not satisfactory. It is needed to equalize lighting, and align image, it is also preferred to remove background and other noise that could have negative influence. This algorithm is using PCA (Principal Component Analysis) to reduce dimensions and find vectors. Vector defines subspace – face space, and training set is projected, to find weights, or similarities in other words.

#### Fisherfaces

Fisherfaces are similar to Eigenfaces. Method uses both principal component analysis and linear discriminant analysis and produce a subspace projection matrix. It is minimizing variation within classes and maximizing separation. It should provide better results when it comes to different facial expressions and light variance. Although computation time is more complex, and time needed is greater [13].

### Hybrid approaches

These methods are based on mixing holistic approach and feature-based matching. These two methods have their advantages and disadvantages, but the idea is that combination of those two might give a better result and eliminate each other disadvantages [14] [15].

A lot of hybrid approaches draw from the Gabor wavelets that were acknowledged as a reliable local feature extraction method, due to its sturdiness when it comes to light, distortions, translation [15].

### Feature-based

Every face has some distinctive features related to some regions that can be extracted. Eyes, nose, mouth, cheeks can be used to classify a face. Complexity of these solutions are often hidden under recognizing particular parts of the face. For example, eye processing could be interrupted by reflexes on the iris, that should be eliminated, eyebrows are often described as a parabola approximation. Nose is one of the simplest properties that can be used as a base, where gray levels contrasts are great within neighbor regions. Mouth shape can be also described as a function. Knowing that many facial points, chin location can be also estimated. Approach like this was described in [16] where efficiency was often exceeding 90 percent.

However, this is a limited approach when it comes to providing information about face details. Facial features are rich in texture, but not enough to tell them apart from background. This problem is being approached by adding context information of each feature. But this is a tough task, especially when within- class variance is big.

### Soft computing methods

Face recognition comes with a lot of variance, imprecision, uncertainty and approximations. Soft computing methods are addressing those problems and tries to solve them. These approaches contain methods like fuzzy logic, artificial neural networks, machine learning, generic algorithms [17].

#### Artificial neural networks

Inspired by the biological nervous system and the way it works. This solution aims to solve non-linear problems. Artificial neural networks are in fact an interconnected web of so-called neurons. Each of the neurons performs little operations that adjusts weights, so that desired output is shown. Training is often accomplished by feeding the neural network with patterns

#### Fuzzy logic

This is an approach mimicking human knowledge, that is naturally imprecise. It introduces concept of partial truth and false. That is since most of the human body properties are nonlinear and trimming them down to linear solutions makes them often impossible to achieve high accuracy [18]. This method is often incorporated as a part of recognition process with other methods. For example, fuzzy k-nearest neighbor classification to find suitable scatter matrices.

#### Genetic algorithms

Genetic algorithm approaches derive its ide from natural evolution. It is based on inheritance, mutation, natural selection, recombination, where goal is to select fittest solution. It is randomized, but also directed.

Some terms are needed for further explanation to understand basics of those solutions. Chromosomes are set of genes, and gene contains part of the solution. Fitness describes closeness to solution, and its function assign fitness value to individual. Mutation is a change in random gene, crossover is an operation to create new chromosomes for offspring, the better they are, the better chance of “surviving” they have [19].

Solutions can be found where GA solutions are mixed with other approaches, for example systems where principal component analysis is used for feature extraction and Genetic Algorithm to recognize [19].

# Requirements and tools

Clear requirements description is a good indicator of a direction of a work. This chapter contains a few functional and non-functional requirements that will define further work on the thesis. It also describes the tools available in the market that would allow to execute the requirements, and the arguments why given tools were used.

## Requirements

Software requirements are a basis that specify what should be included in a project. Early definition allows to easily avoid redesigns in further stages of the development.

### Functional requirements

Basic database of photos should be easy to provide by the user to the given folder. Every photo needs to contain one face, preferably with good lighting. Phots should be grouped in folders, one folder to one person that detection will be take place on. Folders should be named accordingly, uniquely and describe the person, as this name will be used to recognize given face. Every folder should contain the same number of photos, to allow the training process to achieve best performance.

Aligned and processed images should be placed in a separate folder, with the same labels, segregated in folders the same way as before the processing.

The application should provide possibility to specify video sequence that will be further processed as a string of characters.

Face on a processed video media should be tracked and clearly marked with a square shaped box. Above the square should be a text with the prediction result, that is name of the face owner. Text should be readable both when the face detected is far, so the square and text above is small, and when the face is close, and the text could be too big to fit into the screen.

Output video after face detection should be placed in separate folder marked as the output videos, with each video named the same way as before processing with “output” appended to the original title to be easily distinguished.

### Non-functional requirements

Application should be compatible with different sizes and qualities of videos.

Processed images should be loaded and exported to extensions of jpeg format.

Output photos and videos should be marked with “output” word.

Recognized faces should be marked green, wrongly recognized red, and false positive detections white on output video.

## Tools

Before proceeding to the work, important part of the thesis was research about what tools are available, that would allow to finish the thesis goal efficiently and with a good result. Below, the research is presented with its results, tools of choice and why they were used.

### Programming language

#### Research results

In data processing, especially in image processing there are a few most viable options when it comes to programming languages. There are basically four different options available.

First option is Java language that comes with two libraries that would help with writing this thesis. Wrapper for OpenCV and native JavaCV. However, due to the lack of experience and personal preferences, choice was made not to use Java for development.

MATLAB is a really powerful tool and can be successfully used for tasks like image processing. There are toolboxes available, that further extend possibilities, for example image processing toolbox, computer vision toolbox and many more. Also, the program allows for easy access to data, and easy visualization, with simple methods. MATLAB also supports OpenCV which is a next huge advantage. It also comes with a great documentation. However, cost of the software is enormous for an ordinary user, that completely brought chances of using this tool to zero percent [20].

More of a low-level approach would be to use C++, it comes with a great support of OpenCV and great performance. Although, it comes with responsibility of managing memory.

#### Language of choice

Language of choice for this project was Python. It is widely used in data science field, due to the huge number of libraries and supportive community. There are a few advantages that decided for this language to be used.

The cleanness of language – this was the main purpose while Python was designed. To keep it simple, consistent and compact, which is already a distinctive feature in a programming world. Code is easy to write and easy to read, and this makes it really great for code maintenance. Yet, it was designed to fully support Object Oriented Programming, imperative, procedural and functional programming.

Efficiency of prototyping – This is a dynamic type language, which means, that there are no strict types defined. Of course, they are assigned later, during interpreting, but a programmer doesn’t have to think about that while creating the code. Python code is often much shorter, which further extends ease of upkeeping clean code.

Portability – Python code is multiplatform for most part. It can be easily executed on many different platforms. It allows not only to make graphical user interface applications, but also web applications. This makes it a universal tool for many applications.

Libraries – Python offers great number of libraries that makes it really easy to develop applications. This includes web development like Django or Flask, calculations like NumPy, game development like Pygame and many more.

Quality of code – On the base of Python stands so-called Zen of Python (PEP 20), that includes aphorisms like “Beautiful is better than ugly” or “complex is better than complicated” or “readability counts”. It has also a lot of other PEP rules defining style convention of Python like PEP 8, where it is defined how many spaces for indentation level, maximum line length or occurrence of white spaces. Even though there are a lot of rules, most of IDE (programming environments) give possibility to install extensions that keep track of those rules and often even automatically correct them. This makes Python code very universal and easy to read among developers.

### Libraries

#### OpenCV

It’s an open source, free for commercial and educational use library. It was written in C and C++ and is developed using wrappers in a lot of programming languages like C#, Python, MATLAB, Java. One of the great advantages of this library possible multiplatform usage. It’s compatible with Windows, Mac OS X, Linux, Android and iOS.

Library was created to serve in real-time applications, where computing efficiency is a main issue. C and C++ as a relatively low-level language gives possibility to optimize and use multicore processor units.

One of the main goals of OpenCV is to provide an all in one tool that would allow to easily create advanced projects by people on different knowledge levels. It can be easily use both for educating beginners, but it also gives the professionals to use advanced tools without reinventing the wheel.

#### NumPy

NumPy package is a basic Python tool that allows for advanced mathematical calculations, mainly for scientific purposes like numerical methods, operations on matrixes, diagonalization, integration, solving of equations and so on.

One of the basic objects that is introduced by NumPy is ndarray. It can be treated as a universal container for data in form of vectors or arrays. The main difference between arrays provided by ndarray and default Python arrays is that objects inside needs to be the same type, they keep their size, while changing size new object is being created, and old is removed, objects contain functions to operate on contained data, optimized to use on big datasets.

#### Matplotlib

Powerful and rich library used for plotting and data visualization. It produces 2D graphics in interactive environments. It allows to draw graphs like lines, scattered, bar and many more.

Matplotlib has Pyplot module that provide MATLAB-like interface. Which makes it easy for developers moving from that environment to begin. It also has object-oriented API to easily embed plotting into graphical interfaces like Qt or Tkinter.

It can be also extended to use with Microsoft Excel, or to create 3D plots by using mplot3d. Active group of developers comes also with great profit with constant support and big community.

#### Scikit-learn

Package that provides simple and efficient tools for data mining and analysis. Includes various classification, regression, clustering algorithms including support vector machines, random forests, gradient boosting, k-means and more. It is also built upon and designed to cooperate with other scientific libraries like NumPy and SciPy.

### Methodology of design and implementation

The main design consists of three parts of the application that needs to be executed sequentially to achieve desired output starting from the non-processed database of photos.

To allow clear architecture of the project, separation of tasks in the design of application was applied. Every part has completely different responsibility and whole program is in a pipe form, but also allows to execute different parts separately, if user, for example wants only to process images

.

## Use cases

Due to the simple nature of the program, and very limited interaction with the user, the UML diagram is very simple.

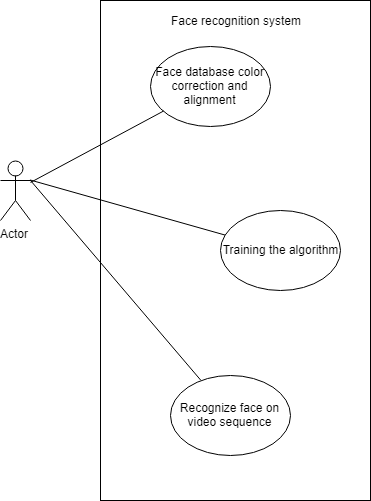


Figure 5Use cases UML

# External specification

External specification of application describes the software requirements, what programs needs to be installed, and how to correctly perform the installation. There is also a process of executing the files described.

## Software requirements

The application was implemented using Python language, which means that it should be compatible with many different platforms, like Mac OS X, Linux or Windows, where procedure should consist only of copy and pasting the code. However, apart from code, system is required to have installed Python. Preferable way to install the Python is to install it by installing Anaconda package. Anaconda is a free and open-source distribution for Python and R for scientific computing. Installation of this package would simplify the whole process of setting up the environment needed to run the program. It is also cross-platform.

## Installation process

Application described is a concept, that could be incorporated into bigger system, but not a standalone application per se, which means that preferable use case is for developers with development environment.

First requirement is an Anaconda distribution. It can be easily downloaded on anaconda.com/download.

On the Table 1 list of needed packaged can be seen, although their number is great, most of them come with the Anaconda package, and the installation process is not that complicated.

Table 1 List of libraries needed to run the program

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Version** | **Build** | **Channel** |
| astroid | 2.1.0 | py36\_1000 | conda-forge |
| basemap | 1.2.0 | py36h4e5d7af\_0 | anaconda |
| blas | 1.0 | mkl |  |
| ca-certificates | 2018.03.07 | 0 | anaconda |
| certifi | 2018.10.15 | py36\_0 | anaconda |
| cloudpickle | 0.6.1 | py36\_0 | anaconda |
| cmake | 3.13.2.post1 | <pip> |  |
| colorama | 0.4.0 | py\_0 | conda-forge |
| cycler | 0.10.0 | py\_1 | conda-forge |
| Cython | 0.29 | <pip> |  |
| dask-core | 1.0.0 | py36\_0 | anaconda |
| decorator | 4.3.0 | py36\_0 | anaconda |
| dlib | 19.9 | np111py36\_0 | conda-forge |
| face-recognition | 1.2.3 | <pip> |  |
| face\_recognition\_models | 0.3.0 | pyh0cf5a0c\_0 | akode |
| freetype | 2.9.1 | he8b6a0d\_1004 | conda-forge |
| geos | 3.6.2 | h9ef7328\_2 | anaconda |
| hdf5 | 1.8.20 | hac2f561\_1 |  |
| icc\_rt | 2017.0.4 | h97af966\_0 |  |
| icu | 58.2 | ha66f8fd\_1 |  |
| imageio | 2.4.1 | py36\_0 | anaconda |
| imutils | 0.5.2 | <pip> |  |
| intel-openmp | 2019.0 | 118 |  |
| isort | 4.3.4 | py36\_1000 | conda-forge |
| jpeg | 9c | hfa6e2cd\_1001 | conda-forge |
| kiwisolver | 1.0.1 | py36he980bc4\_1002 | conda-forge |
| lazy-object-proxy | 1.3.1 | py36hfa6e2cd\_1000 | conda-forge |
| libopencv | 3.4.2 | h20b85fd\_0 |  |
| libpng | 1.6.34 | h7602738\_2 | conda-forge |
| libtiff | 4.0.9 | h36446d0\_1002 | conda-forge |
| libwebp | 0.5.2 | 7 | conda-forge |
| matplotlib | 2.2.3 | py36h31860fd\_0 | conda-forge |
| mccabe | 0.6.1 | py\_1 | conda-forge |
| mkl | 2018.0.3 | 1 |  |
| mkl\_fft | 1.0.10 | py36\_0 | conda-forge |
| mkl\_random | 1.0.2 | py36\_0 | conda-forge |
| networkx | 2.2 | py36\_1 | anaconda |
| numpy | 1.11.3 | py36h4a99626\_4 |  |
| numpy-base | 1.15.4 | py36h8128ebf\_0 |  |
| olefile | 0.46 | py36\_0 | anaconda |
| opencv | 3.4.3 | py36h597e314\_201 | conda-forge |
| opencv-contrib-python | 3.4.3.18 | <pip> |  |
| openssl | 1.0.2p | hfa6e2cd\_0 | anaconda |
| pillow | 5.3.0 | py36hdc69c19\_0 | anaconda |
| Pillow | 5.3.0 | <pip> |  |
| pip | 18.1 | py36\_1000 | conda-forge |
| pip | 18.1 | <pip> |  |
| proj4 | 5.1.0 | hfa6e2cd\_1 | anaconda |
| py-opencv | 3.4.2 | py36hc319ecb\_0 |  |
| pylint | 2.2.2 | py36\_1000 | conda-forge |
| pyparsing | 2.3.0 | py\_0 | conda-forge |
| pyproj | 1.9.5.1 | py36hb98d9bb\_1 | anaconda |
| pyqt | 5.6.0 | py36h764d66f\_1007 | conda-forge |
| pyshp | 1.2.12 | py36\_0 | anaconda |
| python | 3.6.6 | he025d50\_0 | conda-forge |
| python-dateutil | 2.7.5 | py\_0 | conda-forge |
| pytz | 2018.7 | py\_0 | conda-forge |
| pywavelets | 1.0.1 | py36h8c2d366\_0 | anaconda |
| qt | 5.6.2 | h2639256\_8 | conda-forge |
| scikit-image | 0.14.0 | py36h6538335\_1 | anaconda |
| scikit-learn | 0.20.1 | py36hb854c30\_0 | anaconda |
| scipy | 1.1.0 | py36hc28095f\_0 |  |
| setuptools | 40.6.2 | py36\_0 | conda-forge |
| sip | 4.18.1 | py36h6538335\_0 | conda-forge |
| six | 1.11.0 | py36\_1001 | conda-forge |
| sqlite | 3.26.0 | hfa6e2cd\_1000 | conda-forge |
| tk | 8.6.8 | hfa6e2cd\_0 | anaconda |
| toolz | 0.9.0 | py36\_0 | anaconda |
| tornado | 5.1.1 | py36hfa6e2cd\_1000 | conda-forge |
| typed-ast | 1.1.0 | py36hfa6e2cd\_1000 | conda-forge |
| vc | 14.1 | h21ff451\_3 | anaconda |
| vs2015\_runtime | 15.5.2 | 3 | anaconda |
| wheel | 0.32.3 | py36\_0 | conda-forge |
| wincertstore | 0.2 | py36\_1002 | conda-forge |
| wrapt | 1.10.11 | py36hfa6e2cd\_1001 | conda-forge |
| zlib | 1.2.11 | h2fa13f4\_1003 | conda-forge |

Should be noted, that although every package can be installed manually, following the above list could be time consuming or confusing for beginner user. Therefore, list of requirements can be generated by the command, with Anaconda previously installed conda create --name <envname> --file requirements.txt. Using this file, procedure is very simple, below there are contents of requirements.txt file, together with the instruction of installation process, placed on the first three lines.

To install all the dependencies at once, following command should be executed with given requirements file conda create --name <envname> --file requirements.txt. Where <envname> is a name that environment will be given.

Table 2 requirements.txt file

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72 | # This file may be used to create an environment using:  # $ conda create --name <env> --file <this file>  # platform: win-64  astroid=**2.1**.**0**=py36\_1000  basemap=**1.2**.**0**=py36h4e5d7af\_0  blas=**1.0**=mkl  ca-certificates=**2018.03**.**07**=**0**  certifi=**2018.10**.**15**=py36\_0  cloudpickle=**0.6**.**1**=py36\_0  colorama=**0.4**.**0**=py\_0  cycler=**0.10**.**0**=py\_1  dask-core=**1.0**.**0**=py36\_0  decorator=**4.3**.**0**=py36\_0  dlib=**19.9**=np111py36\_0  face\_recognition\_models=**0.3**.**0**=pyh0cf5a0c\_0  freetype=**2.9**.**1**=he8b6a0d\_1004  geos=**3.6**.**2**=h9ef7328\_2  hdf5=**1.8**.**20**=hac2f561\_1  icc\_rt=**2017.0**.**4**=h97af966\_0  icu=**58.2**=ha66f8fd\_1  imageio=**2.4**.**1**=py36\_0  intel-openmp=**2019.0**=**118**  isort=**4.3**.**4**=py36\_1000  jpeg=**9**c=hfa6e2cd\_1001  kiwisolver=**1.0**.**1**=py36he980bc4\_1002  lazy-object-proxy=**1.3**.**1**=py36hfa6e2cd\_1000  libopencv=**3.4**.**2**=h20b85fd\_0  libpng=**1.6**.**34**=h7602738\_2  libtiff=**4.0**.**9**=h36446d0\_1002  libwebp=**0.5**.**2**=**7**  matplotlib=**2.2**.**3**=py36h31860fd\_0  mccabe=**0.6**.**1**=py\_1  mkl=**2018.0**.**3**=**1**  mkl\_fft=**1.0**.**10**=py36\_0  mkl\_random=**1.0**.**2**=py36\_0  networkx=**2.2**=py36\_1  numpy=**1.11**.**3**=py36h4a99626\_4  numpy-base=**1.15**.**4**=py36h8128ebf\_0  olefile=**0.46**=py36\_0  opencv=**3.4**.**3**=py36h597e314\_201  openssl=**1.0**.**2**p=hfa6e2cd\_0  pillow=**5.3**.**0**=py36hdc69c19\_0  pip=**18.1**=py36\_1000  proj4=**5.1**.**0**=hfa6e2cd\_1  py-opencv=**3.4**.**2**=py36hc319ecb\_0  pylint=**2.2**.**2**=py36\_1000  pyparsing=**2.3**.**0**=py\_0  pyproj=**1.9**.**5.1**=py36hb98d9bb\_1  pyqt=**5.6**.**0**=py36h764d66f\_1007  pyshp=**1.2**.**12**=py36\_0  python=**3.6**.**6**=he025d50\_0  python-dateutil=**2.7**.**5**=py\_0  pytz=**2018.7**=py\_0  pywavelets=**1.0**.**1**=py36h8c2d366\_0  qt=**5.6**.**2**=h2639256\_8  scikit-image=**0.14**.**0**=py36h6538335\_1  scikit-learn=**0.20**.**1**=py36hb854c30\_0  scipy=**1.1**.**0**=py36hc28095f\_0  setuptools=**40.6**.**2**=py36\_0  sip=**4.18**.**1**=py36h6538335\_0  six=**1.11**.**0**=py36\_1001  sqlite=**3.26**.**0**=hfa6e2cd\_1000  tk=**8.6**.**8**=hfa6e2cd\_0  toolz=**0.9**.**0**=py36\_0  tornado=**5.1**.**1**=py36hfa6e2cd\_1000  typed-ast=**1.1**.**0**=py36hfa6e2cd\_1000  vc=**14.1**=h21ff451\_3  vs2015\_runtime=**15.5**.**2**=**3**  wheel=**0.32**.**3**=py36\_0  wincertstore=**0.2**=py36\_1002  wrapt=**1.10**.**11**=py36hfa6e2cd\_1001  zlib=**1.2**.**11**=h2fa13f4\_1003 |

After installing above listed software, user should test if the environment is correctly prepared, by conda list command, and check if the list of installed packages is being shown.

If list shown is matching list above, it means that the setup is probably correct. User should also assure that the virtual environment is visible in the system, it can be done by conda info –envs command. There should be environment listed, that was created before, during conda create.

Then, user should activate environment by conda activate <envname> where <envname> is previously created environment name. After that command, in terminal before the actual path, in parenthesis envname should be visible, indicating that user is currently using chosen environment.

## Example of usage

Next command is python <file\_name> that is used to execute given python file. To cut and align images facescropper.py should be executed, next user should train the recognizer by using faces-train.py, and the last should be facesrec-video.py.

## Processing video sequence

After loading, cutting, aligning photos and training model, next file should be executed. Processing the input video will take proportional amount of time to video length and its bitrate.

On the output video sequence, if there is a face visible and correctly detected, square around that face should be visible. Inside this square, prediction is made, and its results are displayed above the square, with predicted name of the person and confidence of prediction.

|  |
| --- |
|  |
|  |

# Internal specification

This part is describing the internal workings of the system. The most important functionalities of code are explained and examples of processes taking place during scripts working are given.

## Application implementation details

There are three most important steps described as separate subtopics. Description of image processing, with examples, how the training process is accomplished, and further description of actions during last phase, which is video sequence processing.

### Face image processing

First file that user needs to run is a file responsible for cropping images of people, so that only the processed image of a face is saved. To detect the face first method that is used, is previously described haar-like cascades classifier, with proper xml file that will allow to quickly find location of frontal part of face. The loop is recursively walking though folders from image folders, reading labels, which is used to identify recognized person later, and put that on a video material above the face.



Figure 6Examplary input file

Haar-like cascades method works in greyscale, so the next step is image being changed to greyscale from blue green red scale, which is provided by the OpenCV library. Then, detector tries to find a face on a given image, where two arguments are being used, scaleFactor and minNeighbors. ScaleFactor parameter is specifying how much the image size is reduced at each image scale, minNeighbors specifies how many neighbors each candidate rectangle should have to retain it [21]. Those parameters often need to be checked manually, as there is no one given indicator of how well they will perform on a chosen material.

One of the anchors that can be used to align a face image, are eyes, and they were chosen in this thesis. Again, a haar-like classifier is used, but this time with a different xml file. To reduce possibility of false positive, eyes are only detected on a surface, where face was previously detected. If there was only one eye detected, it is being ignored and no alignment is applied. If there are two eyes detected, angle between them is calculated, but the result is returned in radians, so multiplication is required.

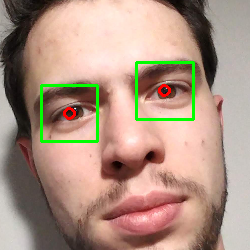


Figure 7Eyes detection in scope of face

Then, a few rules are being applied to filter whether the eyes that were detected, are not a false positive. Assuming that pictures are of the people with the face only in position more or less perpendicularly to the bottom of the image, when rotation above 20 degrees is detected, it is zeroed. Python atan2 function is working in such a way, that depending on which eye is higher, whether right or left, arctan value differs and function can also return a big negative number like -177. With such value rotation matrix that is further calculated would flip image about upside down. To adjust that value would be applied.



Figure 8Photo after rotation matrix was applied

Taking above angle, rotation matrix is calculated and applied to the photo. Although to avoid black corners that appear when rotation is applied to previously cut photo, rotation is applied to the main photo before the face cutout and the procedure with looking for the face is retried.

To avoid noises and training algorithm with unnecessary details Gaussian blur is applied. Also, histogram normalization method is applied, to emphasize the most distinct spots.



Figure 9Examplary output file

Last operation is to create a separate folder that processed images will be stored in. Every face picture has the same height and width, that is previously set in the program code. Images are saved in folders, that are named the same as the folders that they were originally stored in.

### Training recognizer

Second file available for user is a file that launches training process. OpenCV makes this process very straightforward. There is Eigenface Recognizer method used to create an object that contains train method. Then, previously created and aligned images are walked though, and based on the names of folders labels are created. Every image is appended to an array both with label. Then, train method with two previously created arrays is called and then save method is called to save trainer to yml file.

### Video processing

OpenCV provides VideoCapture function where name of the file to be processed can be specified. Then, haar-like cascade classifier is created to find faces on the image. Then, again Eigenface recognizer is created and previously created trainer.yml file can be read. This will allow to recognize already trained faces. A loop is used to iterate though every frame of the video. Each frame is put though following procedure. It is transformed to greyscale, and faces are detected by cascades method.

For reach face there is a prediction made, thanks to the predict function from recognizer object. Each recognition is compared to the predefined values with coordinates for each person that are manually provided, to test accuracy. Then, prediction is printed on the frame, above face of given person, with coordinates visible.

# Verification and validation

A few steps were taken to check that scripts meets the requirements and specifications and fulfills the intended purpose. Achieved accuracy and performance for test videos is described in this part, together with procedure that led to those outcomes.

## Verification procedure

Main focus of verification was to check how many frames of the video material has correctly recognized face. For each frame in a video haar-like cascade classifier tries to detect a face, and for each detection, a prediction made by using Eigenface method. To test whether the prediction is correct there were few steps taken. Each frame has a few predefined properties. Name of the person that the face is appearing at the given frame, and its location as coordinates of the video sequence. There is also an offset that can be set, assuming that not only the expression of the face is changing, but also its location.

For each frame confidence of detection is saved. Then, it can be used to set the threshold and filter predictions with low confidence. Each prediction is compared with previously set threshold and set to true or negative. Threshold testing is in range from the lowest possible prediction value to the highest that the algorithm calculated during recognition phase.

## Verification results

For the previously prepared video with three faces appearing one by one the recognition rate was quite satisfying. Correct recognition was performed in 80.7% of the frames of the video. It should be noted, that video material was containing faces that were turned straight into the camera and with a good lighting. In 12% of the frames haar-cascade classifier didn’t manage to detect a face, so no prediction or it’s accuracy could be measured. Wrongly recognized frames were only a 7.3% of the video, which could be treated a good outcome, considering that Eigenfaces is a rather simple classifier. When filtering out the frames where haar-like classifier didn’t manage to detect a face, recognition on already detected face is a 91.7% which is even a better result.

Then, extended version of a previous video was tested, and the results were very similar. Correct recognition was performed in 76.7% of frames. Wrong recognition was performed in 17.6% of the video and face was not located in 5.7% of the video. This translates into 81.3% of correct predictions considering only frames where the face was correctly detected.

The ROC Curve was prepared for the system:

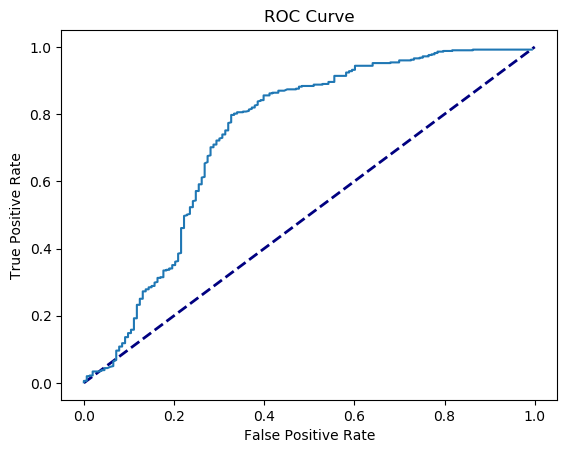


Figure 10ROC Curve for Eigenface recognition

# Conclusions

The main goal in this thesis was to implement the Eigenface detection method in video sequence and interpretation of results. Work on thesis was proceeded in previously defined steps. The choice of the programming language was made first, to allow for a fluent workflow. Then, analysis of current solution was made, and haar-like cascades were chosen for a face detection, and Eigenface recognition method was chosen to recognize faces. Implementation of the algorithms was successful and allowed to test the performance on video sequences. All the requirements specified on the beginning of the thesis were successfully fulfilled.

The process of implementing the recognition process was rather simple and straightforward. However, image processing came out to be more complicated, and this caused the drastic decrease of code quality. There should also be more tests with different lighting conditions, and the photo database could have been extended for better results.

Further extensions should target improvement of time execution, as some of the image transformations were redundant. Quality of the video could be also much more lowered for the purpose of testing, and for increased speed of detection.

Algorithms presented here can be easily implemented in a bigger system, for example for detection of dangerous people. It can also be extended to count attendance to school, or almost anywhere, where biological identification can be used to measure some statistics.

# Bibliography

|  |  |
| --- | --- |
| [1] | P. Viola i M. Jones, „Rapid object detection using a boosted cascade of simple features,” w *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, 2001. |
| [2] | P. Viola i M. Jones, „Robust Real-Time Face Detection,” *International Journal of Computer Vision,* tom 57, nr 2, p. 137–154, 2004. |
| [3] | "userbenchmark," [Online]. Available: https://cpu.userbenchmark.com/Compare/Intel-Pentium-4-160GHz-vs-Intel-Core-i5-4690K/m15237vs2432. [Accessed 29 12 2018]. |
| [4] | justinwl, "ancientelectronics," [Online]. Available: https://ancientelectronics.wordpress.com/tag/pentium-4-vs-pentium-iii/. [Accessed 29 12 2018]. |
| [5] | N. Ipe, "N Recursions," 16 10 2018. [Online]. Available: http://nrecursions.blogspot.com/2018/10/a-better-tutorial-on-haar-features-used.html. [Accessed 29 12 2018]. |
| [6] | R. Lienhart i J. Maydt, „An extended set of Haar-like features for rapid object detection,” w *Proceedings. International Conference on Image Processing*, Rochester, 2002. |
| [7] | T. Mita, T. Kaneko i O. Hori, „Joint Haar-like features for face detection,” w *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, Beijing, 2005. |
| [8] | W. K. Taylor, „Machine learning and recognition of faces,” IET, London, 1967. |
| [9] | M. &. B. D. Roomi, „A Review Of Face Recognition Methods,” *International Journal of Pattern Recognition and Artificial Intelligence,* tom 27, nr 4, 2013. |
| [10] | T. Shakunaga i K. Shigenari, „Decomposed eigenface for face recognition under various lighting conditions,” w *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Kauai, 2001. |
| [11] | G. M. Zafaruddin i H. S. Fadewar, Face recognition: A holistic approach review, Mysore: IEEE, 2014. |
| [12] | P. Belhumeur, J. Hespanha i D. Kriegman, „Eigenfaces vs. Fisherfaces: recognition using class specific linear projection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* tom 19, nr 7, pp. 711-720, 1997. |
| [13] | S. Jaiswal, D. (. S. S. Bhadauria i D. R. S. Jadon, „COMPARISON BETWEEN FACE RECOGNITION ALGORITHM-EIGENFACES, FISHERFACES AND ELASTIC BUNCH GRAPH MATCHING,” *Journal of Global Research in Computer Science,* tom 2, nr 7, pp. 187-192, 2011. |
| [14] | R. Huang, V. Pavlovic i D. N. Metaxas, „A hybrid face recognition method using Markov random fields,” w *Proceedings of the 17th International Conference on Pattern Recognition*, Cambridge, 2004. |
| [15] | H. R. R. J. B. C. O. &. M. S. Cho, „An Efficient Hybrid Face Recognition Algorithm Using PCA and GABOR Wavelets,” *International Journal of Advanced Robotic Systems,* tom 11, nr 4, 2014. |
| [16] | P. Campadelli, R. Lanzarotti i C. Savazzi, „A feature-based face recognition system,” w *Proceedings of the 12th International Conference on Image Analysis and Processing* , Milano, 2003. |
| [17] | D. Ibrahim, „An overview of soft computing,” w *12th International Conference on Application of Fuzzy Systems and Soft Computing*, Vienna, 2016. | |
| [18] | M. H. Fadzil i L. C. Choon, „Face recognition system based on neural networks and fuzzy logic,” w *Proceedings of International Conference on Neural Networks*, Houston, 1997. |
| [19] | F. Mahmud, M. E. Haque, S. T. Zuhori i B. Pal, „Human Face Recognition Using PCA based Genetic,” w *International Conference on Electrical Engineering and Information & Communication Technology*, Bangladesh, 2014. |
| [20] | S. Mallick, „Learn OpenCV,” 30 10 2015. [Online]. Available: https://www.learnopencv.com/opencv-c-vs-python-vs-matlab-for-computer-vision/. [Accessed: 6 1 2018]. |
| [21] | „docs.opencv,” Intel Corporation, [Online]. Available: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade\_classification.html. [Accessed: 10 January 2019]. |
| [22] | G. M. Zafaruddin i H. S. Fadewar, „Face recognition: A holistic approach review,” w *International Conference on Contemporary Computing and Informatics*, Mysore, 2014. |

# List of abbreviations and symbols

|  |  |
| --- | --- |
| *CPU* | Central processing unit |
| *ROC* | Receiver operating characteristics |
| *PCA* | Principal component analysis |
| *GA* | Generic algorithm |
| *UML* | Unified Modeling Language |
| *XML* | Extensible Markup Language |
| *JPEG* | Joint Photographic Experts Group |
|  |  |
|  |  |

# Contents of attached CD-ROM

The thesis is accompanied by a CD-ROM containing:

* thesis (PDF file),
* source code of applications,

# List of Figures

[Figure 1Basic Haar-like features [1] 5](#_Toc535110991)

[Figure 2Haar-like feature on face [5] 5](#_Toc535110992)

[Figure 3Extended features proposed by Lienhart and Maydt [6] 6](#_Toc535110993)

[Figure 4 Joint Haar-Like feature 7](#_Toc535110994)

[Figure 5Use cases UML 18](#_Toc535110995)

[Figure 6Examplary input file 27](#_Toc535110996)

[Figure 7Eyes detection in scope of face 28](#_Toc535110997)

[Figure 8Photo after rotation matrix was applied 29](#_Toc535110998)

[Figure 9Examplary output file 30](#_Toc535110999)

[Figure 10ROC Curve for Eigenface recognition 34](#_Toc535111000)

# List of Tables

[Table 1 List of libraries needed to run the program 20](#_Toc535110247)

[Table 2 requirements.txt file 22](#_Toc535110248)