



University of
Zurich^{UZH}

AVVS - Automated Video UZH-ID Verification System

*Anna Katharina Fitze
Zürich, Switzerland
Student ID: 11-931-722*

Supervisor: Bruno Rodrigues, Eder Scheid
Date of Submission: November 26, 2020

Abstract

Im Zuge der Digitalisierung und insbesondere im Verlauf der Pandemie von Covid-19 werden immer mehr Vorlesungen und Kurse auf Digitalunterricht umgestellt. In einer gewöhnlichen Prüfung wird dabei, die Präsenz mittels Liste und Legitimationskarte überprüft. Dies wird digital erschwert. Der vorliegende Prototyp soll in vereinfachter Form modellieren, wie eine Identitätsüberprüfung online stattfinden könnte, dabei wird das Gesicht mit gespeicherten Porträtfotos abgeglichen und die Informationen auf der Legitimationskarte werden mit einer abgespeicherten Liste verglichen. Die Werte, die jeweils mit den gespeicherten Werten übereinstimmen, werden nochmals miteinander verglichen, um zu überprüfen, ob die Person vor dem Computer tatsächlich mit der Person auf der Legitimationskarte übereinstimmt. Die Modellierung eines Prototypen, entspricht einer stark vereinfachten Idee, mit lokalen Ordnern als Datenbanken und mit dem Zugriff auf die eigene Computerkamera anstelle der Simulation einer Konferenz. Es ist festgelegt, dass das spätere System nur ausgewählten Mitarbeitern der Universität (Professoren/Tutoren) zur Verfügung steht, da es mit persönlichen Daten der Studenten hantiert. Am Ende wird der Prototyp evaluiert und an zwei Beispielen getestet.

Caused by digitization and the Covid-19 pandemic courses are transformed to remote classes. In usual exams the presence of students is verified and validated by the teacher, who checks on every student with a list of students and with the UZH-Card. Digitally this is way more difficult. In this thesis a prototype is developed to tackle this problem. In a simplified version it models a way to check the identity of a student in a online class. Therefore the information on the UZH-card is read and compared to a stored list of student information. Also the face of the student in front of the computer is compared to stored values. The matching information from both comparisons are again compared to see if the person in front of the camera is the person, who they claim to be (the person on the UZH card). Obviously the prototype is simplified with dummy databases and access to the own webcam instead of other webcams. Furthermore, it is specified, that this prototype is only available to certain people with a teaching function at the University, because the system works with personal data, which should not be accessible to everyone. In the end there is an evaluation on the precision and two test cases are presented.

Acknowledgments

I would like to thank Bruno Rodrigues, my supervisor, for his time, his advice and his patience guiding me through the last six month with this thesis. I appreciate it a lot, that he regularly took time for a short update-meeting.

I would also like to thank Prof. Dr. Burkhard Stiller, the head of the Communication System Group (CSG) at the University of Zurich, for allowing me to research and write about this topic.

Also I want to thank all my friends, who provided me with their photos for developing the face recognition.

Contents

Abstract	i
Acknowledgments	ii
1 Introduction	1
1.1 Overview: Related Work	1
1.2 Motivation	2
1.3 Thesis goals	3
1.4 Description of Work	3
1.5 Thesis Outline	4
2 Background and Related Work	5
2.1 Background	5
2.1.1 Optical Character Recognition - Tesseract	5
2.1.2 Principal Component Analysis (PCA)	5
2.1.3 Linear Discriminant Analysis (LDA)	6
2.1.4 Geometrical features	6
2.1.5 Template Matching	6
2.1.6 Convolutional Neural Networks	7
2.1.7 Support Vector machines	7
2.2 Related work	7
2.2.1 Tools and Methods	7
2.2.2 Existing projects	9
2.2.3 Tutorials	13

3	Requirements and Design	17
3.1	Non-functional Requirements	17
3.2	Functional Requirements	17
3.3	Design	17
3.3.1	Workflow	18
3.3.2	Restrictions and Simplification	18
3.3.3	General Design	19
4	Implementation	22
4.1	Architecture	22
4.2	Packages	23
4.3	Starting webcam	24
4.4	Text recognition	24
4.5	Face recognition	26
4.6	Matching	28
5	Evaluation	30
5.1	Functional Requirements	30
5.2	Precision	31
5.2.1	Precision Face recognition	32
5.2.2	Precision Text recognition	33
5.3	Test cases	33
5.3.1	Negative example	33
5.3.2	Positive example	34
5.4	Discussion	36

6	Conclusions	38
6.1	Conclusion	38
6.2	Obstacles	39
6.2.1	String comparison	39
6.2.2	Code quality	39
6.2.3	Time management	39
6.2.4	Coding vs writing	40
6.3	Future Work	40
	Bibliography	41
	Abbreviations	44
	List of Figures	44
	List of Tables	45
A	Technical Guidelines	47
A.1	Text recognition	47
A.2	Face recognition	47
B	Contents of the Hand-in	48

Chapter 1

Introduction

Recently, with the COVID-19 pandemic shutdown measures introduced by many countries all over the world, universities were forced to move from physical classroom settings to remote classes to help flatten the contamination curve. With this shift, for a selected set of remote classes and specially examinations, it is necessary to check, if the student participating in the class or in the exam is enrolled in the course or not and if he/she is who he/she claims to be (individuals authentication) to avoid frauds. In this sense, to verify such claims, facial recognition using the student's webcam and a valid identification (ID) card (e.g., the University of Zurich - UZH student card) could be employed. However, such a system will be highly specialized and tailored for a specific university due the different designs of ID cards and no defined standard. Therefore, there is a need to design and develop an automatic system (to reduce the need for human intervention) to verify whether a student is enrolled in the current semester and to check if a Student-ID is valid and also if that student is enrolled in the current course. The idea is to use such a system internally at UZH, such that a necessary database with relevant UZH data can be utilized for certain checks. The thesis plans to run an image/facial recognition test to check if the student is the same as on the photo of the UZH card. With a text extraction, the system checks if the student is enrolled in the current semester. And also with a text extraction the system compares Student-IDs with the student IDs enrolled in the particular course. For these tests dummy data can be used to investigate if the system works effectively.

1.1 Overview: Related Work

In the next second chapter there is more information about tools, concepts, existing projects and tutorials. Here is a short overview on methods from existing projects that could be helpful on this project.

There are two major building blocks toward student identification. On the one hand, it is testing if the student in front of the student's webcam is the student on the ID card, on the other hand, first it is the text on the UZH card for figuring out the Student-Id and if the card is validated, the second part is the photo of the student which needs to be

compared to the student in front of the camera. While the first part concerns the use of OCR (Optical Character Recognition), the second relies on image processing in order to extract visual patterns that can be matched with a database. There are ways to detect text in images [1]. The authors introduce multi-valued image decomposition, which is a method deciding whether pixels are foreground or background to further analyze the color uniformity. The authors also talk about text location using connected component analysis, which is only applicable to binary images [1].

For the next step an OCR method is required to not only locate textual elements in the image, but also to be able to read it. The authors of a paper [2] made an approach for this issue and used Tesseract, which is an open source OCR engine, which can be used for extracting text from an image, which works in two passes of going through a text and recognizing words, in the second pass it recognizes the words, which were not recognized well in the first paper [2]. Tesseract accepts binary images as input, what would allow text location with connected component analysis [2]. In the same paper [2] they also develop a Text to speech engine which is not necessary for this project. In another paper [3] Fisherface and Eigenface are introduced as facial recognition algorithms, this might be a suitable tool in this thesis, because it is required to compare the photo on the card with the student live image. According to this paper [3] the Eigenface algorithm usually returns better results than the Fisherface.

The Eigenface uses Principal Component Analysis (PCA) as a method for facial recognition, it works with a dimensionality reducing linear projection that maximizes the scatter of all projected samples [6]. And the Fisherface uses a Linear Discriminant Analysis (LDA), this is a class specific method, which tries to bring the scatter in a suitable form in order to make it more reliable for classification [6]. The authors of another paper claim that the Eigenface approach is rather fast and reliable in a constrained environment [5]. In an attendance system the Eigenface algorithm performance outclassed the Fisherface algorithm [3]. Based on research about the Eigenface methods [5] an algorithm is shown on how to apply the Eigenface method [4]. The last paper [4] is an interesting extension or helpful for understanding the Eigenface algorithm.

1.2 Motivation

Due to COVID-19 pandemic lectures were held as online classes. Also, examinations could not be taken physically. Remote exams increase the possibilities to commit frauds in exams. Therefore, it is of vital importance to identify the student taking the exam, to make sure the student, taking the exam, is the student meant to take the exam. This consists of multiple parts. In physical classes the students have to put their UZH-card on the table and the lecturer or an assistant has to check on the card, compare the photo with the present student and compare the id-number with the list of id-number registered for the course. The intention of this work is to develop a prototype for remote classes to do this task automatically via webcam. As already indicated, this prototype consists of major two parts. One part is to read and compare the Identification (ID) number on the UZH-card with the list of ID-numbers registered in the particular course. And the other is to compare the face shown in the webcam to the photo on the UZH-card.

1.3 Thesis goals

Driven by the description of the work description, the following bachelor thesis goals are required:

- Problem analysis and planning: At first, it is required to gain a profound understanding of image recognition and processing. Tools for Optical character recognition needs to be analyzed in order to compare the student-id on UZH-card with the student-ids on the list. Also methods for facial recognition needs to be researched to compare the photo on the UZH-card with the student in front of the webcam. Probably knowledge about Machine Learning is also preconditioned. Since there needs to be set a threshold for similarity, when it comes to facial recognition.
- Elicitation of functional and non-functional requirements for the development of the facial identification and validation system for exams. In this goal, the first and second stages as described above are involved, in which existing approaches must be contrasted with the specific benefits and drawbacks of the application, in order to verify which approaches are most efficient. In the same step a system design will be created.
- Development and evaluation of the proposed system through a prototype. Based on the requirements presented in the previous objective, it is expected in this objective the design of a system satisfying the previously proposed requirements, as well as the implementation and evaluation of this system in a prototype. As mentioned earlier, data must be produced for evaluation due to student privacy. At last it is utterly important to verify the precision of the algorithm. The percentage of correct identification is an important evaluation parameter.

Furthermore, these goals may result in slightly different requirements, which may influence the activities schedule. In this sense, premises could be considered in agreement with the supervisor in order to restrict, change, or adapt activities to the scope of the thesis, and consequently, its objectives. Moreover, a scientific publication containing the results of the thesis might be produced.

1.4 Description of Work

This bachelor thesis requires, at an initial stage, that the student acquires all conceptual elements being involved in the proposal as well as to understand the baseline system to perform identification and validation of students enrolled in exams, in order to fulfill the objectives of this thesis. Also, it is expected in the initial stage an overview of the entire system (i.e., basic concepts and existing tools and approaches) toward the understanding of the problem and deriving requirements through the main related works, which includes the planning and preparation of the activities to be performed in the thesis in a timeline comprising the main stages. After understanding the theoretical concepts

the requirements and the design is to be created. Based on these requirements and design a prototype will be developed. This prototype will be evaluated regarding its reliability.

The system, involves facial recognition of students to take an online exam. In this exam, students must present their university identification card. Thus, the system must be able to identify whether the person in the image corresponds to the photo of the card, as well as to validate whether the information on the card corresponds to that existing in the database. Also, it is important to clarify that the data used for the development and validation of the system must be produced with test/experimental data, not interfering with the privacy of people until the approach presented in the prototype is valid by the educational institution's. The third and last stage involves the bachelor thesis development, evaluation and documentation. Based on the design decisions listed on the second stage, these features shall be integrated and evaluated on the system. The working prototype consists of the integration of the designed and implemented elements, producing results to be contrasted with the thesis goals. Also, during this stage decisions on how to best implement and evaluate those design decisions shall be made during periodical meetings with supervisors. The Evaluation should include, how precise the prototype is. A final report should include a motivation and problem description, background information, related work, design decision, implementation details, evaluation and conclusions. Note that two reports are required, a mid-term report describing the current progress of the thesis and the final report.

1.5 Thesis Outline

Since this work is based on existing tools in **Chapter 2** related work and existing tools will be discussed. Also there is a section about existing projects, which solved similar problems. In addition, there are some tutorials presented, which were useful in the development of the prototype. **Chapter 3** lists the requirements and shows the design of the prototype. In **Chapter 4** there is a detailed description of the implementation of the prototype. **Chapter 5** there is a short evaluation about the precision of the prototype and two test cases are examined. In the last chapter **Chapter 6** are the conclusions, a short description of obstacles during the implementation and an impression of possible future works.

Chapter 2

Background and Related Work

In this chapter concepts, tools, existing projects and tutorials can be found in the field of text or face recognition. This is divided into **Background** for concepts and methods, and **Related Work** for tools, existing projects and tutorials.

2.1 Background

In the first part of this chapter, there are some concepts introduced which build a basis for further methods in Image Recognition.

2.1.1 Optical Character Recognition - Tesseract

Tesseract is a well-known, open-source engine for Optical Character Recognition (OCR). It was developed by Hewlett Packard in 1980s. Today it is further developed and maintained by Google [23]. Tesseract accepts binary images as input[7], therefore black-on-white text as well as white-on-black text can be processed. Its architecture is built in a way, that a text is run through twice. In the first run an attempt is made to recognize each word in turn. Those recognized words are handed over to an adaptive classifier as training data. With the newly acquired knowledge the adaptive classifier runs through the text again to recognize the words that have not been recognized yet [7][2] .

2.1.2 Principal Component Analysis (PCA)

PCA is a method to reduce dimensions especially useful in large datasets. The main goals of PCA are following [15]:

- extracting the most important information from a datatable
- reduce the size of the dataset by keeping the important information

- simplify the description of the dataset
- analyse the structure of the dataset

To achieve this goal the principal components are computed. The principal components are a linear combination of the existing variables. The first principal component should have the largest possible variance. The second component has to be orthogonal to the first one and to have the largest possible inertia. The inertia of a column is the sum of the squared elements of a column. The inertia of the table, also known as total inertia, is the sum of all inertia of columns. All the other components are computed likewise. Those new variables return values that are called factor scores. And these factor scores can be interpreted as projections of the observations onto the principal components. The inertia is here to tell us about the importance of the component [15].

2.1.3 Linear Discriminant Analysis (LDA)

LDA is another well-known method for data classification, it pursues similar goals as the PCA. The main difference between PCA and LDA is that LDA does more of data classification whereas PCA does more of feature separation [16]. LDA maximizes the ratio of between-class variance to within-class variance to achieve maximal separability [16]. In the first step, after formulating the data sets, the mean of each data set and the entire data set is to be calculated. Then the between-class and within-class scatters have to be calculated. The expected covariance of each of the classes is the within-class scatter. Eigenvectors of a transformation represent an invariant subspace of the vector space in which the transformation is applied. A set of linearly independent Eigenvectors is invariant under the transformation. The matrix of the Eigenvectors can transform to the data set to the new subspace[16].

2.1.4 Geometrical features

One of the most obvious approaches of facial recognition is computing ratios and relative positions. However in corresponding research papers this method reached recognition rate varying between 45% - 75% in a paper with a database of 20 people by Takeo Kanade [10] or a recognition rate of 90% with a database of 47 people by Brunelli and Poggio [11]. Brunelli and Poggio compared the method of geometrical features to template matching, and coming to recognition performance Template matching was superior, the feature based strategy however is faster and needs lower memory requirements [11].

2.1.5 Template Matching

In Template Matching a face is compared to a template, and the Euclidean distances are computed. This may be done with one or more templates [11]. Also the image can be preprocessed e.g by adjusting the light or the colors. For a more complex version a

qualitative prior model of how a generic face transforms under a change of viewpoint can built in the matching templates approach [11] .

2.1.6 Convolutional Neural Networks

An alternative to the Eigenface Method, which is based on the principal component analysis, is a Convolutional Neural Network (CNN) [9]. CNN are widely known for image analysis but also classification problems, so it seems to be an interesting tool for Pattern detection. Simplified CNN have layers and every layer has a filter, the more layers it has the more sophisticated is the image analysis. In [9] the first 50 principal components of the image are extracted and reduced, which seems not that different from a usual PCA. However, then for reducing the dimensions an auto-associative neural network is used. A standard MultiLayer Perceptron (MLP) then classifies the representation, resulting from the neural network [9]. CNN can also be applied in fields of Character Recognition [8].

2.1.7 Support Vector machines

Support Vector Machines (SVM) are well-known for classification problems. Same as the CNN it belongs to the supervised learning models. It creates a n-dimensional vector. Now the bigger the euclidean distance is the more different are the elements for classification[30].

2.2 Related work

In the first part of this section methods are introduced, that can be used for face recognition. In the second part there are existing projects and papers presented, which did similar things as planned in this thesis. And in the last part, there are descriptions of some tutorials that are used in to build the AVVS.

2.2.1 Tools and Methods

Here is a short introduction about tools and methods for face and also text recognition. There are only some selected methods presented, which appeared in scientific papers.

Eigenface

Eigenfaces is a method for facial recognition using the Principal Component Analysis (PCA) in order to overcome expensive computation and the need of big amounts of storage of older facial recognition techniques [1]. This method reduces the dimensionality of linear projection that maximizes the scatter of all projected samples. So a high-dimensional

training set is represented in a lower dimensional subspace [1]. It is reported that this method delivers rather reliable results, especially if the face is to be compared with known individuals, which is the case in this project [8]. It does, however, require similar circumstances in terms of light or head variation for comparisons [8] .

Fisherface

The Fisherface method uses the Linear Discriminant Analysis (LDA). LDA is also used for reducing the dimensionality finding a linear set of features that characterizes or separates different classes of objects. The Eigenface methods extracts the most expressive features, whereas the Fisherface methods extracts the discriminating features [12], which makes it very suitable for classification.

Open Intelligent Multimedia Analysis for Java

Open intelligent Multimedia Analysis for Java (OpenIMAJ) is a set of libraries and tools for Java for analyzing multimedia content. It is amongst other things suitable for implementing the Eigenface method, since it contains the class EigenImages. This class can automatically deal with converting input images into vectors and subtracting the mean before applying the PCA. However the Eigenfaces work best in full-frontal view and they work better if the faces are aligned, this should be considered when using this approach [13].

Tesseract - Google Engine

The current version of Tesseract is open source and can be found on github [23]. It is written in C++ and maintained by google. It recognizes about 100 languages. Tesseract has different textual output formats. For opening input images tesseract uses the Leptonica-library, which supports png, tiff and zlib[23].

Pytesseract

Python tesseract (pytesseract) is a library implementing the OCR-engine tesseract [3]. It supports the following formats png, jpeg, gif, bmp and tiff. Also the language can be configured. Its possible inputs are also restricted by Leptonica [3].

Leptonica

Tesseract is based on Leptonica[22], which is a open-source software mainly used for image processing and analysis. This is why tesseract-libraries have some restrictions on allowed input formats. Leptonica allows: Jpeg, png, tiff, bmp, pnm and gif [22].

2.2.2 Existing projects

There are already project, which deal with the topic of face recognition and text recognition, here a selection of the corresponding papers is presented.

A real-time face recognition system using Eigenfaces from Daniel Georgescu

Georgescu [14] differentiates facial recognition systems between face verification, face identification and the watch list. **Face verification** is about a comparing a face to a certain entry in a database. The person claims to be someone and now it is needed to prove, that this claim is correct. **Face identification** is about identifying an unknown person. The whole database must be searched through. And the **watchlist** is figuring out if a person is part of a list *e.g.* a passenger at the airport is compared to a list of known criminals. Generally a face recognition system divides into four modules: the face detection, image processing, extraction of face features and feature matching modules. For face detection he uses the **Canny edge algorithm** consisting of noise reduction, finding gradients, non maximum suppression, double thresholding and edge tracking by hysteresis. The first step noise reduction is removing all noise in the picture which is naturally contained in pictures. This is to prevent that noise is mistaken for edges. Georgescu does this with applying a Gaussian filter. For the human eye a noise reduction is not visible [14].

Finding gradients within the Canny algorithm is finding the edges where the gray-scale intensity of the image changes the most. The purpose of the step non-maximum suppression is to convert the blurred edges into sharp edges. To avoid edges by color variations or remained noise a threshold is set. Double thresholding includes two thresholds, where edge pixels above the high threshold is marked as strong, edge pixels weaker as the threshold is suppressed and edge pixels between the low and the high thresholds are marked as weak. Edge tracking by hysteresis is the last step where the final edge image is created. Strong edges are directly includes whereas the weak edges are only included if they are connected to a strong edge, because noises and color variations are unlikely to end in a strong edge. This algorithm results in a picture with all the shapes as the original picture but no colors [14].

Georgescu then applies the **Viola-Jones face detection algorithm**. This algorithm again consists of two parts first the **integral image** that allows a fast features evaluation, integral image are rectangle feature computed using an intermediate representation for the image. The second part is the **AdaBoost Learning Algorithm**. This method constructs a strong classifier by selecting a small number of important features. The Viola-Jones face detection algorithm uses a slightly modified version of the AdaBoost Learning algorithm, that not only combines a collection of weak classification functions in order to build a strong classifier, but selects the features too. The algorithm helps to determine the optimal threshold classification function and selects the single rectangle, which separates the positive and negative examples best. For facial recognition Georgescu uses the Eigenface method based on PCA. According to him a small number of eigenfaces is sufficient for identification, if the vectors with the largest associated eigenvalues are taken into account. Furthermore he quotes a comparison from Toygar and Acan of PCA, LDA

and Independent Component Analysis (ICA) [20] to be seen in Figure 2.1. According to this, PCA, which is included in the Eigenface methods is rather reliable, when it comes to the success rate. It sets also the highest threshold [14].

Number of Training Faces	Method	Threshold	Success rate (Percentage)
25 (5x5)	PCA	3000	93
	LDA	500	85
	ICA	60	36
50 (10x5)	PCA	3000	79
		4000	89
		5000	93
	LDA	500	45.5
		800	67
		1000	79.5
		1200	85
	ICA	60	32
		100	32
		500	32

Table 2.1: The PCA, LDA and ICA are compared, PCA in terms of the threshold and the success rate with two numbers of training faces [20]

Real-time face recognition using Eigenfaces from Raphael Cendrillon and Brian C. Lovell

Cendrillon et al [19] use a system architecture consisting of following modules:

- **Temporal filter:** Since the input is a video feed, the video is segmented into regions of movements.
- **Face Location:** Within those regions of movements the faces are located and passed onto the feature location module.
- **Feature Location:** Locates the eyes in the segmented face.
- **Scale and Rotation Normalization:** According to the location of the eyes the faces are normalized in terms of scale and rotation.
- **Lighting normalization:** After the image is geometrically normalized, this module compensates for variations in lighting conditions.
- **Face Recognition:** Finally the normalized faces are projected into face-space to determine its identity.

Simplistic methods for **temporal filters** include background subtraction and frame differencing. But uniform changes in the ambient lighting might decrease the performance. Using several frames to locate regions of movement can be used to compensate this problem [21]. To decrease the sensitivity to lighting variations, the weighted sum of several frames can be computed. There was a temporal filter implemented, which works by convolving several frames in the temporal dimension with a Gaussian second derivative. After this temporal filter zero-crossing is used to determine the region of movements. Then the face is located. The more movement is in the scene the more selective the detected regions the filter becomes. On the contrary is there little movement within the scene, the filter becomes more accepting in the detected regions of movement [19].

The temporal filters already located moving regions prior to **face location** to increase efficiency and robustness. Those regions are then passed over to the face detection algorithm, which finds the most probable location of a face by computing a face-similarity by projecting into face-space [19].

Prior to recognition a located face needs to be **normalized for scale, rotation and lighting**. Not normalizing faces will result in a decreased performance because of the increased dimensionality of the problem. To normalize the face image for scale and rotation, the locations of the eyes are used. To normalize for lighting they set the mean to zero and the L2-norm to unity[19].

For the **facial recognition** Cendrillon et al used the Eigenface method and the Mahalanobis distance. They achieved a recognition rate of 95.5%. However the recognition rate decreased significantly for rotations of more than 10 degrees and size reductions[19].

Implementation of Face recognition algorithm for biometrics-based time attendance system by Siswanto, Nugroho and Galinium

The purpose of this project by Siswanto et al [3] is to implement and develop a face recognition provided by **OpenCV 2.4.8**. Therefore they compare the performance of the Eigenface algorithm with the Fisherface algorithm based on their **receiver operating characteristics (ROC)** curve. The ROC curve is a visual characterisation of the trade-off between false match rate (FMR) and the false non-match rate (FNMR). FMR is obtained when a certain threshold is set for the recognition algorithm to recognize an impostor as a genuine match. FNMR is obtained, when a genuine user is blocked to his own access. The created ROC plot for both Eigenface and Fisherface will inform the researcher about the best algorithm to use for handling a certain training set. This project is divided into two parts, one is the "collect face data" and the other one is the "attendance recognition". The collecting the face data part creates a training set was with students in front of the camera. To minimize potential variations in the images the faces are taken in the same room, in the same chair with black background and all the subjects used the same pose. Facing straight to the camera allowed pre-trained face detection algorithms inside OpenCV 2.4.8 to work. **Haar cascade from OpenCV 2.4.8** was used in this research for eye detection. The used LBP-cascade is used for frontal face detection. In this process face detection will detect the whole face and both eyes. In this research the user presses a button "Collect face", whenever his/her face is displayed correctly. Differently as planned

in this thesis the webcam is only activated when a valid Student-id is provided. For the attendance recognition the student's face must be presented in front of the camera and a button "Attend" must be pressed in order to be marked as present. The student in the training set will be recognized, whereas the others are marked as "unrecognized person". A Threshold too high will produce a FMR too low and a FNMR too high, but a threshold too low will produce a high FMR and a low FNMR. In this research the Eigenface outclasses the Fisherface, since the ROC curve is a bit lower and therefore more reliable [3].

Optical Character Recognition by Mithe, Indalkar and Divelkar

The goal of this project [2] is to create a image to speech application, where text from images are read and converted to speech. Clearly the conversion to speech part is not necessary for this project, but the text from image extraction can be useful. Optical Character recognition (OCR) can recognize handwritten as well as printed text, but the better the quality of the input document, the better is the performance of OCR. The authors decided to use **Tesseract** [23] for OCR, because of widespread approbation, its flexibility and extensibility, its community of active developers and because it works out of the box. The process of character recognition is divided into three steps. **Segmentation, Features extraction and classification**. However, for the image input were Android mobile cameras used in the project by Mithe, which is not what is planned for this project [2].

Automatic text recognition in digital videos by Lienhart and Stuber

The authors of this paper [24] wanted to extract text from videos to generate more information about the content. The authors assume monochrome characters and therefore, they segment the frame into homogeneous gray scale segments in the first step. Then the **split and merge algorithm** by Horowitz and Pavlidis is applied [26]. Every image is split into quarters, if a quarter is not homogeneous enough it is again split into quarters. A homogeneous segment is assigned its average gray level. Then adjacent segments are merged together if their average gray tone intensity difference is lower than a certain threshold. This should lead to all monochrome characters appearing in the image should be contained in some of the monochrome segments. Some of the homogeneous regions are too large or too small to be instances of characters. So they are removed. Since text is usually in a number of consecutive frames, it can improve the segmentation result by using these multiple instances of the same text. Text might however be either stationary or linearly moving, and even stationary text might move by some pixels from frame to frame, it is necessary to perform motion analysis in order to detect corresponding character candidate regions in consecutive frames [24].

Automatic number plate recognition system for vehicle identification using optical character recognition by Qadri and Asif

The created Automatic Number Plate Recognition (ANPR) in this paper [25] consists of three steps. The first one is the detection and capturing of a vehicle image. The

second step is the detection and extraction of number plates in an image. The third one is using **image segmentation techniques** to get individual character and optical character recognition to recognize the individual character with the help of database stored for each and every alphanumerical character. The Software part of the project presented here is written in Matlab 7.0.1. The Algorithm has three tasks: **capture the image, extract the plate from the image and recognize the numbers** from the extracted plate. The detection of the plate are here seems to be rather easy since the number plates in this region of Pakistan have a yellow background. This is not very helpful for locating the UZH-card area in an image, since the UZH card is mainly white, with some blue parts. After the **Region of interest (ROI)** is identified, the authors of this paper used two different filtering techniques. The first techniques removes all white patches that are connected to any border. The other technique removes all other regions in an image but the plate region. In the end the image contains only the vehicle number plate. In the OCR process the text is also separated in rows, because vehicle number plates in Pakistan can have more than one line. In the end the extracted vehicle number is compared to the stored database for vehicle authorization, which can be very useful, since we need to compare the number on the UZH-card to some sort of Database. For the ANPR there is also a hardware part, starting from sensors to sense the vehicle to finally open the barrier when accepted, this is mostly not useful for the validating the UZH-card. So we will not further investigate into the hardware model in this case [25].

2.2.3 Tutorials

When implementing it turned out, that there are already some really useful tutorials existing.

Face recognition with Python in under 25 lines of code

The author of this tutorial [37] explains how to use the OpenCV library for face recognition. OpenCV is a popular library for computer vision not only for C/C++ but also for Python. OpenCV uses machine learning algorithms to search for faces within a picture. Face detection in an image is a complicated matter, every piece of an image has to be searched and decided if this is a face or not. Because of this issue OpenCV uses cascades. The OpenCV cascade breaks the problem of detecting faces into block. For each block it does a quick test, if this quick test is positive it does a more detailed test. The algorithm has 30 to 50 stages of tests , and a face is only detected, when all tests result positive. The most of the picture parts will return a negative result in the first tests, so there is no time wasted on these parts. This leads to a faster face recognition. Those cascades are a bunch of XML files for the wanted object, that need to be initialise with your code. Here was a XML file called `haarcascade_frontalface_default.xml` was used. This XML file an be downloaded on Github[38]. The face recognition is here shown on a picture of a music band. After passing this image and the cascade names, the image is converted to grayscale. Because when using OpenCV grayscale is used for a lot of operations.

```
1 image = cv2.imread(imagePath)
2 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Then for detecting the faces following code is used.

```

1 faces = faceCascade.detectMultiScale(
2     gray,
3     scaleFactor=1.2,
4     minNeighbors=5,
5     minSize=(30, 30),
6     flags = cv2.cv.CV_HAAR_SCALE_IMAGE
7 )

```

The detectMultiscale function detects objects. And here it is called on the faceCascade, because it looks for faces. The first option saying "gray" is the grayscale image. Faces closer to the camera, seem bigger than faces in the background. Scalefactor compensates for this. The minNeighbors line defines how many objects are detected next to the current one before it declares a face found and minSize returns the size of each windows.

```

1 for (x, y, w, h) in faces:
2     cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
3 cv2.imshow("Faces found", image)
4 cv2.waitKey(0)

```

The last step it to draw rectangles around the faces to receive a visual result on the program[37].

Face detection in Python using a Webcam

There is a follow-up tutorial for the tutorial mentioned right before. This tutorial [39] addresses to face detection when using a webcam, instead of just one photo. So some code is similar or even equal to the tutorial before. But now we can lay the focus on the webcam part. The initial part with importing packages and initialising the XML-file is the same. Until the first line, addressing the webcam, appears.

```

1 video_capture = cv2.VideoCapture(0)

```

This line sets the video source to the default webcam in our system. Then the tutorial introduces a while-loop.

```

1 while True:
2
3     ret, frame = video_capture.read()

```

The loop starts with the video_capture.read(). The returned values are the frame. So, the video is cut into frames to examine them. The "ret" is a return code which tells if there anymore frames, since this is a video input, this value is not utterly important to us. Then follows the code from the tutorial with a small difference.

```

1 cv2.imshow('Video', frame)

```

To show the results there is a window on the display where the video is shown. At some point the user wants probably to stop the video with the detected face running on the display.

```

1 if cv2.waitKey(1) & 0xFF == ord('q'):
2     break

```

Here it is solved to press the letter "q" to quit the program.

```
1 video_capture.release()
2 cv2.destroyAllWindows()
```

Now the last thing to do is to release the resources [39].

Text detection and extraction using OpenCV and OCR

This tutorial [40] provides insights for the text recognition part. The packages `opencv-python` and `pytesseract` are required to be installed first.

```
1 pip install opencv-python
2 pip install pytesseract
```

This is somehow interesting, because in the import part of the program this `opencv-python` is no more mentioned. Eventhough to install it, it is called 'opencv-python', in the import statement it is called 'cv2'. The only import statements needed for the text recognition are these two:

```
1 import cv2
2 import pytesseract
```

Then the location where in the system the Pytesseract-OCR is installed has to be defined.

```
1 ct.pytesseract.tesseract_cmd = 'System_path_to_tesseract.exe'
```

After installing and importing the packages and reading the image, the colorspace of the input needs to be converted. The two options are to use `cv2.COLOR_BGR2GRAY` to convert into grayscale color or to convert to HSV (Hue, Saturation, Value) colorspace using `cv2.COLOR_BGR2HSV`.

Then a thresholding is applied. The thresholding is to determine if a pixel is counted as yes or no. So if the value of a pixel is below the threshold it is set to zero and if it is above the threshold it is set to the maximum, which is generally 255 [41]. On grayscale images the thresholding is easier done than on color images. That is why the color was converted before [41]. There are three types of thresholding [40]. The simple thresholding, adaptive thresholding and the OTSU Binarization. The thresholding function `cv2.threshold()` has four parameters. The first parameter is the color-space changed image, the second is the minimum threshold value, followed by the maximum threshold value and the last one is the type of thresholding that is applied here.

```
1 # color conversion
2 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
3
4 # Performing OTSU threshold
5 ret, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
```

Then the structural element is chosen. The tutorial goes for a rectangular structural element. An extra size of the kernel is taken as a parameter for the `cv2.getStructuringElement`. The bigger the kernel the larger blocks of text are grouped together. So the Kernel size increases or decreases depending on the rectangle to be detected. A smaller value like (10,10) will detect each word instead of each sentence. After choosing the structural element and the kernel, dilation is applied to the image. Dilation expands a text and leads to more accurate detection.

```

1 #choosing structural element
2 rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
3
4 #applying dilation
5 dilation = cv2.dilate(thresh1, rect_kernel, iterations = 1)

```

In the now dilated image contours should be found. For `cv2.findContours()` are three arguments passed: the source image, the contour retrieval mode and the contour approximation method. The returned values are contours and hierarchy. The returned value contours delivers a list of all the contours in the image. Each contour is an array containing the coordinates of the boundary points.

```

1 contours, hierarchy = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

```

The last thing after all these preprocessing of the image is the actual OCR. Therefore there is a loop through each contour to receive the (x,y) coordinates as well as the heights and the widths. For this task the function `cv2.boundingRect()` is used.

```

1
2 file = open("recognized.txt", "w+")
3 file.write("")
4 file.close()
5
6 for cnt in contours:
7     x, y, w, h = cv2.boundingRect(cnt)
8
9     # Drawing a rectangle on copied image
10    rect = cv2.rectangle(im2, (x, y), (x + w, y + h), (0, 255, 0), 2)
11
12    # Cropping the text block for giving input to OCR
13    cropped = im2[y:y + h, x:x + w]
14
15    # Open the file in append mode
16    file = open("recognized.txt", "a")
17
18    # Apply OCR on the cropped image
19    text = pytesseract.image_to_string(cropped)
20
21    # Appending the text into file
22    file.write(text)
23    file.write("\n")
24
25    # Close the file
26    file.close

```

With the help of the received information `cv2.rectangle()` draws a rectangle in the image. The five parameters used by `cv2.rectangle()` are following: specification of the input image, the starting (x,y) coordinates (lower left), the ending (x+w, y+h) coordinates (upper right), the boundary color of the rectangle in RGB value and the size of the boundary. This rectangle can be cropped and passed on to the tesseract engine to extract the text, which is here directly written to a text file [40].

Chapter 3

Requirements and Design

In the first part of this chapter the functional and non-functional requirements are listed. The second part is about the design of the system.

3.1 Non-functional Requirements

- The system works on all operating systems

3.2 Functional Requirements

- A list of students including name and student ID should be accessed.
- Information from the UZH-card should be read by the AVVS.
- The information from the UZH-card must be compared to the information on the list with names and IDs.
- The AVVS has to compare the face on the webcam to the photo on the database/-database replacement and tell if it is the same person
- The AVVS makes clearly visible if a student is accepted or not.

3.3 Design

To give a short overview about the design, we will start with the workflow as an introduction. For illustration purposes there is also a workflow diagram in the figure 3.1. Next is the description of the restrictions and simplifications, that needed to be set to create a prototype. At last the general design is more closely described.

3.3.1 Workflow

The general design is proposed as in Figure 3.1. It all starts with the initialisation of the webcam. To mention here again, this is just a prototype. So in the prototype some decisions were made, that might not be feasible in a real system. In the prototype for example the own webcam is started. In a real system, it probably has to be integrated into a online-meeting tool. And the students have to start the webcam on their own. So the video input here is simulated to be the the own webcam. The webcam delivers a video stream as an output to further proceed this video stream is sliced into frames, so the images can be processed. Then the text extraction and comparison is made. The mock database in the prototype is a local folder. First it is tried to simplify the image, by reducing the colors to make the procedure less complex. Then it is looked for contours to draw a rectangle around them. The recognized text is filled into a file and compared to the database. The matches are filled into a file to figure out who is validated. Similar to the text recognition the face recognition works. It has also a mock database with a local folder containing different pictures. This folder is somewhere on the computer, and the path to it, is written in the program. All the pictures in this folder are loaded are encoded and stored in a list. The input from the webcam is used, again it is sliced in frames and encoded. Then the webcam face are compared to the faces in the database. The first match in the list from the folder, will be assigned to the face on the video stream. If there is no matching face in the database/folder the face is classified as unknown. Around the face on the webcam a rectangle is drawn and a name is written to it. At the end of these two tasks to clean up, the video capture has to be released. Those two parts, the face recognition and the text recognition can technically be executed in either order. In the end it has to be compared if the student is correctly validated, everything is fine. Otherwise a manual check is necessary.

3.3.2 Restrictions and Simplification

Since this is only supposed to be a prototype, some simplifications and restrictions had to be made. One very obvious simplification was to design a prototype working on the camera that of the own computer, instead of using for a video conference tool. Implementing a prototype for video conferences directly would put a strain to friendships when regularly trying to see if it works. Another restriction occurs with regard to the databases. Due to privacy reasons the prototype can not access the data from the University System. To model the database part, local folders were created. One is containing images and another folder is containing names and id-numbers. There were made two database models on purpose. In reality those entries might be stored all at the same place. Then the text recognition and the face recognition are made sequentially First the text recognition and then the face recognition. This sequence can be inverted or both parts could made to run in parallel. This was just a decision to make it easier, when implementing. Because the two parts were done separately and then merged together. Also the testing part was very restricted because the Home-office situation due to Covid-19, which was still lasting. And at home the possibilities of varying between faces and UZH-card are usually very limited. Testing was done mostly by pictures. Privacy issue leads also to the conclusion

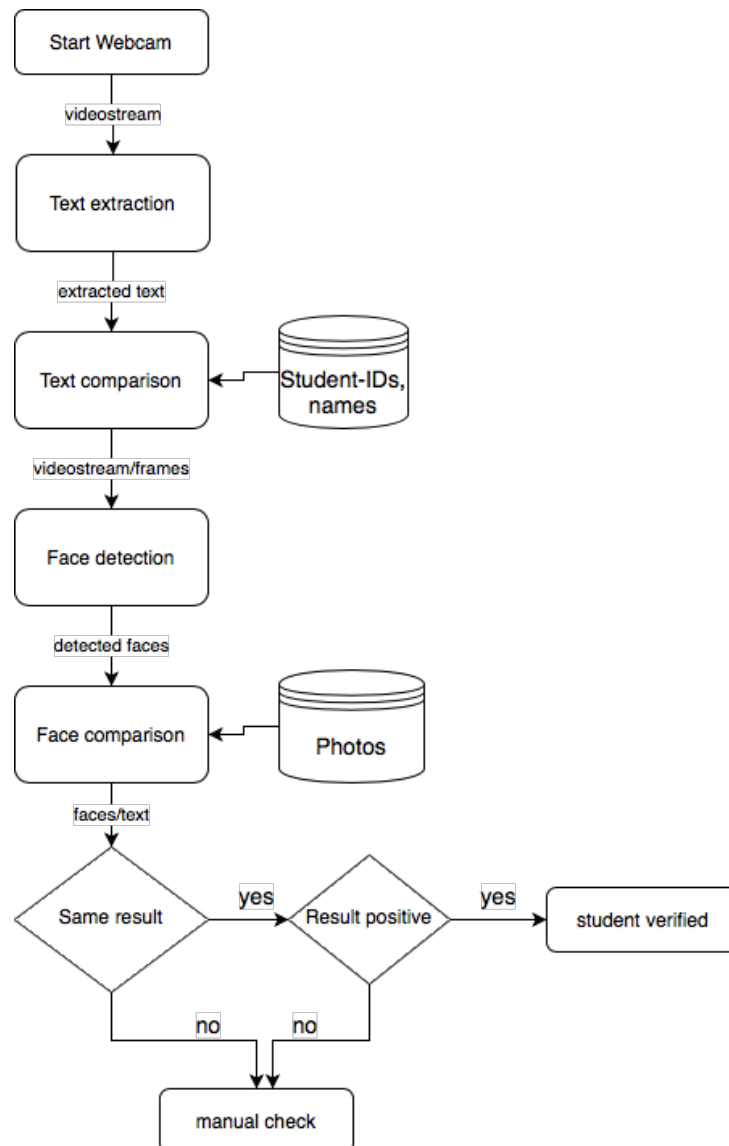


Figure 3.1: This is the design of the prototype, two databases were assumed, one for the student-ids and one for the pictures, as they were done in the prototype. The Interface of the university database is unknown.

that only teaching members of the UZH are allowed to use this system, when it is further developed. Because obviously not everyone should have access to this internal database system.

3.3.3 General Design

As to be seen in Figure 3.2. The first part (1) is starting the webcam. The system here is sequentially as it is initially designed. The first task then is the text recognition (2). For this the Student-id must be shown to the camera(2.1) and videoframes (2.2) for further processing are returned Then the face recognition follows(3). The face is shown to the camera (3.1), the user is probably anyway in front of the camera. Again videoframes

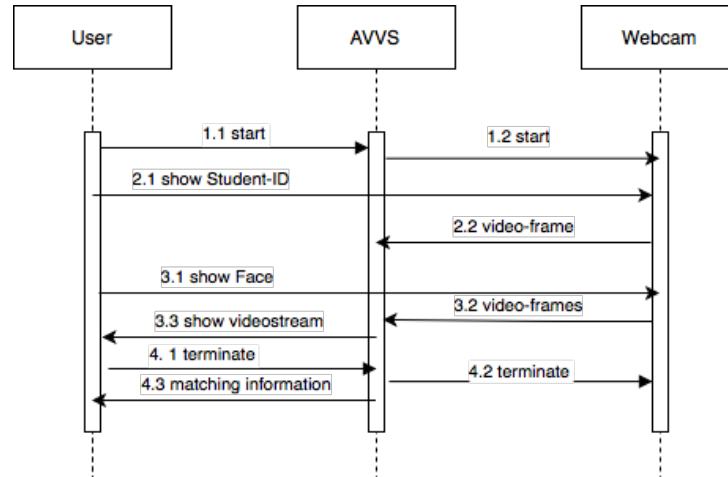


Figure 3.2: This sequence diagram shows the sequence of steps in the system. After the user starts the system and the system starts, the Text recognition (2) starts after the text recognition the the face recognition (3) starts. In the end when the AVVS program ended and the webcam is shut down the matching information is shown.

are returned here (3.2) in this case also a videostream is shown on screen (3.3), this videostream includes a rectangle a name or an "unknown". So it is visible if the person could be assigned to someone in the database. Then the last step is terminating the program, which also shuts down the webcam and prints the matching information.

Text recognition

The first part has to deal with the text recognition. At the first glance text recognition seemed to be easier and also more reliable. One could think that recognizing faces is more difficult than text, since the variations in faces are probably bigger than in text. But it turned out that the Pytesseract library which seemed to be the pretty obvious solution for text recognition is not very reliable in recognizing text. One difficulty of Pytesseract might be the colours. The UZH card does have black font on white or at least bright background, which is a good start. Something else that might cause troubles are the Umlauts. They usually are not recognized, when it comes the student-Id this is not a problem, but if we also want to match with a name, it could be a problem, because in Switzerland there are a lot of people with Umlauts in their name.

Face recognition

The building of the prototype started in two parts. In one part the face recognition has to be done. After the initial research it seemed obvious to go for the Eigenface-method, it used to show rather good result and were not as complicated to implement as some Convolutional Neural network, which would be to excessive for a Bachelor thesis. Then the opportunity of using a OpenCV Library turned up. OpenCV stands for Open Computer Vision Library. It provides algorithms for image processing and computer vision. It is

made for programming languages like C, C++, Java and Python. Which is adequate for this project, since it is supposed to be written in Python. OpenCV is a free library under the condition of BSD (Berkeley Software Distribution) [27].

Matching

There are a lot more string comparisons in this program than expected. Because not only are the text and the face information compared, but each information is also compared to the database. So the the text information is compared to the information in the database text file. If positive, they are marked as validated. The face information, so the name, is likewise compared to the textfile database to see if there is a corresponding name, if so again the information is marked as validated. Then the validated text information and the validated face recognition is compared to see if those information match with each other, this is to ensure the information on the UZH-card matches with the person behind the camera.

Chapter 4

Implementation

In this chapter the implementation is explained. First the architecture is roughly explained. For the start there is something to say about the packages and some other elements that need to be set at the beginning. Then the part of the text recognition will be explained, after that the face recognition and in there is some information about the matching.

4.1 Architecture

In the Figure 4.1 there is a overview on how the implementation is built. First the video-input is started. To process this video-stream it is sliced into frames to treat it like static pictures. Actually both process to this on their own, but since both do it in the beginning this step was put in the common part. Also the processes are sequential, but it does not matter, what process is first and for better understanding a parallel diagram was made. When doing text recognition the first thing to do is change the color system to grayscale. This and the following thresholding is to better distinguish between black and white pixels to get a more precise result. Based on the distinguished pixels between black and white, there are contours made. Because the edges tell, if there is something interesting. When contouring is done and there is some text, the AVVS draws a rectangle around the text to make the ROI visible, where the text is detected. Then the text is written to a file and matched with the information from the database. And the validated information is written in a separate file.

When doing the face recognition which can be done before or after the text recognition. Here the picture from the database are first loaded into an array. Then the faces are encoded and resized for a better processing. The color system from BGR needs to be changed so RGB. Because the videostream default color system is BGR but to process it with OpenCV RGB is needed. Then a rectangle is drawn around the face on the video stream and the face is tried to match with the ones in the database.

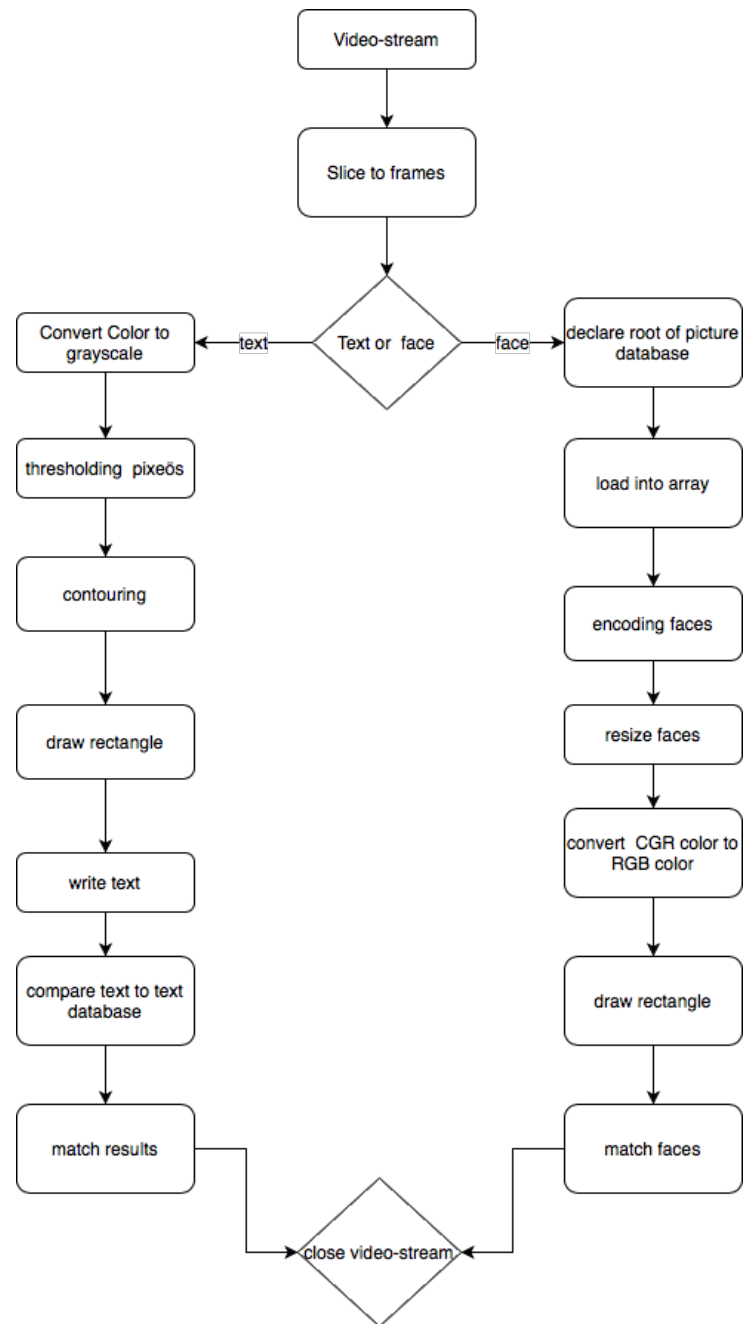


Figure 4.1: Here is a more detailed view to the architecture of the system, even though the system works sequentially the process here is drawn in parallel for a better understanding, because the processes are independent from each other

4.2 Packages

The very first thing to do is importing and installing some packages.

```

1 import pytesseract
2 import face_recognition
3 import os
4 import cv2

```

Pytesseract is obviously for the text processing part and face_recognition for the face recognition. The os package, where 'os' stands for Operation System, is a package, that provides functions for the interaction with the operation system. This is needed in this program, because local folders are used instead of a central database. To access this folders, the package 'os' is used. And cv2 is the package for computer vision, that contains the OpenCV library.

When using Pytesseract it is necessary to tell the system, where your tesseract executables are, otherwise it will not work. So this is an important line in the code. This line has to be adapted, by whoever is going to reproduce this work.

```
1 pytesseract.pytesseract.tesseract_cmd='path to tesseract executable'
```

4.3 Starting webcam

The first step after importing and installing different packages is to open and access the webcam.

```
1 cap=cv2.VideoCapture(0)
2 cap.open(0)
3 ret,image=cap.read()
```

As seen in the code above the OpenCV package is used here for the first time. The cap.open(0) starts the video from the webcam. And the cap.read() returns two values. One of them is ret, which is a boolean telling if there is a return value. And the frame is the frame, returned. This is because operations directly on the video are not possible, so the video is cut into frames to operate on the images.

4.4 Text recognition

```
1 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Then the color of the frames is converted to gray scale. This conversion is recommended because of different reasons. Color information is not always necessary for applications of image processing. Processing gray-scale images is usually less complex than colored pictures. Less complex means also less computing power is necessary. At this point only text is processed. So it is sufficient to divide the picture in black and white.

```
1 ret, thresh1 = cv2.threshold(gray, 0, 255,
2 cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
```

A thresholding is done, to determine if a pixel is black or white. The Otsu binarization, tries to minimize the within-class variance to determine the threshold [28].

Next step are some operations on the forms we have. With getStructuringElement we can get a rectangular shaped kernel. So far a lot of noises has been removed, this leads also to potential characters being shrunk. To increase the area of our object, the dilate function is applied.

```

1     rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (18, 18))
2     dilation = cv2.dilate(thresh1, rect_kernel, iterations = 1)

```

From the dilated frame, the contours are wanted. Besides the before defined dilation, also the contour retrieval mode and the contour approximation are passed as argument.

```

1     contours, hierarchy = cv2.findContours(dilation,
2     cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

```

The output hierarchy is no longer used. The output contours returns a python list of all contours in the examined frame. Every contour is a numpy array of coordinates of boundary points of the object [29]. Through this list called contours, an iteration is done.

```

1     for cnt in contours:
2         x, y, w, h = cv2.boundingRect(cnt)

```

In this iteration x and y denote the the lower-left coordinate of the rectangle and w and h denote the width and the height.

```

1     rect = cv2.rectangle(image2, (x, y), (x + w, y + h), (0, 255, 0), 2)
2     cropped = image2[y:y + h, x:x + w]

```

The rectangle method draws an rectangle around the image. The parameter image2 is called image2 because the image has been copied. In the following line, the image is cropped to feed it to the OCR.

```

1     file = open("recognized.txt", "w+")
2     file.write("")
3     file.close()
4
5     file1 = open("recognized.txt", "a")

```

A file is created, called *recognized.txt*, everything the OCR recognizes is written in this file. This is done in the following paragraph.

```

1     text = pytesseract.image_to_string(cropped)
2
3     file1.write(text)
4     file1.write("\n")
5     file.close

```

Then the other text file called *MatrikelNr.txt* is open. This is supposed to model the database. Then those two textfiles compare line by line and put the lines which are equal to each other in the text and write it to a file called *validated_text.txt*. On the UZH-card might be more than one vital information for the AVVS in one line. Therefore the information is also compared element-twise.

```

1     with open('MatrikelNr.txt', 'r') as file2:
2     with open('recognized.txt', 'r') as file1:
3         same=set(file1).intersection(file2)
4
5     same.discard('\n')
6     with open('validated_text.txt', 'w') as file_out:
7         for line in same:
8             file_out.write(line+'\n')
9     with open('MatrikelNr.txt', 'r') as file2:
10    with open('recognized.txt', 'r') as file1:
11        list2=file2.read()
12        list2=list2.split()
13        list1=file1.read()
14        list1=list1.split()

```

```

15         same=set(list1).intersection(list2)
16         same=list(same)
17         for elem1 in list1:
18             for elem2 in list2:
19                 if elem2 == elem1:
20                     same.append(elem1)

```

4.5 Face recognition

Again the camera is started. Then some empty lists are initialised.

```

1     cap =cv2.VideoCapture(0)
2
3     if not cap.isOpened():
4         raise IOError("Cannot open webcam")
5
6     known_face_encodings =[]
7     known_face_names=[]
8     user_appeared=[]

```

The variable root is defined to be where the mock picture database is. It is iterated through this database and all files that ends with the ".jpg" or ".png" are extracted, so all pictures. For all pictures in this folder the path is declared.

```

1     root= "path to photos"
2     for filename in os.listdir(root):
3         if filename.endswith('.jpg' or '.png'):
4             try:
5                 path =os.path.join(root, filename)
6                 filt_img = face_recognition.load_image_file(path)
7                 filter_face_encode= face_recognition.face_encodings(filt_img)
8                 known_face_encodings.append(filter_face_encode[0])
9                 known_face_names.append(filename)
10            except:
11                print("An exception :"+ filename)
12    print(known_face_names)

```

In this iteration the face_recognition package is used. In the load_image_file the image files are loaded into an array. This array is fed to the function face_encodings, which returns the 128-dimension face encoding for each face in the image [31]. Those encoded faces are then appended to one of the lists initialised before. If this procedure does not work a message is returned, telling that there is an exception. Then the list with the names of the known faces is printed. Before defining a function, some variables are initialised to use later on.

```

1     face_location=[]
2     face_encodings=[]
3     face_names=[]

```

The face recognition is built as a function.

```

1     def face():
2         while True:
3             process_this_frame = True
4             ret, frame = cap.read()
5             small_frame = cv2.resize(frame, None, fx=0.5, fy =0.5,
6                                     interpolation=cv2.INTER_AREA)
7             rgb_small_frame= small_frame[:, :, ::-1]

```

At the very beginning of this process there was something quite familiar, the `cap.read()` in the text recognition before, here a single frame is grabbed to process, because the video can not be processed as such. Then the frame is resized to 1/2, this should improve the face recognition. OpenCv uses the BGR color, but `face_recognition` uses RGB color. So the color format has to be converted to further proceed.

```

1      c=cv2.waitKey(1)
2      if c==27:
3          break
4
5      if process_this_frame:
6          face_locations = face_recognition.face_locations(rgb_small_frame)
7          face_encodings = face_recognition.face_encodings
8                          (rgb_small_frame, face_locations)
9          face_names=[]
10         for face_encoding in face_encodings:
11             matches = face_recognition.compare_faces
12                     (known_face_encodings, face_encoding)
13             name="Unknown"
14
15             if True in matches:
16                 first_match_index= matches.index(True)
17                 name=known_face_names[first_match_index]
18                 print(name)
19             face_names.append(name)

```

The `cv2.waitKey` method is a keyboard binding function. The argument passed in the parenthesis is the time in milliseconds, that the method waits for any keyboard event. If zero is passed it waits indefinitely for a key stroke [32]. Here 1 is passed as an argument. In the following line `c==27` is set, 27 is the Ascii Code for ESC, so to interrupt the program, the user can hit ESC on the keyboard.

In the code snippet before a variable called `process_this_frame` was set as `True`. Now this is used as condition for an if statement. In the next two lines all faces and face encoding is collected. Then in the for loop over all the faces listed in `face_encoding`, there is a check, if there is a match in the face database. The first name in the `known_face_names` is assigned to the face and printed. Names that match with the database are also written into the *validated_face.txt* file, to later on compare it to the information we got from the text recognition. This process can be seen in the following code. Then the value of `process_this_frame` is changed to `False`, to not get stuck in the loop.

```

1      with open('MatrikelNr.txt', 'r') as file1:
2          name=name.replace("-", " ")
3          name=name.replace(".", " ")
4          name=name.split()
5          list1=file1.read()
6          list1=list1.split()
7          compared=set(list1).intersection(name)
8          compared.discard('\n')
9      with open('validated_face.txt', 'w') as file_out:
10         file_out.write('Face recognition: '+ '\n')
11         for k in compared:
12             file_out.write(k + '\n')
13     process_this_frame= not process_this_frame

```

At the end of this function there is a for-loop.

```

1     for (top, right, bottom, left), name in
2         zip(face_locations, face_names):
3         top *=2
4         right *=2
5         bottom *= 2

```

```

6     left *=2
7     #Draw a box around the face
8     cv2.rectangle(frame, (left, top), (right, bottom), (0,0,255), 2)
9     # Draw a label with a name below the face
10    cv2.rectangle(frame, (left, bottom -35),
11                  (right, bottom), (0,0,255), cv2.FILLED)
12    font = cv2.FONT_HERSHEY_DUPLEX
13    cv2.putText(frame, name, (left +6, bottom -6),
14               font, 1.0, (255,255, 255), 1)
15    #Display the resulting image
16    cv2.imshow('Video', frame)
17    #Hit 'q' on the keyboard to quit!
18    if cv2.waitKey(1) & 0xFF == ord('q'):
19        break

```

The face location is scaled up back to a normal size. It was shrunk to half the size. Now every side of the rectangle is multiplied by two. To mark the face found in the frame a rectangle is drawn around it by the `cv2.rectangle` method, with the `cv2.putText`, also the name is added to the frame. The `cv2.imshow` displays an image in a window, where the window size automatically fits to the image size [32]. To stop this window again, there is a `cv2.waitKey` method defined as above, but here the key "q" is used to stop the window.

```

1 cap.release()
2 cv2.destroyAllWindows()

```

In the end of this function called `face`, there is a `cap.release()` to do, this is to release resources. With `cv2.destroyAllWindows()`, all created windows are destroyed. To get the face recognition started, just the function `face()` has to be called.

4.6 Matching

So to get a clear result, there needs to be checked, if there are matches between text and face recognition. The *validated.txt* files are opened and the information in there are read. To have a better overview, there were each an validated file created, there is a *validated_face.txt* and a *validated_text.txt*. This makes clear, where to find what information and also it makes it easier to compare those two lists. The information from both lists is searched for matches. Those matches are printed in the end, to make it visible, if there are any corresponding entries. If there are no corresponding entries an empty list will be printed. To not only compare lines but also elements in the lists, there is not only an intersection made but also an element wise comparison.

```

1 fileText1 = open("validated_text.txt", "r")
2 fileText2 = open("validated_face.txt")
3 data1 = fileText1.readlines()
4 data2= fileText2.readlines()
5
6 match=set(data1).intersection(data2)
7 match=list(match)
8
9 for elem1 in data1:
10     elem1 = elem1.replace("\n", " ")
11     elem1=elem1.split()
12     for elem2 in data2:
13         elem2=elem2.replace("\n", " ")
14         if elem2 in elem1:
15             match.append(elem1)
16

```

```
17 print("Matches are:")
18 print(match)
```

This is because of the text recognition, there is a possibility, that there is more than one word in one line. And the intersection just checks for identical strings. If there are two words in one line it doesn't match with a an element in the face recognition, where every word is on a new line. To catch bot possible outcomes, there are two methods used to get the matches. The resulting elements are printed as "Matches are:" and then followed by either an empty list or the matches found in the lists.

Chapter 5

Evaluation

In this chapter it is checked, if the aforementioned requirements are fulfilled. Then, the precision of the recognition is examined, and in the end, two test cases are shown.

5.1 Functional Requirements

There were the following functional requirements:

1. A list of students including name and student ID should be accessed.
2. Information from the UZH-card should be read by the AVVS.
3. The information from the UZH-card must be compared to the information on the list with names and IDs.
4. The AVVS has to compare the face on the webcam to the photo on the database/-database replacement and tell if it is the same person
5. The AVVS makes clearly visible if a student is accepted or not.

Based on the architectural details described in Chapter 4, the following requirements are satisfied by design:

1. **A list of students including name and student ID should be accessed**

The importance of this point was a bit a victim of the restrictions. There is a list of students stored. This file was stored in the the program file, where also the code is. Alternatively, the list could be placed somewhere in a local folder and the AVVS accesses it over the path. Anyhow, this might be done different when finally implementing with access to the real system which may include design and implementation details often not revealed in a proof-of-concept system.

2. Information from the UZH-card should be read by the AVVS

This requirement is fulfilled. Unfortunately, the result by pytesseract is not as good as expected. The performance depends on the light and on how close and how the UZH-card is hold against the camera. For example, depending on a distance that the card is shown and the quality of the camera on the student's side, card information may not be detected. Also, information may not be retrieved depending on how fast the card is displayed. In this case, a person verifying the student's ID could promptly ask to repeat the procedure until the required information is retrieved.

3. The information from the UZH-card must be compared to the information on the list with names and IDs.

This requirement is fulfilled, but following from the above it depends on the output quality from the information read from the UZH-card. When this information is correct the comparison is not a problem.

4. The AVVS has to compare the face on the webcam to the photo on the database/-database replacement and tell if it is the same person

This requirement is fulfilled. However, there was a small change due to the assumption, that the pictures of the students is also stored in the UZH database. There was a local folder with pictures created. Hence, the face on the webcam is compared to the photo in the local folder which would correlate with the UZH-database, which would again correlate with the picture on the UZH card.

5. The AVVS makes clearly visible if a student is accepted or not.

On the one hand, there is a result of the text recognition, which states whether or not the student is validated by the Student-id. On the other hand the student is or is not recognized by picture. Those results are printed into the *validated.txt* files. There are two lists for validated information one for the text and one for the face. Those two lists are read and it is compared, which information is in the text and in the face recognition information present, and prints matches. Thus, one is able to check whether there are matches between face and text.

5.2 Precision

The precision of the AVVS is dependent on various things. One could think face recognition is more difficult and sensitive than text recognition since there are more possible variations. However, such assumption does not hold since the text recognition with pytesseract has a limited precision. It is sensitive to the light effect into the camera, meaning that if there is too little light it does not work. Also the UZH-card or the paper with the text written on it has to be hold calm and close to the camera. Otherwise, the text is not recognized, all the attempts to improve this issue failed. Whereas the face recognition part is working rather reliable. It recognizes faces under different angles and light conditions and even when moving. The difficulty in text recognition can also lead to imprecision in the result of matching text and faces.

5.2.1 Precision Face recognition

In twenty of twenty attempts a person was correctly recognized, but it would be interesting to know more about the matching process. There is a built-in method in the package `face_recognition` called `face_distance`. This seemed appropriate to investigate the precision of the face recognition.

```
1 face_distance = face_recognition.face_distance(known_face_encodings , face_encoding)
2 inv_face_distance=1-face_distance
```

The `face_distance` tells how different two faces are. The threshold for a match is with a face distance lower than 0.6 [42]. The face distance was inverted in the code to have an indication for the similarity. In *the photo* folder, the substitute database, are nine pictures. When doing the face matches, the system can simultaneously calculate the face distance, and followed from there also the inverse face distance. To illustrate this, there is a list below. The face on the webcam is compared to each face in the folder, so every number returns a value about the similarity to each picture in the folder.

```
1 [0.13149432 0.28197851 0.2561369 0.17975971 0.55643808 0.22756177
2 0.25237962 0.24049715 0.24541102]
```

It is clearly visible, that the fifth picture is the match, since it is the only one having a similarity higher than 0.4. In the *photo* folder are four photos of the same person. on the list below the equivalent list is shown for the case, where this persons has to be identified.

```
1 [0.16305194 0.27651531 0.53172272 0.29705242 0.20092066 0.59533536
2 0.45755746 0.32819369 0.61331698]
```

The photos of the person are the numbers three, six, seven and nine. They are all clearly bigger than 0.4, what would be the inverse threshold. The match that is chosen is the number three, even though the last picture shows higher similarity. So the system picks the first match that hold to the condition. The matching could be set to the best match, but in this system, there was hardly ever a face mismatched so there seems no need to be more strict on the matching.

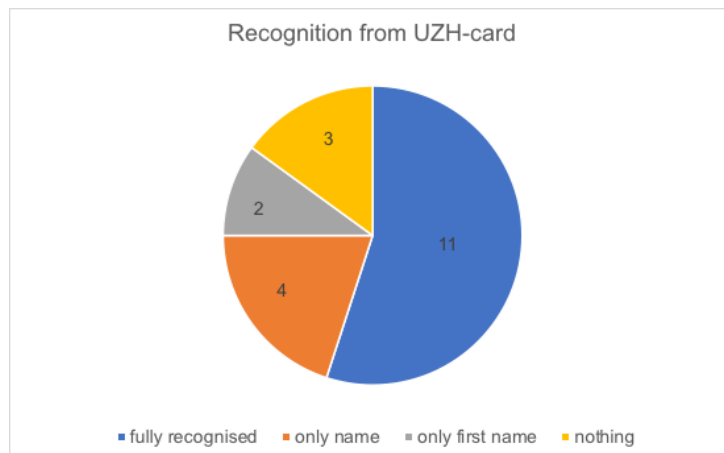


Table 5.1: Graphic representation of the results of the text recognition from the UZH-card

5.2.2 Precision Text recognition

For text recognition, a test was made, with showing the UZH card 20 times to the camera as to be seen in the Table 5.1. In eleven cases the name and the Student-Id and even further information like birthday date or validity date were correctly. In four cases there was only the name recognized, and the Student-id recognized was only similar to the Student-Id on the UZH-card, but not the same. In three cases there was only the first name recognized. And three times nothing was recognized. It is not straightforward to determine if this was coincidence or if the person holding the card was shaking in that moment, or why three times there was nothing recognized. However, to state a contrast, there was a fictional name and a fictional Student-id written on a piece of paper to check how it works there. Therefore, font size 14 was used, black color on white background.

As expected the results turned out better than on the UZH-card, since there was less writing on it and the writing was bigger and probably also clearer than on the UZH-card, this is illustrated in the Table 5.2. In twenty trials 15 were fully successful. In two tries nothing was recognized and in two other tries the name was recognized but to the Id-number was a number added. Lastly, in one was only a part of the name and a part of the number recognized, this might have been caused by holding the paper in a bad angle to the webcam.

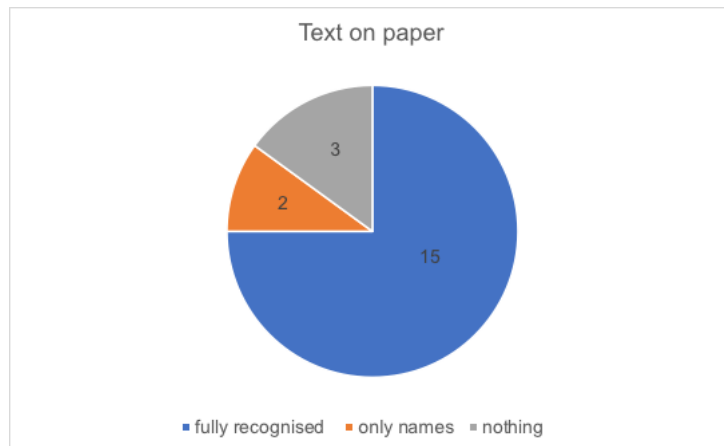


Table 5.2: Graphic representation of the results of the text recognition from black text on white paper

5.3 Test cases

5.3.1 Negative example

In one use case it is to be tested whether an unknown person is not identified to reassure that an unknown person is not falsely identified. For that reason some face, which is not in the database is shown to the camera. To prove that, two generic faces from a language book were taken [34]. As seen in Figure 5.1, those two faces are marked by the system

as unknown. This "unknown" does not generate entries in the validated file for the face recognition.

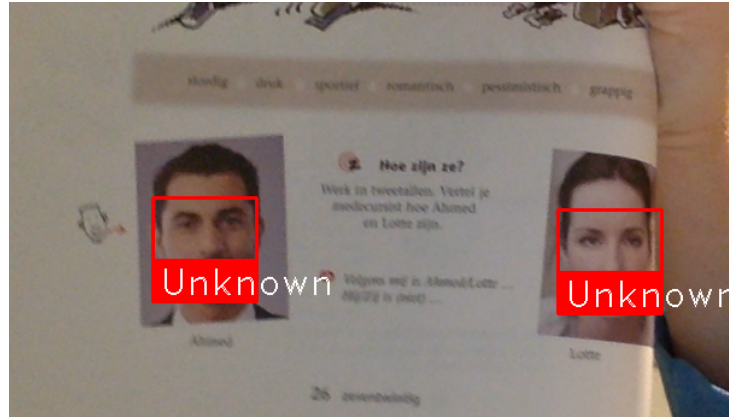


Figure 5.1: These are two faces from a language book, which are not contained in the database.[34]

Also, it is necessary to check on what happens when there is a non-valid ID shown. For that reason an invented Student-id and an invented name, which were not in the database, were shown to the camera, which lead to the result in having an empty *validated.txt* file.

5.3.2 Positive example

In contrast to the previous use case, a positive (*i.e.*, detected face and text match information stored at the database) example had to be tested. For this case, first a picture of the person to be recognized has to be inserted into the database. To make it realistic, a different picture was inserted into the database, than was shown to the camera. The picture inserted into the database is in Figure 5.2.

Then, there was a picture with the same person shown to the camera to test if it was recognized. As seen in Figure 5.3, the face is recognized as the same person. Since it is recognized, the name of the picture was extracted and written into a text file for validated faces.

The recognition from this face extracts the name, because the name of the picture is the name of the person. Then, this name is compared to the list of students, if there is a match.

If there is a match not only with the database, but also a match between the name and the list of students. Then, this name is also considered as validated this is visible in Figure 5.4.

Then, there also have to be tested, how it is about text recognition. To check on the text recognition, it is required to insert a name and student Id into the database. For this name and this student-id, there is no UZH-card available. Thus, this information was printed on a paper to make the test on the text recognition. As observed in Figure 5.5,



Figure 5.2: This was the face inserted into the database for testing purposes [35].



Figure 5.3: This face was correctly recognized to be the same person as a photo stored in the database.

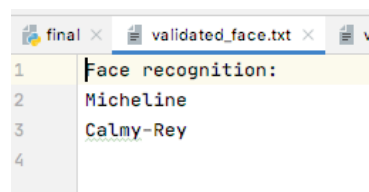


Figure 5.4: The name of this face is compared to the list of students and the result shows the name is contained in this list.

the text is not only recognized but also validated, as the same information is present in the *MatrikelNr.txt* file.

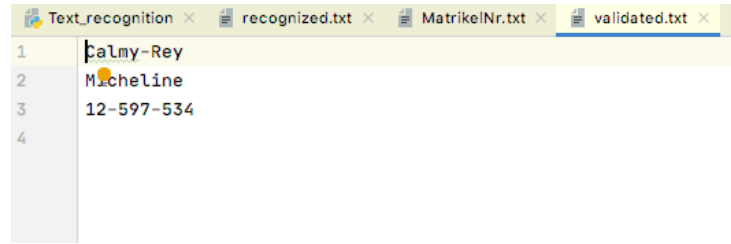


Figure 5.5: This text printed on a paper, to replace an UZH-card, with fictional information about the person, was validated as the same information is also in the database.

The text should be recognized and also considered to be valid. If this is the case it is first loaded into the file for recognized text and after a comparison with the database material considered as validated. Then, the both validated information from the text and the face part, are compared, with each other to check if there is a match. And in this case (*cf.* Figure 5.6) it happens to be a match and the person is verified and validated.

```
Matches are:
['Calmy-Rey\n', 'Micheline\n']
```

Figure 5.6: The information from the face and the text recognition have a match, so the person in front of the camera is identical to the person on the card/substitute.

5.4 Discussion

In general, the objective of this thesis has been achieved. There is a prototype, which perform text recognition, face recognition and analyzes whether they are valid or not according to information previously stored at a database. Hence, the overall goal is achieved, but maybe also the quality of the prototype should be assessed. The topic of precision has already been discussed. What has indeed room for improvement would be the structure of the code. For one thing would be the text recognition and the face recognition could have been similarly structured. The prototype grew rather organically from the tutorials used. This lead to a text recognition, which is working in a short time, because there is basically one frame analyzed, whereas the face recognition is in a stream and the face recognized in the last frame is taken for the matching. Also, the face recognition is a function and the text recognition is written in the script and not as a function. Also, the AVVS could run in parallel instead of sequentially. Something like threading could be done on the AVVS to have it in parallel.

Because of this sub-optimal structuring the user, to be verified, does not exactly know how long he has to hold the UZH-card to the camera. Thus, the User, who needs to be verified, has to wait until they sees the face recognition starts, which is visible, when

the names from the picture database are listed. Nonetheless, again this listing of the database names would actually not be necessary. The prototype may not be perfect, but it is working and the focus was clearly set on working code instead of well structured and documented code.

Chapter 6

Conclusions

In this last chapter there is time to look back on the project and its process. After deducting conclusions, there is an overview on the obstacles and what can be learnt from them and then have a short view at possible future work.

6.1 Conclusion

Usually in projects at the beginning there is just a big picture of how the project should be built. While developing, new approaches and ideas are coming up and sometimes it happens that something does not work out as planned or is not as necessary as initially thought. This was also in this project the case. In the beginning there was a vague idea on how to proceed. When beginning it turned out, that it is not necessary to check the validation date on the UZH card, because on one hand students can not be enrolled in a course without being valid students (which means they paid their semester fee). On the other hand when a pandemic situation is lasting students might not be able to go to university to validate their card.

Another thing that changed in this process was the methods used. When doing research on the existing work, it seemed clear what method would be appropriate for face recognition: the Eigenface method. This was a method where the most information could be found, and the results were mostly positive. Due to a lack of experience in the respective field at the beginning, the implementation process was accompanied by reading and following tutorials on this matter. When looking through tutorials it turned out, that for the image recognition part the OpenCV library was highly recommended and it delivered satisfying results. OpenCV was however not very often mentioned in the scientific research paper, when talking about face recognition.

Whereas for the text recognition it was clear to use the pytesseract module, based on the tesseract engine, because there are no significant other text recognition engines. Unfortunately, the result of pytesseract was not as convincing as the results of the face recognition. And the attempts to improve failed. So the text recognition works only under the best

conditions in terms of light, position and movement. It was surprising that the text recognition is less reliable than the face recognition, because face recognition was supposed to be more complex.

In the end the most tiring part was to match the results together, to get the final information to the user. This was less exciting than the face and text recognition. The topic of thesis was chosen, because of the interest in Face and Text recognition. There was a lot of string comparison and writing from files and to files. But in the end the prototype works and returns results as initially wanted, so the main goal was achieved.

6.2 Obstacles

6.2.1 String comparison

Besides the mediocre reliability of pytesseract, the biggest obstacle was the string comparison in the end. Because for one thing the information from text and face recognition, which also returns the name of the person, had to be compared to a database containing student-ids and names. And then again it has to be checked if the text recognition and the face recognition deliver intersecting results. This process makes it very easy to lose the overview and increases the chance to produce verbose code. Also having at most one word per line in face recognition and having more than one word per line in the text recognition, makes it again more complicated to compare those two lists. And also the data type has to be changed to get all the results.

6.2.2 Code quality

As already indicated before, the coding structure could be improved. The highest priority was to have working code, that delivers results. And there were modest efforts in improving the code quality, because the result counts, and improvements in performance were more interesting or more important than the code quality. However when there would more time or there would be a follow-up project to do, there should be paid more attention to the code quality. This is also a take-away from this work for future projects, to take a more considered approach when coding regarding the code quality.

6.2.3 Time management

One of the most difficult issues during the elaboration of this thesis, was the time management. In the beginning there was a more or less clear time plan. As usual in projects this time schedule was rather guideline than a real plan. The interest in image recognition was the reason this particular topic was chosen. And there was a positive surprise when leaving the area of scientific papers, because there were a lot of different tutorials to be found on this issue. However to dive really deep into image recognition, there should be a lot more time invested.

6.2.4 Coding vs writing

The beginning of such a project is usually clear, it is writing, about the motivation, about papers on the issue and about methods and concepts. At some point the coding has to begin, and it was more difficult than expected to find a good balance between writing and coding, because in the end there is the prototype and then go back to writing about all the things is somehow more difficult than expected. Because some thoughts about what decisions or insights are made, while coding can go lost. Nevertheless until the last week there were changes in the code made, to get more clear results. This also felt stressful so maybe this is also not the best option. Perhaps dividing the time more thoughtful between coding and writing should be considered in such a project.

6.3 Future Work

To get back to the initial idea, the system is supposed to verify students in a remote exam or online meeting. This prototype works with its own webcam. So the prototype could be extended to work in online meetings to recognize the other participants. And not just one self. For the the prototype described in this thesis a lot of restrictions and assumptions had been made. For example the databases were just local folders, instead of real databases. This is also due to privacy reasons, because the UZH database, which would have been necessary, contains obviously personal data, which cannot be accessed by everyone. So creating the interface to the UZH database would be a possible way to build up on this work. There could also a front-end be built, to show if there is a match or not.

Other ways to improve the prototype would be improve the performance of it. The face recognition uses quite a lot of computing power, so increasing the computer power would improve the performance of the system. Also there might be some possibilities in increasing the text recognition performance. Then of course there would be the option of restructuring the code in a better way.

Bibliography

- [1] Jain, A. K., & Yu, B. (1998). Automatic Text Location in Images and Video Frames. *Pattern recognition*, 31(12), 2055-2076.
- [2] Mithe, R., Indalkar, S., & Divekar, N. (2013). Optical Character Recognition. *International Journal of Recent Technology and Engineering (IJRTE)*, 2(1), 72-75.
- [3] Siswanto, A. R. S., Nugroho, A. S., & Galinium, M. (2014, September). Implementation of Face Recognition Algorithm for Biometrics based Time Attendance System. In *2014 International Conference on ICT For Smart Society (ICISS)* (pp. 149-154). IEEE
- [4] Pissarenko, D. (2002). Eigenface-based facial recognition. December 1st.
- [5] M. A. Turk and A. P. Pentland. Face Recognition using Eigenfaces. In *Proc. of Computer Vision and Pattern Recognition*, pages 586–591. IEEE, June 1991b.
- [6] Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using Class Specific Linear Projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7), 711-720
- [7] R. Smith. "An overview of the Tesseract OCR Engine", *Proc. 9th Int. Conf. on Document Analysis and Recognition*, IEEE Curitiba, Brazil, Sep 2007, pp629-633
- [8] Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1), 98-113.
- [9] D. DeMers and G. W. Cottrell, "Nonlinear dimensionality reduction," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. Lee Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 580–587.
- [10] Kanade, T. (1974). Picture processing system by computer complex and recognition of human faces.
- [11] Brunelli, R., & Poggio, T. (1993). Face recognition: Features versus templates. *IEEE transactions on pattern analysis and machine intelligence*, 15(10), 1042-1052.
- [12] Shan, S., Cao, B., Gao, W., & Zhao, D. (2002, May). Extended Fisherface for face recognition from a single example image per person. In *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353)* (Vol. 2, pp. II-II). IEEE.

- [13] <http://openimaj.org/tutorial/eigenfaces.html>, 30.6.2020.
- [14] Georgescu, D. (2011). A real-time face recognition system using eigenfaces. *Journal of Mobile, Embedded and Distributed Systems*, 3(4), 193-204.
- [15] Abdi, H., & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4), 433-459.
- [16] Balakrishnama, S., & Ganapathiraju, A. (1998, March). Linear discriminant analysis-a brief tutorial. In *Institute for Signal and information Processing* (Vol. 18, No. 1998, pp. 1-8).
- [17] Rodrigues, Bruno, Thomas Bocek, Andri Lareida, David Hausheer, Sina Rafati, and Burkhard Stiller. "A blockchain-based architecture for collaborative DDoS mitigation with smart contracts." In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pp. 16-29. Springer, Cham, 2017.
- [18] Bocek, T., Rodrigues, B. B., Strasser, T., Stiller, B. (2017, May). Blockchains everywhere-a use-case of blockchains in the pharma supply-chain. In *2017 IFIP/IEEE symposium on integrated network and service management (IM)* (pp. 772-777). IEEE.
- [19] Cendrillon, R., & Lovell, B. (2000, May). Real-time face recognition using eigenfaces. In *Visual Communications and Image Processing 2000* (Vol. 4067, pp. 269-276). International Society for Optics and Photonics.
- [20] Ö. Toygar, A. Acan, Face Recognition Using PCA, LDA And ICA Approaches On Colored Images, *Journal Of Electrical & Electronics Engineering*, Vol.3, No.1, pp. 735-743, Turkey, 2003.
- [21] Reisfeld, D., & Yeshurun, Y. (1998). Preprocessing of face images: Detection of features and pose normalization. *Computer vision and image understanding*, 71(3), 413-430.
- [22] <https://pypi.org/project/pytesseract/>, 23.11.2020.
- [23] <https://github.com/tesseract-ocr/>, 14.7.2020.
- [24] Lienhart, R. W., & Stuber, F. (1996, March). Automatic text recognition in digital videos. In *Image and Video Processing IV* (Vol. 2666, pp. 180-188). International Society for Optics and Photonics.
- [25] Qadri, M. T., & Asif, M. (2009, April). Automatic number plate recognition system for vehicle identification using optical character recognition. In *2009 International Conference on Education Technology and Computer* (pp. 335-338). IEEE.
- [26] S. L. Horowitz and T. Pavlidis, "Picture Segmentation by a Traversal Algorithm", *Comput. Graphics Image Process.* 1, pp. 360-372, 1972
- [27] <https://opencv.org/about/>, <https://opencv.org/about/> 19.10.2020.
- [28] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html, Datum.

- [29] <https://www.geeksforgeeks.org/find-and-draw-contours-using-opencv-python/>, 20.10.2020.
- [30] <https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>, 24.10.2020.
- [31] https://face-recognition.readthedocs.io/en/latest/face_recognition.html, 30.10.2020.
- [32] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html, 30.10.2020.
- [33] Gruhler, Andreas, Bruno Rodrigues, and Burkhard Stiller. "A reputation scheme for a blockchain-based network cooperative defense." 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2019.
- [34] Schneider-Broekmans, J., Fox, S., Morsbach, B., & Kennedie, C. (2010). Taal vi-taal; Nederlands voor beginners/door Josina Schneider-Broekmans: concept en red.: Stephen Fox; met medew. van Bernd Morsbach... et al.; ill.: Ofzcarek!... et al.; Ned-erlandse bew.: Caroline Kennedie; met medew. van Lidy Zijlmans... et al. Intertaal.
- [35] https://de.m.wikipedia.org/wiki/Datei:Micheline_Calmy-Rey,_2009.jpg, 16.Nov 2020.
- [36] <https://www.aargauerzeitung.ch/schweiz>, 16.Nov 2020.
- [37] , <https://realpython.com/face-recognition-with-python/>, 19.11.2020.
- [38] <https://github.com/shantnu/FaceDetect/>, 19.11.2020.
- [39] <https://realpython.com/face-detection-in-python-using-a-webcam/>, 19.11.2020.
- [40] <https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr/>, 19.11.2020.
- [41] <https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/>, 19.11.2020.
- [42] https://github.com/ageitgey/face_recognition/wiki/Calculating-Accuracy-as-a-Percentage, 25.11.20.
- [43] <https://stackoverflow.com/questions/41688849/opencv-python-how-to-save-name-of-the-recognized-face-from-face-recognition-pro>, 31.8.2020.

Abbreviations

AAA	Authentication, Authorization, and Accounting
ANPR	Automatic Number Plate recognition
AVVS	Automated Video UZH-ID Verification System
CNN	Convolutional Neural Networks
FMR	False Match Rate
FNMR	False Non-Match Rate
ICA	Independent Component Analysis
LDA	Linear Discriminant Analysis
OCR	Optical Character Recognition
PCA	Principal Component Analysis
ROC	Receiver Operating Characteristic
ROI	Region of interest

List of Figures

3.1	This is the design of the prototype, two databases were assumed, one for the student-ids and one for the pictures, as they were done in the prototype. The Interface of the university database is unknown.	19
3.2	This sequence diagram shows the sequence of steps in the system. After the user starts the system and the system starts, the Text recognition (2) starts after the text recognition the the face recognition (3) starts. In the end when the AVVS program ended and the webcam is shut down the matching information is shown.	20
4.1	Here is a more detailed view to the architecture of the system, even though the system works sequentially the process here is drawn in parallel for a better understanding, because the processes are independent from each other	23
5.1	These are two faces from a language book, which are not contained in the database.[34]	34
5.2	This was the face inserted into the database for testing purposes [35]. . . .	35
5.3	This face was correctly recognized to be the same person as a photo stored in the database.	35
5.4	The name of this face is compared to the list of students and the result shows the name is contained in this list.	35
5.5	This text printed on a paper, to replace an UZH-card, with fictional information about the person, was validated as the same information is also in the database.	36
5.6	The information from the face and the text recognition have a match, so the person in front of the camera is identical to the person on the card/-substitute.	36

List of Tables

2.1	The PCA, LDA and ICA are compared, PCA in terms of the threshold and the success rate with two numbers of training faces [20]	10
5.1	Graphic representation of the results of the text recognition from the UZH-card	32
5.2	Graphic representation of the results of the text recognition from black text on white paper	33

Appendix A

Technical Guidelines

Some things need to be set up to use the prototype.

A.1 Text recognition

For the text recognition the path to the root folder containing the pytesseract executable needs to be set. On Mac OS the path to the executables can be set with the following command.

```
1 brew list tesseract
```

Then this path needs to be inserted here.

```
1 pytesseract.pytesseract.tesseract_cmd='path to pytesseract executables'
```

There was a local text file called *MatrikelNr.txt* used as database. This was locally created and included in the IDE.

A.2 Face recognition

As picture database a local folder was created. The path to this folder was defined in the code as root.

```
1 root= "Path to picture database"
```

When reproducing this, please consider that the photos were named after the person visible on the photo. If they are named otherwise the AVVS will not be able to perform a matching.

The face recognition accesses the *MatrikelNr.txt*, too.

With the following line of code the default camera of a computer is accessed, if any other camera is used this has to be specified first.

```
1 cap =cv2.VideoCapture(0)
```

Appendix B

Contents of the Hand-in

- AVVS Source Code
 - Text recognition
 - Face recognition
 - Information match
 - Final version
- LaTeX Source Code Thesis
- Thesis PDF