STATE COMMITTEE OF COMMUNICATION, INFORMATIZATION AND TELECOMMUNICATION TECHNOLOGIES OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES

УДК 004.93

UMAROV ABROR BAKHTIYOROVICH

Methods of image recognition using template matching algorithm

5A330202 –Information and Multimedia Technology

DISSERTATION

For the master's academic degree

Research supervisor

d.ph.m.s., professor

Aripov M.M.

Tashkent – 2014

Content

Introduction

Subject topicality. The government of Uzbekistan considers that implementation and usage of information-communication technologies in the different spheres of socially-economical life of the society is a high-priority task. Starting from the independence, state leaders paying a high attention for the improvement of the sphere of communication service, as well as, for providing its intensive development. In accordance with the Law of the Republic of Uzbekistan «on informatization», in order to improve efficiency of the activities of the Ministries of government and economy control, field state authorities, to provide with wide usage of modern information-communication technologies in the sphere of government and social development by the President of the Republic of Uzbekistan on March 21, 2012 has been adopted a decree «On measures of further implementation and development of modern information-communication technologies » № ПП-1730.

By the present Decree has been determined main objectives of further implementation and development of the information-communication technologies of the Republic of Uzbekistan for 2012-2014 years.

In particular:

- ensuring of the establishment of National information system based on phased integration of the information systems of the state authorities, as well as, legal entities and individuals;
- developing of the information systems of automation of the activity of the state authorities, which let to improve the quality and efficiency of the functions performed by them;
- enlarging the list and improvement of the interactive government services provided by the state authorities for business entities and population, ensuring of the wide access for the appropriate information resources, in rural areas as well;

- improving of the regulation systems in the sphere of information-communication technologies by taking into consider of the condition of development of the information resources, technologies and systems, including information security;
- ensuring of the information security of National information systems, the security of its information systems and resources.

In last years image recognition has been faced more and more applications. Recognition of the voice, printed and handwritten text, different pictures considerably simplifies the interaction between human and computer, it brings a premise for applying different systems of artificial intelligence.

Ability to perceive the external world in the form of images allows recognizing infinite amount of images with some certainty on the basis of acquaintance with a finite number of them, and the objective character of the main properties of the images allow simulating the process of their recognition.

Image recognition (objects, signals, situations and processes) – a task of object identification or determination of any kind of its properties based on its image (optical character recognition) or audio record (acoustic recognition) and other characters.

Image – classification grouping in the system of classification, which combines (highlight) a certain group of objects based on some features. Images have the characteristic property that manifests itself in the fact that introduction of a finite number of events from the same set gives you the opportunity to learn an arbitrarily large number of its representatives. The images have the characteristic properties of the objective in the sense that different people trained on different observational material mostly identical and independently classify the same objects. In the classical formulation of the problem of recognition, the universal set is broken into pieces images. Each mapping of an object to perceive bodies of the recognition system, regardless of its position relative to

these bodies, called the image of the object, and a plurality of such images, united by common properties represents by itself images.

Object and subject of research. Object of research is image recognition. Subject of research are methods of image recognition using template matching algorithm.

Research purpose. The purpose of work consists of the research on existing major algorithms of image recognition on the basis of face recognition using template matching technique, conduction of a comparative analysis of these algorithms, development and implementation of a unique parameter for one of these algorithms and development of face recognition software in the programming language C#.

Research objectives:

- review of the sources of literature devoted to the solution of image recognition problems;
- development of a unique parameter for existing image recognition algorithm on the basis of the template matching technique;
- implementation of face recognition computer software in the field of programming C# using algorithm based on unique proposed parameter;
- testing of the developed computer software in the real-life situations.

Research methods. To achieve the goal were used methods of empirical research, probability theory, algorithms and methods of image recognition based on template matching.

Scientific novelty. Scientific novelty of research work consists of comparative analysis of the major algorithms and methods of image recognition on the basis of face recognition, implementation of the face recognition method using proposed unique parameter to the existing image recognition algorithm.

Practical value. Proposed software on this research work is able to capture a human face via web camera and store it in the database, hereinafter

detect a human face and recognize it by matching it with the images which are stored in the database using several major algorithms.

Publications. On the subject of dissertation following 2 articles have been published:

- “Methods of image recognition using template matching algorithm” in the Republician scientific-technical conference of young scientists, researchers, master students and students called “Information technologies and telecommunication problems” held in March 14-15, 2013 in TUIT.
- “An impact of (video-based) face recognition approach” in the Republician scientific-technical conference called “Perspectives of an effective development of the information technologies and telecommunication systems” in 13-14 March, 2014 in TUIT.

Structure and content. Mater’s dissertation work consists of introduction, three chapters, conclusion, list of used references and application.

Chapter I. Theoretical basis of image recognition and face recognition

1. Image recognition theory

Image recognition is largely synonymous to machine learning. This branch of artificial intelligence focuses on the recognition of patterns and regularities in data. In many cases, these images are learned from labeled "training" data (supervised learning), but when no labeled data is available other algorithms can be used to discover previously unknown patterns (unsupervised learning).

The terms image recognition, machine learning, data mining and knowledge discovery in databases (KDD) are hard to separate, as they largely overlap in their scope. Machine learning is the common term for supervised learning methods and originates from artificial intelligence, whereas KDD and data mining have a larger focus on unsupervised methods and stronger connection to business use. Image recognition has its origins in engineering, and the term is popular in the context of computer vision: a leading computer vision conference is named Conference on Computer Vision and Pattern Recognition. In image recognition, there may be a higher interest to formalize, explain and visualize the image; whereas machine learning traditionally focuses on maximizing the recognition rates. Yet, all of these domains have evolved substantially from their roots in artificial intelligence, engineering and statistics; and have become increasingly similar by integrating developments and ideas from each other.

In machine learning, image recognition is the assignment of a label to a given input value. In statistics, discriminant analysis was introduced for this same purpose in 1936. An example of image recognition is classification, which attempts to assign each input value to one of a given set of *classes* (for example, determine whether a given email is "spam" or "non-spam"). However, image

recognition is a more general problem that encompasses other types of output as well. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); and parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence.

Image recognition algorithms generally aim to provide a reasonable answer for all possible inputs and to perform "most likely" matching of the inputs, taking into account their statistical variation. This is opposed to *template matching* algorithms, which look for exact matches in the input with pre-existing templates. A common example of a template-matching algorithm is regular expression matching, which looks for templates of a given sort in textual data and is included in the search capabilities of many text editors and word processors. In contrast to pattern recognition, pattern matching is generally not considered a type of machine learning, although template-matching algorithms (especially with fairly general, carefully tailored templates) can succeed in providing similar-quality output to the sort provided by image-recognition algorithms.

Image recognition is studied in many fields, including psychology, psychiatry, ethology, cognitive science, traffic flow and computer science.

Image recognition is generally categorized according to the type of learning procedure used to generate the output value. *Supervised learning* assumes that a set of *training data* (the *training set*) has been provided, consisting of a set of instances that have been properly labeled by hand with the correct output. A learning procedure then generates a *model* that attempts to meet two sometimes conflicting objectives: Perform as well as possible on the training data, and generalize as well as possible to new data (usually, this means being as simple as possible, for some technical definition of "simple", in accordance with Occam's Razor, discussed below). Unsupervised learning, on the other

hand, assumes training data that has not been hand-labeled, and attempts to find inherent templates in the data that can then be used to determine the correct output value for new data instances. A combination of the two that has recently been explored is semi-supervised learning, which uses a combination of labeled and unlabeled data (typically a small set of labeled data combined with a large amount of unlabeled data). In cases of unsupervised learning, there may be no training data at all to speak of; in other words, the data to be labeled *is* the training data.

Sometimes different terms are used to describe the corresponding supervised and unsupervised learning procedures for the same type of output. For example, the unsupervised equivalent of classification is normally known as *clustering*, based on the common perception of the task as involving no training data to speak of, and of grouping the input data into *clusters* based on some inherent similarity measure (e.g. the distance between instances, considered as vectors in a multi-dimensional vector space), rather than assigning each input instance into one of a set of pre-defined classes. Note also that in some fields, the terminology is different: For example, in community ecology, the term "classification" is used to refer to what is commonly known as "clustering".

The piece of input data for which an output value is generated is formally termed an *instance*. The instance is formally described by a vector of *features*, which together constitute a description of all known characteristics of the instance. (These feature vectors can be seen as defining points in an appropriate multidimensional space, and methods for manipulating vectors in vector spaces can be correspondingly applied to them, such as computing the dot product or the angle between two vectors.) Typically, features are either categorical (also known as nominal, i.e., consisting of one of a set of unordered items, such as a gender of "male" or "female", or a blood type of "A", "B", "AB" or "O"), ordinal (consisting of one of a set of ordered items, e.g.,

"large", "medium" or "small"), integer-valued(e.g., a count of the number of occurrences of a particular word in an email) or real-valued (e.g., a measurement of blood pressure). Often, categorical and ordinal data are grouped together; likewise for integer-valued and real-valued data. Furthermore, many algorithms work only in terms of categorical data and require that real-valued or integer-valued data be *discretized* into groups (e.g., less than 5, between 5 and 10, or greater than 10).

Many common image recognition algorithms are probabilistic in nature, in that they use statistical inference to find the best label for a given instance. Unlike other algorithms, which simply output a "best" label, often probabilistic algorithms also output a probability of the instance being described by the given label. In addition, many probabilistic algorithms output a list of the N-best labels with associated probabilities, for some value of N, instead of simply a single best label. When the number of possible labels is fairly small (e.g., in the case of classification), N may be set so that the probability of all possible labels is output. Probabilistic algorithms have many advantages over non-probabilistic algorithms:

- They output a confidence value associated with their choice. (Some other algorithms may also output confidence values, but in general, only for probabilistic algorithms are this value mathematically grounded in probability theory. Non-probabilistic confidence values can in general not be given any specific meaning, and only used to compare against other confidence values output by the same algorithm.)
- Correspondingly, they can abstain when the confidence of choosing any particular output is too low.
- Because of the probabilities output, probabilistic pattern-recognition algorithms can be more effectively incorporated into larger machine-learning tasks, in a way that partially or completely avoids the problem of error propagation[1].

Image recognition encompasses major areas as car number plate recognition, bar-code recognition (figure 1), biometric recognition and so on. But mainly it focuses on biometrical recognition.

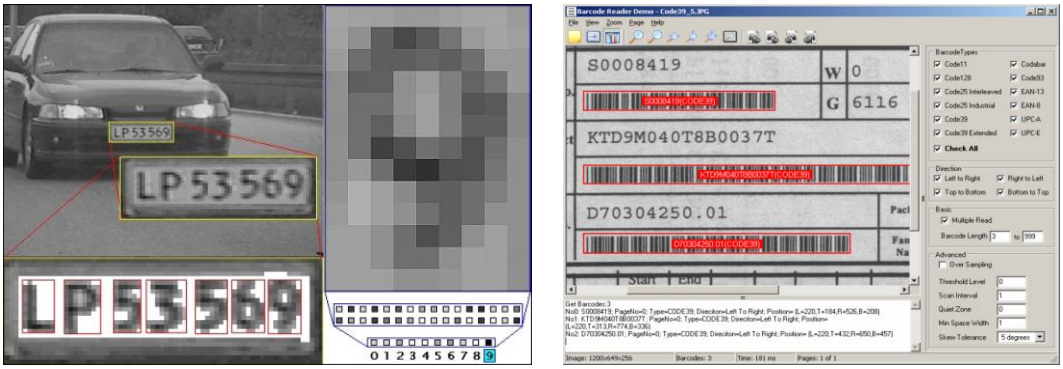


Figure 1. Bar code and car number plate recognition.

2. Biometrics in identity recognition

In general terms, a biometric is observed data of a human that allows the identity of that person to be determined. Examples of biometrics actively being investigated are DNA, shape of the ear, faces, fingerprints, hand geometry, irises, pattern of keystrokes on a key-board, signature and speech (figure 2).

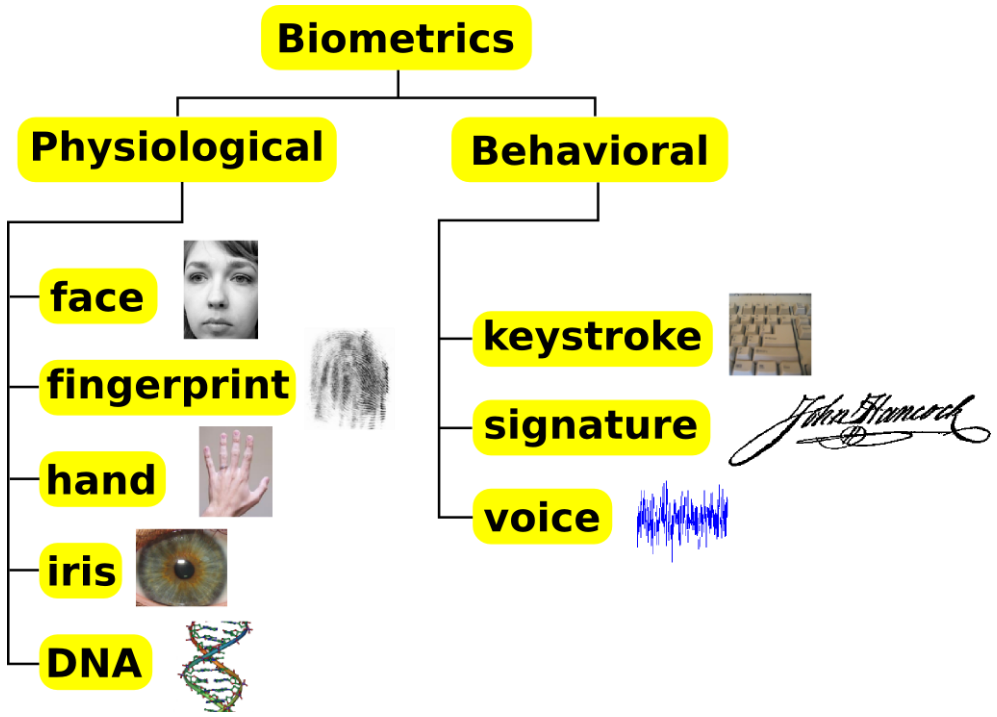


Figure 2. Classification of biometrics which might be used in digital recognition.

Some biometrics can be collected without a person's knowledge (faces, speech, signature, keystrokes, and ears), others (fingerprints and hand geometry) require a person's cooperation, and for some (irises and DNA) this determination has not been made. This distinctive is important. Potential applications for non-instructive biometrics include remote surveillance and monitoring (border crossing and airport security), while potential applications for cooperative biometrics include access control and verifying transactions executed via electronic commerce.

Faces, fingerprints, irises, ears are intrinsically image based and require image processing, pattern recognition, and computer vision techniques to implement. Whereas, hand geometry, keystrokes, signature, and speech fall into the domains of signal processing and image recognition. Recent efforts have looked at combining multiple biometrics (audio-video, faces-fingerprints, etc)[2].

In most cases a biometric system is embedded in the authentication process of an identity management system. Its result is used to decide if the individual that has delivered the biometric data shall be recognized by the identity management system[3].

There are two recognition sub problems, identification and verification:

In identification, the algorithm is presented with a biometric image of an unknown person. The algorithm reports its best estimate of the identity of the unknown person from the database of known individuals. In a more general response, the algorithm will report a list of the most similar individuals in the database. The majority of the identification applications are in law enforcement, forensic, and intelligence. The application includes identifying faces from mugshots, surveillance images, newspapers, photographs, and images of deceased people.

In verification (also referred to as authentication), the algorithm is presented with a biometric image and a claimed identity of the person. The

algorithm either accepts or rejects the claim. Or, the algorithm can return a confidence measure of the validity of the claim. Verification applications include control access to apartment of secure buildings, verifying identities during point of sale transactions, and continuous verification of identity at computer terminals or in secure facilities[2].

So, the biometric system may be used in two modes:

1. *Verification mode*: An individual makes a(n identity) claim. The biometric system compares the captured biometric data sample with the biometric reference template that corresponds to the claim(ed identity). The outcome is the acceptance or the refusal of the (identity) claim.

2. *Identification mode*: The biometric system compares the captured biometric data sample with all available biometric reference templates. All comparisons with a sufficient similarity to a stored reference template are selected and designate a candidate identity. The outcome is a list of identities that may belong to the individual. This list may contain zero, one or more entries. Individuals may be identified in this mode with or without their consent.

All biometric recognition systems have some common main functional components in a typical processing chain. These components are (figure 3):

1. a storage entity with the biometric data samples (reference templates) of the enrolled individuals that is linked to or integrated in a database with the identity information of the corresponding individuals

2. a sensor device and some pre-processing to capture the biometric data sample from an individual as input data

3. a comparison process that evaluates the similarity between the reference templates and the captured data sample and that results in a similarity score and

4. a decision function that decides if a data sample matches to a certain reference template.

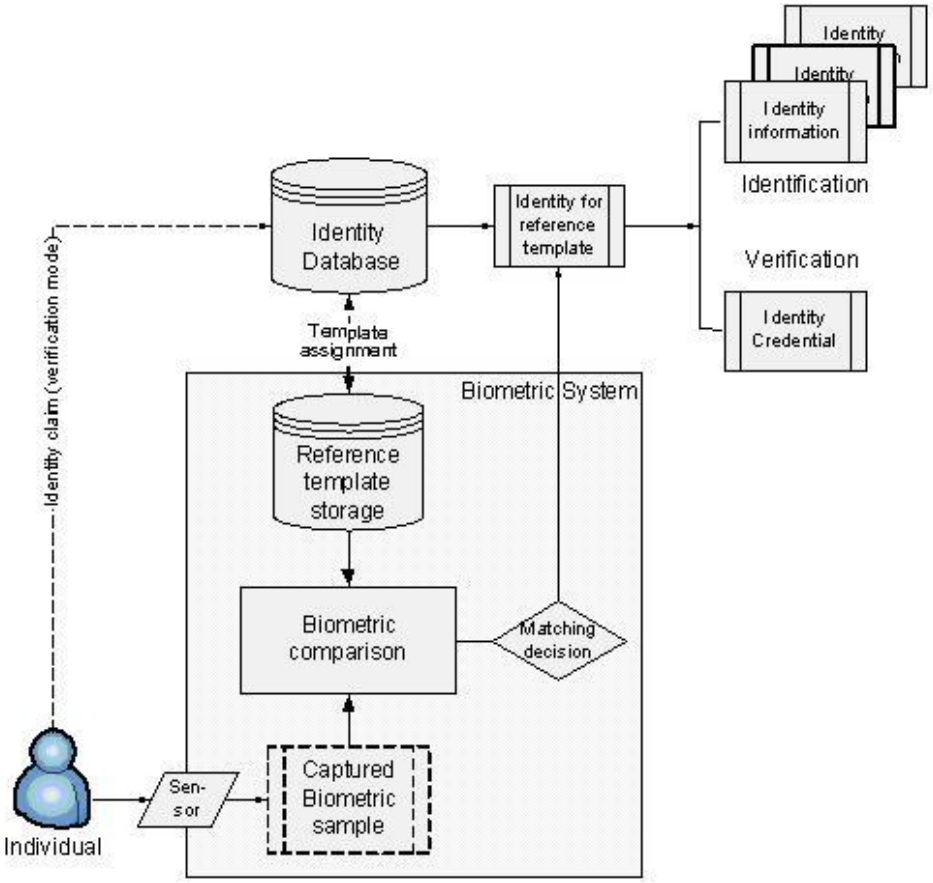


Figure 3. The main processing components of a biometric recognition system.

The main functional components are hereunder described:

Delivery – Protocol that an individual follows (knowingly or unknowingly) to provide a signal of a biological and/or behavioral characteristic to the biometric application system.

Capture – Acquisition of a biometric sample data from the original biometric characteristics of the individual with appropriate sensors (capture devices).

Extract – Conversion of the captured biometric sample data to an intermediate form that contains the concentrated distinguishing biometric property information of the biometric characteristics.

Create Template – Conversion of the intermediate data into an individual's template that can be stored (reference template) or that can be used as input (query template from sample data) for a comparison process that uses previously stored reference templates.

Compare – Comparison and matching of the query template with the information in a stored reference template.

Recognize – Mapping of the recognized query templates on the identity data of the individuals that are stored in the system (identification mode) or acceptance of an identity claim of a specific individual and delivery of a corresponding identity credential (verification mode).

All biometric systems run in two separate processing phases. For each individual that shall be recognized by a biometric system first an initialization, called enrolment, takes place. In this processing phase the individual subject provides samples of a biometric characteristic to establish a new so called reference template. After the enrolment, the subject is known by the biometric system. In the subsequent query phase, the subject provides a new sample called query template that is processed and compared with the saved reference templates of all enrolled subjects (identification) or with the saved template of a specified subject (verification). The output of the system may be a simple yes/no, or an identity credential with identity information about the subject for a system that operates in the verification mode, or a list of identity data that correspond to the best matches (comparison scores) for a system running in an identification mode.

A schematic model of all processing steps in a biometric system with inputs and outputs is shown in figure 4. The red track represents the enrolment mode and the green track the query phase processing. The two different recognition modes (verification of a biometric sample with claimed identity; identification through a mapping of a biometric sample to potential candidate reference templates) are distinguished by the additional ‘identity claim’ input in the recognition step.

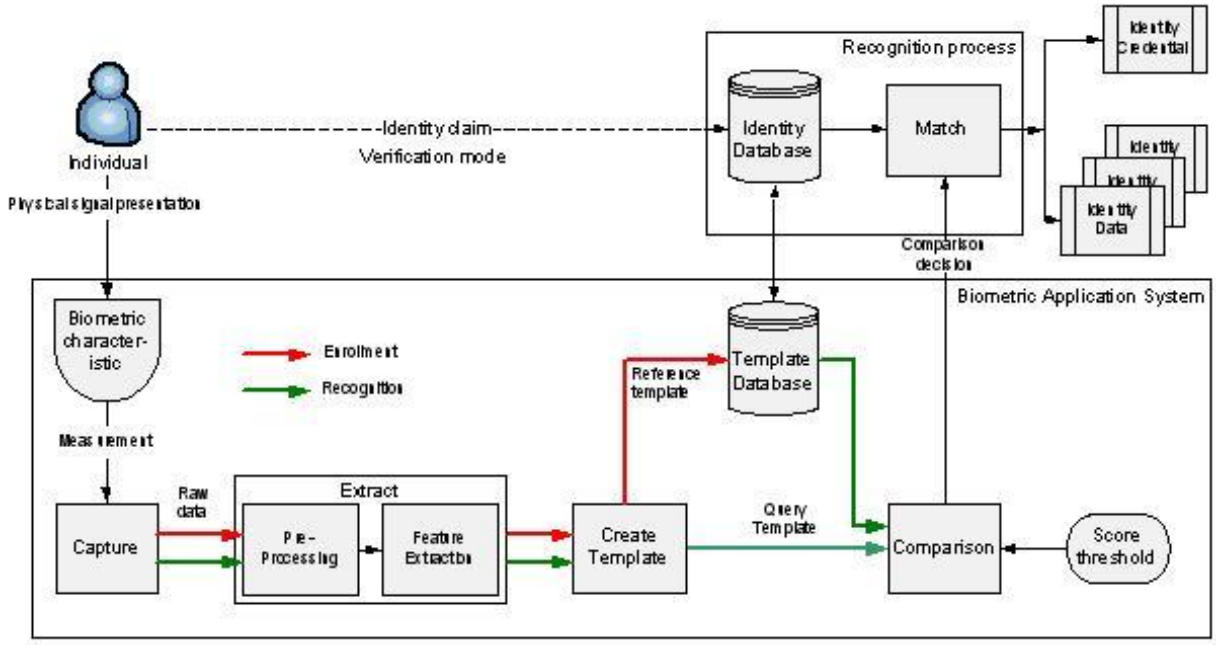


Figure 4. Schematic representation of the processing steps of a biometric system. The processing for the two phases (enrolment, comparison) follows two different flow paths.

The reference diagram presents a simplified logical model of a general biometric application system (fig.4). The biometric data inside the functional chain may be formatted and tagged in a standardized way generically termed as a Biometric Identification Record (BIR) (BioAPI). A real implementation could be complicated by additional factors such as the following:

1. The requirement for confidentiality and integrity of the biometric and user identity data and of the transmission paths between components and involved systems. These paths may be protected by cryptographic mechanisms or other means, e.g. physical access control. Unique session keys may also be used to counter replay attacks.

2. The system may be distributed over multiple locations, such as in a client server architecture.

3. The system may be under the control of different instances such as the user, the operator, a trusted third party or a governmental organization. Such instances may control different parts of the processing chain in various combinations (see discussion below about the different control types and models, such as central control, divided control with trust and multilateral control)[3].

The main study area of the biometric recognition is face recognition technology. In order to recognize human face, first we should detect a face. So that's why first, we will refer to the concept of Face detection.

3. Face detection and recognition

Face detection. Human face detection and recognition techniques play an important role in applications like face recognition, video surveillance, human computer interface and face image databases. Using color information in images is one of the various possible techniques used for face detection. The novel technique used in this project was the combination of various techniques such as skin color detection, template matching, gradient face detection to achieve high accuracy of face detection in frontal faces. The objective in this work was to determine the best rotation angle to achieve optimal detection. Also eye and mouse template matching have been put to test for feature detection. Traditionally, computer vision systems have been used in specific tasks, such as performing tedious and repetitive visual tasks of assembly line inspection. Current development in this area has moved toward more generalized vision applications such as face recognition, video coding techniques, biometrics, surveillance, man-machine interaction, animation and database indexing and many other applications that have face detection as the primary building block of their systems.

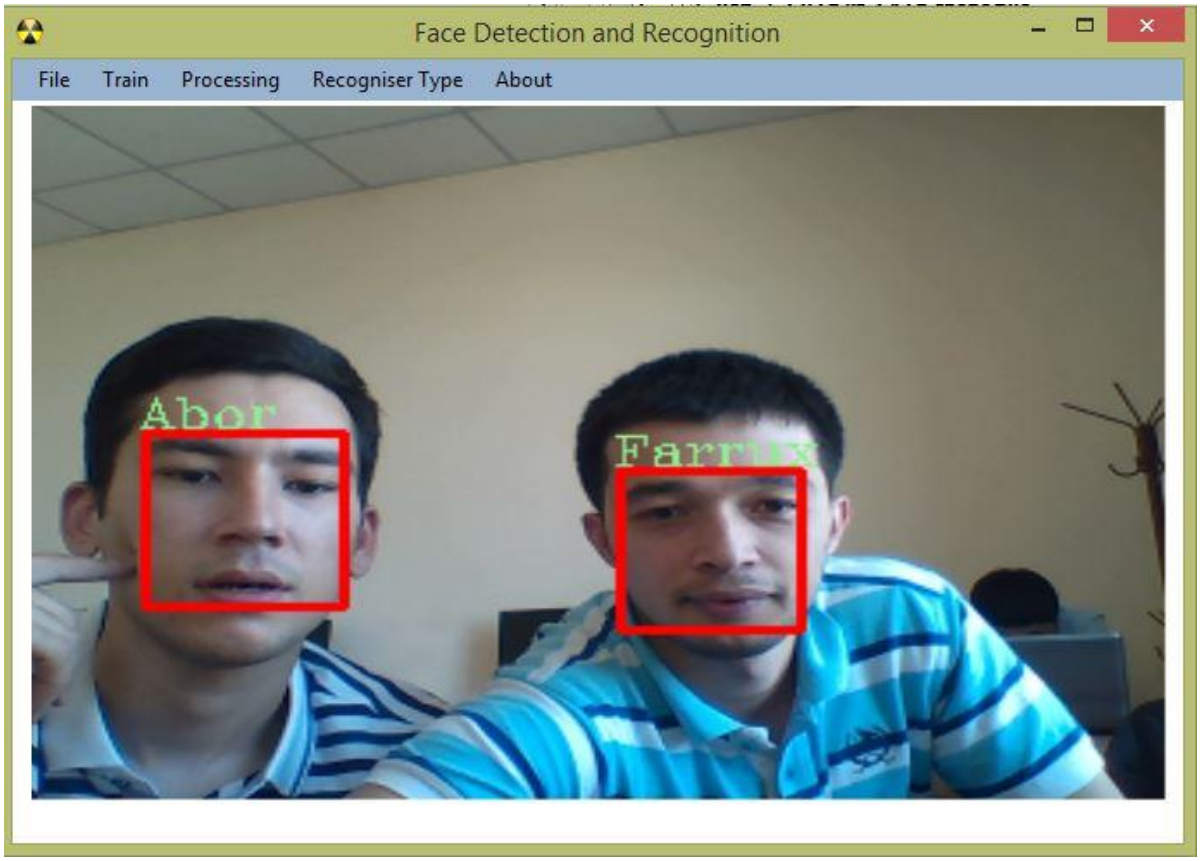


Figure 5. Sample of face detection (and recognition).

Many of the current face recognition systems assume the availability of frontal faces. In reality this assumption may not hold due to the nature of face appearance and environment conditions. The exclusion of background in these images is necessary for reliable face classification. However in realistic application scenarios a face could occur in a complex background and in many different positions. Recognition systems that are built on the standard face images are likely to mistake areas of the background as faces. In order to rectify the problem, a visual processor is needed to localize and extract the face region from the background. Face detection is one of the visual tasks that humans can do effortlessly. However, in computer vision terms, this task is not easy. A general statement of the problem can be defined as follows: Given an image or a video sequence, detect and localize an unknown number(if any) of faces. The solution to this problem involves segmentation, extraction and verification of faces and possibly facial features from an uncontrolled background. An ideal

face detector should achieve this aim despite illumination, rotation, different facial expressions, orientations and camera distance from the object.

In the last two decades huge progress has been made to increase the accuracy of the face detectors while many different methods have been introduced in this area.

Research on face detection started in the beginning of the 1970s, where simple heuristic and anthropometric techniques were used. The techniques that were used were too rigid and worked only on the plain background and any challenge would confuse the system to perform properly. Despite these problems the growth of research interest remained stagnant until the 1990s, when practical face recognition and video coding systems started to become a reality.

Over the past two decades there has been a great deal of research interest spanning several important aspects of face detection. More robust segmentation schemes have been presented, particularly those using motion, color, and generalized information. The use of statistics and neural networks has also enabled faces to be detected from cluttered scenes at different distances from the camera. Additionally, there are numerous advances in the design of feature extractors such as the deformable templates and the active contours which can locate and track facial features accurately. Because face detection techniques require a priori information of the face, they can be effectively organized into two broad categories distinguished by their different approach to utilizing face knowledge. The techniques in the first category make explicit use of face knowledge and follow the classical detection methodology in which low level features are derived prior to knowledge-based analysis.

The apparent properties of the face such as skin color and face geometry are exploited at different system levels. Typically, in these techniques face detection tasks are accomplished by manipulating distance, angles, and area measurements of the visual features derived from the scene. Since features are

the main ingredients, these techniques are termed the feature-based approach. These approaches have embodied the majority of interest in face detection research starting as early as the 1970s and therefore account for most of the literature reviewed herein. Taking advantage of the current advances in pattern recognition theory, the techniques in the second group address face detection as a general recognition problem. Image-based representations of faces, for example in 2D intensity arrays, are directly classified into a face group using training algorithms without feature derivation and analysis. Unlike the feature-based approach, these relatively new techniques incorporate face knowledge implicitly into the system through mapping and training schemes[4]. Different methods that were just mentioned in this section are shown in figure 6.

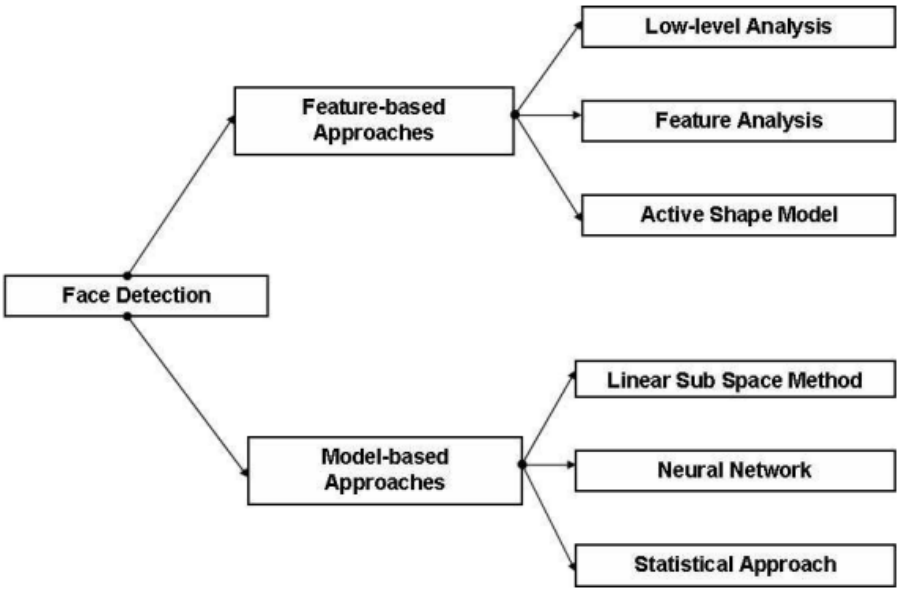


Figure 6. Different approaches to face detection.

Face recognition. As one of the most successful applications of image analysis and understanding, face recognition has recently received significant attention, especially during the past few years. A general statement of the problem can be formulated as follows: given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces. Available collateral information such as race, age, gender, facial expression and speech may be used in narrowing the search (enhancing recognition). The solution to the problem involves segmentation of faces (face detection) from cluttered scenes, feature extraction from the face region, recognition or verification. In identification problems, the input to the system is an unknown face, and the system reports back the determined identity from a database of known individuals, whereas in verification problems, the system needs to confirm or reject the claimed or reject the claimed identity of the input face[5].

Facial recognition records the spatial geometry of distinguishing features of the face. Different vendors use different methods of facial recognition, however, all focus on measures of key features of the face. Because a person's face can be captured by a camera from some distance away, facial recognition has a clandestine or covert capability (i.e. the subject does not necessarily know he has been observed). For this reason, facial recognition has been used in projects to identify card counters or other undesirables in casinos, shoplifters in stores, criminals and terrorists in urban areas.

Although the concept of recognizing someone from facial features is intuitive, facial recognition, as a biometric, makes human recognition a more automated, computerized process. What sets apart facial recognition from other biometrics is that it can be used for surveillance purposes. For example, public safety authorities want to locate certain individuals such as wanted criminals, suspected terrorists, and missing children. Facial recognition may have the potential to help the authorities with this mission.

Facial recognition offers several advantages. The system captures faces of people in public areas, which minimizes legal concerns for reasons explained below. Moreover, since faces can be captured from some distance away, facial recognition can be done without any physical contact. This feature also gives facial recognition a clandestine or covert capability.

For any biometric system to operate, it must have records in its database against which it can search for matches. Facial recognition is able to leverage existing databases in many cases. For example, there are high quality mugshots of criminals readily available to law enforcement. Similarly, facial recognition is often able to leverage existing surveillance systems such as surveillance cameras or closed circuit television (CCTV).

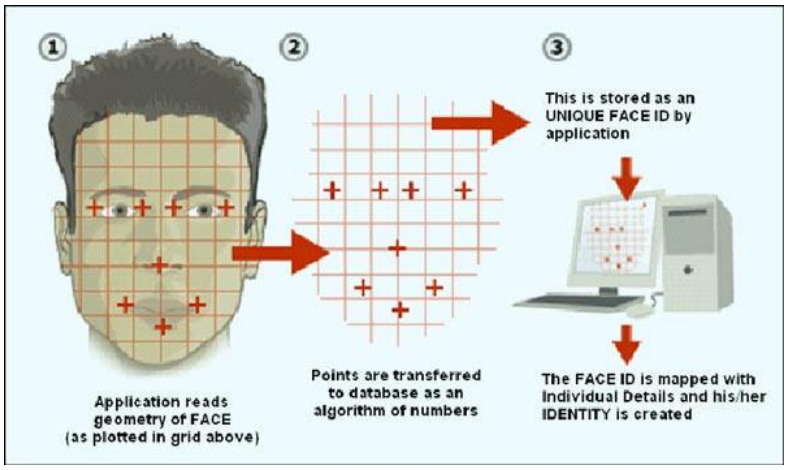


Figure 7. Face recognition sequence.

As a biometric, facial recognition is a form of computer vision that uses faces to attempt to identify a person or verify a person's claimed identity. Regardless of specific method used, facial recognition is accomplished in a five step process.

- First, an image of the face is acquired. This acquisition can be accomplished by digitally scanning an existing photograph or by using an electro-optical camera to acquire a live picture of a subject. As video is a rapid

sequence of individual still images, it can also be used as a source of facial images.

- Second, software is employed to detect the location of any faces in the acquired image. This task is difficult, and often generalized patterns of what a face “looks like” (two eyes and a mouth set in an oval shape) are employed to pick out the faces.
- Once the facial detection software has targeted a face, it can be analyzed. As noted in slide three, facial recognition analyzes the spatial geometry of distinguishing features of the face. Different vendors use different methods to extract the identifying features of a face. Thus, specific details on the methods are proprietary. The most popular method is called Principle Components Analysis (PCA), which is commonly referred to as the eigenface method. PCA has also been combined with neural networks and local feature analysis in efforts to enhance its performance. Template generation is the result of the feature extraction process. A template is a reduced set of data that represents the unique features of an enrollee’s face. It is important to note that because the systems use spatial geometry of distinguishing facial features, they do not use hairstyle, facial hair, or other similar factors.
- The fourth step is to compare the template generated in step three with those in a database of known faces. In an identification application, this process yields scores that indicate how closely the generated template matches each of those in the database. In a verification application, the generated template is only compared with one template in the database – that of the claimed identity.
- The final step is determining whether any scores produced in step four are high enough to declare a match. The rules governing the declaration of a match are often configurable by the end user, so that he or she can determine how the facial recognition system should behave based on security and operational considerations.

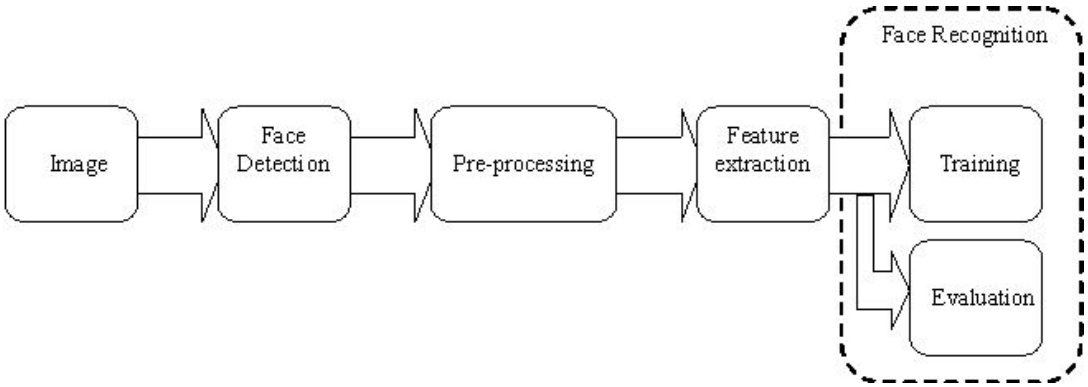


Figure 8. Face recognition steps.

People are generally very good at recognizing faces that they know. However, people experience difficulties when they perform facial recognition in surveillance or watch post scenario. Several factors account for these difficulties: most notably, humans have a hard time recognizing unfamiliar faces. Combined with relatively short attention spans, it is difficult for humans to pick out unfamiliar faces. Considerable evidence supports this claim. For example, in a British study, trained supermarket cashiers were tested on their ability to screen shoppers using credit cards that included a photograph of the card owner. Each shopper was issued four cards: one with a recent picture of the shopper, one that included minor modifications to the shopper's hairstyle, facial hair or accessories (e.g., glasses, hat), another card with a photograph of a person similar in appearance to the shopper, and the last card with a photograph of a person who was only of the same sex and race as the shopper. When the various cards were presented to the checkout clerks, more than half of the fraudulent cards were accepted. The breakdown was as follows: 34 percent of the cards that did not look like the shopper were accepted, 14 percent of the cards where the appearance had been altered were accepted, and 7 percent of the unchanged cards were rejected by the clerks. In addition to unfamiliar face recognition problems, the ability of human beings to detect critical signals drops rapidly from the start of a task, and stabilizes at a significantly lower level

within 25 to 35 minutes. Thus the ability of people to focus their attention drops significantly after only half an hour.

Machines also experience difficulties when they perform facial recognition in surveillance or watch post scenario. Dr. James L. Wayman, a leading biometrics expert, has explained that performing facial recognition processes with relatively high fidelity and at long distances remains technically challenging for automated systems. At the most basic level, detecting whether a face is present in a given electronic photograph is a difficult technical problem. Dr. Wayman has noted that subjects should ideally be photographed under tightly controlled conditions. For example, each subject should look directly into the camera and fill the area of the photo for an automated system to reliably identify the individual or even detect his face in the photograph. Thus, while the technology for facial recognition systems shows promise, it is not yet considered fully mature[6].

There are at least two reasons for this trend; the first is the wide range of commercial and law enforcement applications and the second is the availability of feasible technologies after 30 years of research.

The strong need for user-friendly systems that can secure our assets and protect our privacy without losing our identity in a sea of numbers is obvious. At present, one needs a PIN to get cash from an ATM, a password for a computer, a dozen others to access the internet, and so on, Although extremely reliable methods of biometric personal identification exist, e.g., fingerprint analysis and retinal or iris scans, these methods rely on the cooperation of the participants, whereas a personal identification system based on analysis of frontal or profile images of the face is often effective without the participant’s cooperation or knowledge.

Areas	Specific Applications
Biometrics	Drivers' Licenses, Entitlement Programs
	Immigration, National ID, Passports, Voter Registration
	Welfare Fraud
Information Security	Desktop Logon (Windows NT, Windows 95)
	Application Security, Database Security, File Encryption
	Intranet Security, Internet Access, Medical Records
	Secure Trading Terminals
Law Enforcement and Surveillance	Advanced Video Surveillance, CCTV Control
	Portal Control, Post-Event Analysis
	Shoplifting and Suspect Tracking and Investigation
Smart Cards	Stored Value Security, User Authentication
Access Control	Facility Access, Vehicular Access

Figure 9. Typical applications of face recognition.

Automatic face recognition consists of subtasks in a sequential manner: face detection, face segmentation/normalization, and face recognition/verication. Many methods of face recognition have been proposed. Basically they can be divided into holistic template matching based systems, geometrical local-feature-based schemes, and hybrid schemes. Even though schemes of all these types have been successfully applied to the task of face recognition, they do have certain advantages and disadvantages. Thus an appropriate scheme should be chosen based on the specific requirements of a given task.

Despite the success of face recognition systems, there are many recognition issues remain. Among those issues the following two are prominent for most systems: 1) the illumination problem, 2) the pose problem. The illumination problem is illustrated in Fig. 10 where the same face appears differently due to the change in lighting. More specifically, the changes induced by illumination could be larger than the differences between individuals, causing systems based on comparing images to misclassify the identity of the input image.

Figure 10. The Illumination problem.

The pose problem is illustrated in Fig. 11 where the same face app ears differently due to changes in viewing condition. Moreover, when illumination variation also appears in the face images, the task of face recognition becomes even more difficult (Fig. 3).

Figure 11. The pose (and illumination) problem.

Difficulties due to illumination and pose variations have been documented in many evaluations of face recognition systems. An even more difficult case is the combined problem of pose and illumination variations. Unfortunately, this happens when face images are acquired in uncontrolled environments, for instance, in surveillance video clips. In the following, we examine the two problems in turn and review some existing approaches to these problems. More importantly, we point out the pros and cons of these methods so an appropriate approach can b e applied to the specific task[5].

Chapter I summary

Chapter one is devoted to the theoretical view of the image recognition study area. There you can find basic explanation of the concepts of image recognition and its impact in human life, its classification and fields of application. Furthermore, there were given information about biometrics in identity recognition and its classification as well. Further, main attention paid for the face recognition technology which is probably the most important aspect of image recognition. Theoretical basis of face recognition has been thoroughly overviewed and explained, as well as recognition process, its sequence of steps as face detection prior to face recognition have been showed. Also, first chapter encompasses important issues as problems of face recognition, recognition tasks and so on.

Chapter II. Research on the template matching technique and major algorithms of face recognition

1. Template matching technique

Template matching is one of the areas of profound interests in recent times. It has turned out to be a revolution in the field of computer vision. Template matching provides a new dimension into the image-processing capabilities, although there have been many attempts to resolve different issues in this field there have always been newer concepts emerging in this ever challenging field. Template matching is a technique used in classifying an object by comparing portions of images with another image. One of the important techniques in Digital image processing is template matching. Template matching is widely used for processing images and pictures. Some of its wide-spread applications object to location, edge detection of images, to plot a route for mobile robot and in image registration techniques. In general, a technique includes its unique algorithm or method, which compares the template image with input image and finds similarity between them.

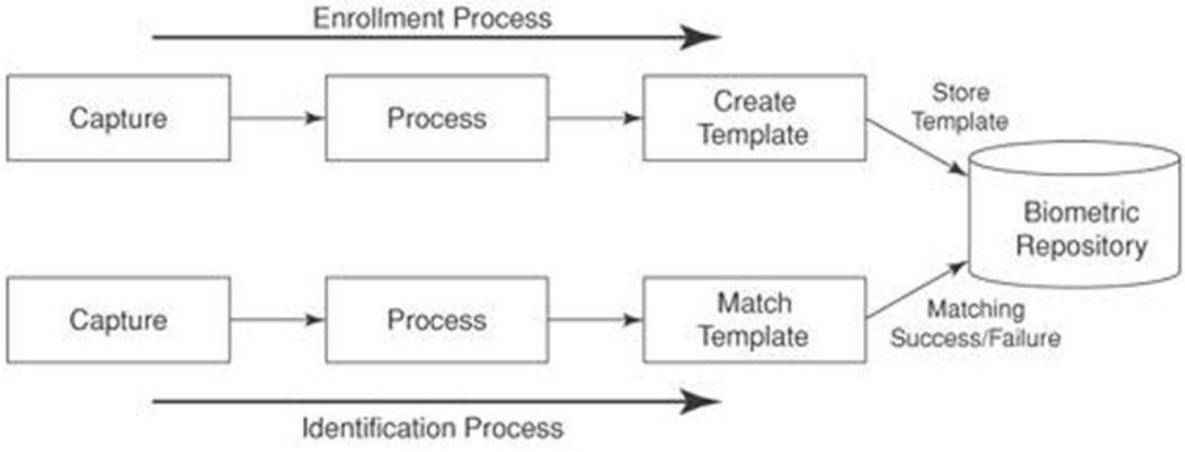


Figure 12. Template matching.

The choice of matching depends on the nature of the image and the problem to be solved. General classifications of template or image matching approaches are: Template approaches and Feature-based approaches[7].

Feature-based approach

If the template image has strong features, a feature-based approach may be considered; the approach may prove further useful if the match in the search image might be transformed in some fashion. Since this approach does not consider the entirety of the template image, it can be more computationally efficient when working with source images of larger resolution, as the alternative approach, template-based, may require searching potentially large amounts of points in order to determine the best matching location.

Template-based approach

For templates without strong features, or for when the bulk of the template image constitutes the matching image, a template-based approach may be effective. As aforementioned, since template-based template matching may potentially require sampling of a large number of points, it is possible to reduce the number of sampling points by reducing the resolution of the search and template images by the same factor and performing the operation on the resultant downsized images (multiresolution, or pyramid, image processing), providing a search window of data points within the search image so that the template does not have to search every viable data point, or a combination of both.

Template-based matching and convolution

A basic method of template matching uses a convolution mask (template), tailored to a specific feature of the search image, which we want to detect. This technique can be easily performed on grey images or edge images. The convolution output will be highest at places where the image structure matches the mask structure, where large image values get multiplied by large mask values.

This method is normally implemented by first picking out a part of the search image to use as a template: We will call the search image **S(x, y)**, where **(x, y)** represent the coordinates of each pixel in the search image. We will call the template **T(x_t, y_t)**, where **(x_t, y_t)** represent the coordinates of each pixel in the template. We then simply move the center (or the origin) of the template **T(x_t, y_t)** over each **(x, y)** point in the search image and calculate the sum of products between the coefficients in **S(x, y)** and **T(x_t, y_t)** over the whole area spanned by the template. As all possible positions of the template with respect to the search image are considered, the position with the highest score is the best position. For example, one way to handle translation problems on images, using template matching is to compare the intensities of the pixels, using the **SAD** (Sum of absolute differences) measure.

A pixel in the search image with coordinates **(x_s, y_s)** has intensity **$I_s(x_s, y_s)$** and a pixel in the template with coordinates **(x_t, y_t)** has intensity **$I_t(x_t, y_t)$**. Thus the absolute difference in the pixel intensities is defined as **$\mathrm{Diff}(x_s, y_s, x_t, y_t) = | I_s(x_s, y_s) - I_t(x_t, y_t) |$**.

$$SAD(x,y) = \sum_{i=0}^{T_{\text{rows}}} \sum_{j=0}^{T_{\text{cols}}} \text{Diff}(x+i, y+j, i, j) \tag{1.1}$$

The mathematical representation of the idea about looping through the pixels in the search image as we translate the origin of the template at every pixel and take the SAD measure is the following:

$$\sum_{x=0}^{S_{\text{rows}}} \sum_{y=0}^{S_{\text{cols}}} SAD(x,y) \tag{1.2}$$

S_{rows} and **S_{cols}** denote the rows and the columns of the search image and **T_{rows}** and **T_{cols}** denote the rows and the columns of the template image, respectively. In this method the lowest SAD score gives the estimate for the best

position of template within the search image. The method is simple to implement and understand, but it is one of the slowest methods.

Implementation: In this simple implementation, it is assumed that the above described method is applied on grey images: This is why **Grey** is used as pixel intensity. The final position in this implementation gives the top left location for where the template image best matches the search image.

```
minSAD = VALUE_MAX;

// loop through the search image
for ( int x = 0; x <= S_rows - T_rows; x++ ) {
   for ( int y = 0; y <= S_cols - T_cols; y++ ) {
      SAD = 0.0;

          // loop through the template image

          for ( int j = 0; j < T_cols; j++ )
         for ( int i = 0; i < T_rows; i++ ) {

            pixel p_SearchIMG = S[x+i][y+j];
            pixel p_TemplateIMG = T[i][j];

            SAD += abs( p_SearchIMG.Grey - p_TemplateIMG.Grey );
         }

      // save the best found position
      if ( minSAD > SAD ) {
         minSAD = SAD;
         // give me min SAD
         position.bestRow = x;
         position.bestCol = y;
         position.bestSAD = SAD;
      }
   }
}
```

One way to perform template matching on color images is to decompose the pixels into their color components and measure the quality of match between the color template and search image using the sum of the SAD computed for each color separately[8].

Convolution

In mathematics and, in particular, functional analysis, convolution is a mathematical operation on two functions *f* and *g*, producing a third function that is typically viewed as a modified version of one of the original functions, giving the area overlap between the two functions as a function of the amount that one of the original functions is translated. Convolution is similar to cross-correlation. It has applications that include probability, statistics, computer vision, image and signal processing, electrical engineering, and differential equations[9].

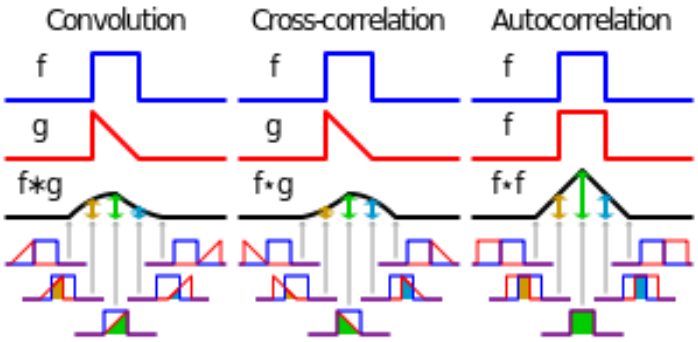


Figure 13: Visual comparison of convolution, cross-correlation and autocorrelation.

Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*[10].

Sum of absolute differences

In digital image processing, the sum of absolute differences (SAD) is an algorithm for measuring the similarity between image blocks. It works by taking the absolute difference between each pixel in the original block and the

corresponding pixel in the block being used for comparison. These differences are summed to create a simple metric of block similarity, the L^1 norm of the difference image or Manhattan distance between two image blocks.

The sum of absolute differences may be used for a variety of purposes, such as object recognition, the generation of disparity maps for stereo images, and motion estimation for video compression.

Example: This example uses the sum of absolute differences to identify which part of a search image is most similar to a template image. In this example, the template image is 3 by 3 pixels in size, while the search image is 3 by 5 pixels in size. Each pixel is represented by a single integer from 0 to 9.

Template	Search image
2 5 5	2 7 5 8 6
4 0 7	1 7 4 2 7
7 5 9	8 4 6 8 5

There are exactly three unique locations within the search image where the template may fit: the left side of the image, the center of the image, and the right side of the image. To calculate the SAD values, the absolute value of the difference between each corresponding pair of pixels is used: the difference between 2 and 2 is 0, 4 and 1 is 3, 7 and 8 is 1, and so forth.

Calculating the values of the absolute differences for each pixel, for the three possible template locations, gives the following:

Left	Center	Right
0 2 0	5 0 3	3 3 1
3 7 3	3 4 5	0 2 0
1 1 3	3 1 1	1 3 4

For each of these three image patches, the 9 absolute differences are added together, giving SAD values of 20, 25, and 17, respectively. From these

SAD values, it could be asserted that the right side of the search image is the most similar to the template image, because it has the lowest sum of absolute differences as compared to the other two locations.

The sum of absolute differences provides a simple way to automate the searching for objects inside an image, but may be unreliable due to the effects of contextual factors such as changes in lighting, color, viewing direction, size, or shape. The SAD may be used in conjunction with other object recognition methods, such as edge detection, to improve the reliability of results[11].

Template Matching in mathematical expression

As a measure, how well an arbitrary pattern of greyvalues, a template g(x,y), matches a given image f(x,y), one uses a (Metric) distance function , e.g.:

$$\int_G (f-g)^2 \quad \text{or} \quad \int_G |f-g| \quad \text{or} \quad \max_G |f-g| \,. \tag{1.3}$$

The minima of these measures are the best match. In the case of the Euclidean distance

$$\int_G (f-g)^2 = \int_G f^2 + \int_G g^2 - 2\int_G f \cdot g \,, \tag{1.4}$$

$\int_G f \cdot g$

the maximum of is the best match, the other terms being constant. This "cross-correlation" yields a result only if the integral is computed over the whole area G. In the discrete case, this takes the form

$$R(i,j) = \sum_m \sum_n f(i+m, j+n) \cdot g(m,n), \tag{1.5}$$

if the variation in the energy of the image f can be ignored. Otherwise the normalized cross-correlation has to be used:

$$R(i,j) = \sum_m \sum_n f(i+m, j+n) \cdot g(m,n) / \sqrt{\sum_m \sum_n f(i+m, j+n)^2}, \qquad (1.6)$$

$$\sum_m \sum_n |f(i,j) - g(i-m, j-n)|$$

$$g \in G$$

It takes the same amount of computing time for any , whereas the computation of the other two measures can be halted as soon as the misregistration

(1.7)

exceeds a given threshold[12].

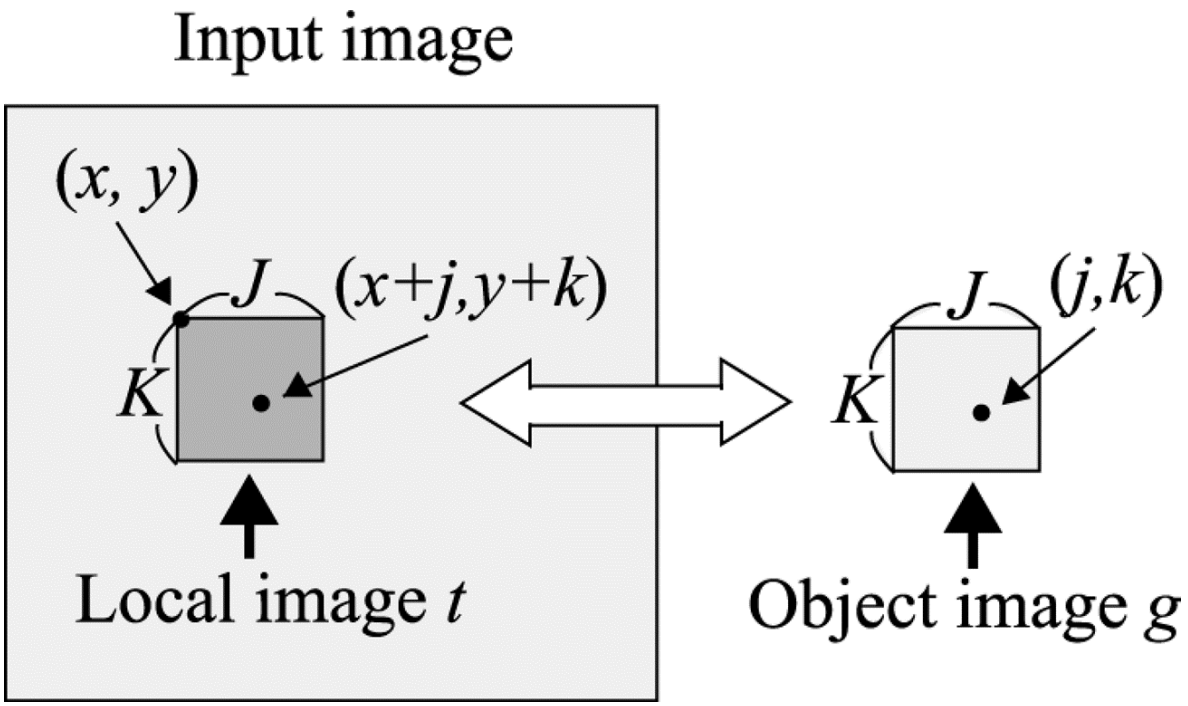


Figure 14. Template matching.

A metric or *distance function* is a function $d(p,q)$ of two points p and q which satisfies[13]:

$$d(p,q) \geq 0 \quad \text{and} \quad d(p,q) = 0 \quad \text{iff} \quad p = q$$
$$d(p,q) = d(q,p)$$
$$d(p,q) \leq d(p,r) + d(r,q)\ .$$

(1.8)

The Euclidean distance or Euclidean metric is the "ordinary" distance between two points that one would measure with a ruler, and is given by the Pythagorean formula. By using this formula as distance, Euclidean space becomes a metric space.

Definition: The Euclidean distance between points **p** and **q** is the length of the line segment connecting them ($\overline{\mathbf{pq}}$).

If $\mathbf{p} = (p_1, p_2, ..., p_n)$ and $\mathbf{q} = (q_1, q_2, ..., q_n)$ are two points in Euclidean n-space, then the distance from **p** to **q**, or from **q** to **p** is given by:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

The position of a point in a Euclidean n-space is a Euclidean vector. So, **p** and **q** are Euclidean vectors, starting from the origin of the space, and their tips indicate two points[14].

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[m+n].$$

In signal processing, *cross-correlation* is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. It is commonly used for searching a long signal for a shorter, known feature.

For continuous functions f and g, the cross-correlation is defined as:

(1.9)

where f* denotes the complex conjugate of f and τ is the time lag.

Similarly, for discrete functions, the cross-correlation is defined as[15]**:**

(2.0)

A *linear subspace* (or vector subspace) is a vector space that is a subset of some other (higher-dimension) vector space. A linear subspace is usually called simply a *subspace* when the context serves to distinguish it from other kinds of subspaces[15].

Image Recognition using Template Matching

The amount of visual information present in the real world is clearly immense. However, recognizing images (objects) from visual information is challenging. The world is also constantly changing, causing most images to be at different state each time they are encountered. This may be due to change in the viewpoint of the observer or simply variations in lighting conditions. Because of this recognizing an image (in a new state) from previous studies of that image (in other states) becomes difficult. Furthermore, if we have studied a number of different images, recognizing a new image becomes a matter of comparing with each of the previous studied images. So the time required to search our studied images may grow proportionally with the number of images studied. Thus, the recognition problem is how to efficiently search our studied images ignoring "noise" (e.g. due to change in viewpoint, lighting variations etc.). Consider an example task where you wish to grab a beer from the fridge. You first need to get up, find the fridge and then after opening the fridge door you are presented with a range of groceries in the fridge. You must then search for the beer. Because you like your

$$(f \star g)(\tau) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f^*(t)\ g(t+\tau)\,dt,$$

beer you remember what it looks like, so you try to find the beer based on what you remember. However the beer is surrounded by other goods and the fridge light is not working. It is clear that you will have some trouble locating the beer. Even from this simple example, the challenges of visual image recognition are evident. For visual image recognition, a large number of views of each image are required due to viewpoint changes and it is necessary to recognize a large number of images, even for relatively simple tasks. Clearly, a large number of images must be gathered in order to correctly classify new objects. However, if we consider a large number of images each possibly containing a object of interest, it is impractical to search through all the images to identify a new object. The computational cost would grow with the number of images and therefore would not be even close to real time.

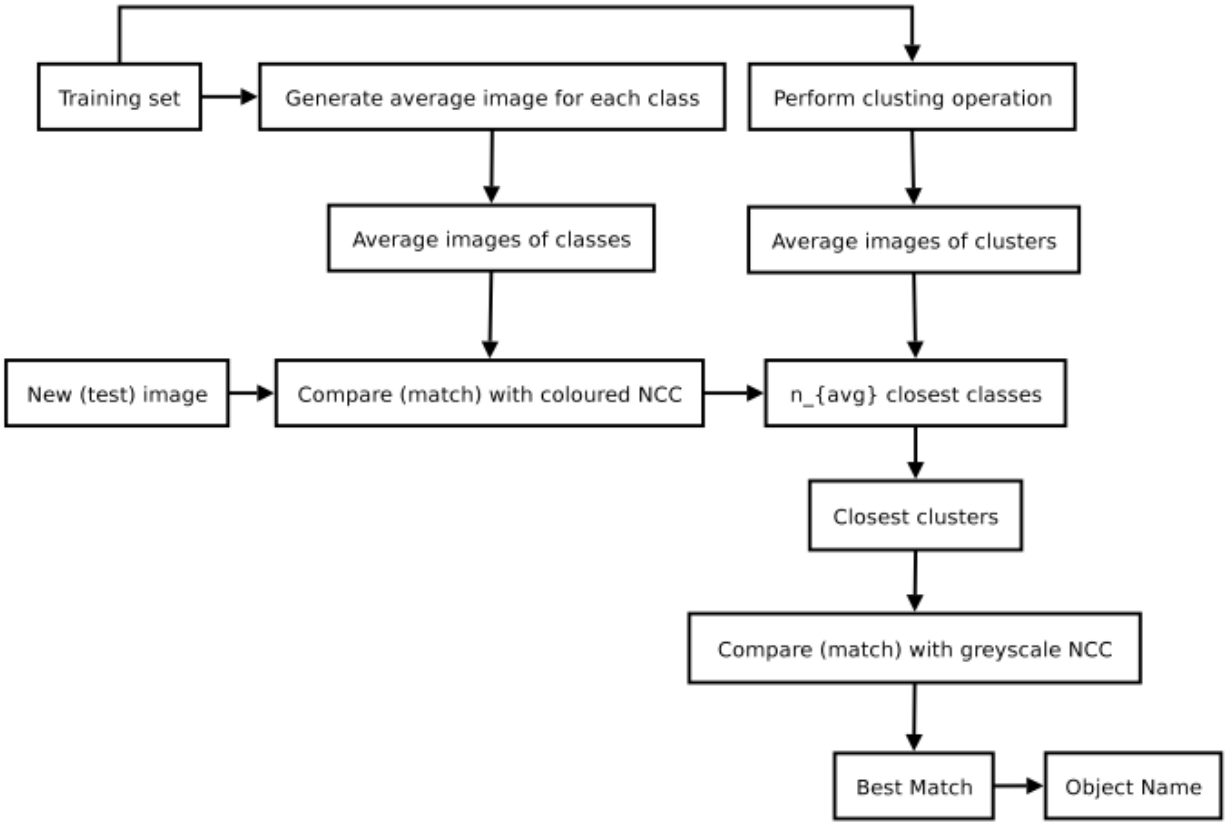


Figure 16. Simple flow chart of approach. A two-tier hierarchical approach is used: first the new image is compared with average images for each class and, second, it is compared with clustered images for some of the classes[16].

2. Existing major algorithms of face recognition and their comparative analysis

Within the last several years, numerous algorithms have been proposed for face recognition. While much progress has been made toward recognizing faces under small variations in lighting, facial expression and pose, reliable techniques for recognition under more extreme variations have proven elusive. Note that lighting variability includes not only intensity, but also direction and number of light sources. As is evident from Figure 17, the same person, with the same facial expression and seen from the same viewpoint, can appear dramatically different when light sources illuminate the face from different directions.

Figure 17. The same person seen under different lighting conditions can app ear dramatically different: In the left image, the dominant light source is nearly head-on; in the right image, the dominant light source is from above and to the right.

We will concentrate on two observations:

1. All of the images taken from a fixed viewpoint but under varying illumination lie in a 3-D linear subspace of the high-dimensional image space.
2. Because of regions of shadowing, specularities, and facial expressions, the above observation does not exactly hold. In practice, certain regions of the face may have variability from image to image that often deviates significantly from the linear subspace and, consequently, are less reliable for recognition.

We make use of these observations by finding a linear projection of the faces from the high-dimensional image space to a significantly lower dimensional feature space which is insensitive both to variation in lighting direction and facial expression. We choose projection directions that are nearly orthogonal to the within-class scatter, projecting away variations in lighting and facial expression while maintaining discriminability. Fisherfaces, a derivative of Fisher's Linear Discriminant (FLD), maximizes the ratio of between-class scatter to that of within-class scatter.

The Eigenface method is also based on linearly projecting the image space to a low dimensional feature space. However, the Eigenface method, which uses principal components analysis (PCA) for dimensionality reduction, yields projection directions that maximize the total scatter across all classes, i.e. across all images of all faces. In choosing the projection which maximizes total scatter, PCA retains unwanted variations due to lighting and facial expression. While the PCA projections are optimal for reconstruction from a low dimensional basis, they may not be optimal from a discrimination standpoint.

We should point out that Fisher's Linear Discriminant is a classical" technique in image recognition, first developed by Robert Fisher in 1936 for taxonomic classification. Depending upon the features being used, it has been applied in different ways in computer vision and even in face recognition.

Eigenfaces. As correlation methods are computationally expensive and require great amounts of storage, it is natural to pursue dimensionality reduction schemes. A technique now commonly used for dimensionality reduction in computer vision – particularly in face recognition – is principal components analysis (PCA). PCA techniques, also known as Karhunen-Loeve methods, choose a dimensionality reducing linear projection that maximizes the scatter of all projected samples.

More formally, let us consider a set of *N* sample $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ $\{X_1, X_2, \ldots, X_c\}$.
taking values in an *n*-dimensional image $\mathbf{y}_k \in \mathbb{R}^m$ space, and assume that each image belongs to one of *c* classes Let us also consider a linear transformation mapping the original *n*-dimensional image space into an *m*-dimensional feature space, where $m < n$. The new feature vectors are defined by the following linear transformation:

$$\mathbf{y}_k = W^T \mathbf{x}_k \qquad k = 1, 2, \ldots, N$$

where $W \in \mathbb{R}^{n \times m}$ is a matrix with orthonormal columns.

If the total scatter matrix S_T is defined as

$$S_T = \sum_{k=1}^{N} (\mathbf{x}_k - \boldsymbol{\mu})(\mathbf{x}_k - \boldsymbol{\mu})^T$$

(2.1)

where *c* is the number of classes $\boldsymbol{\mu} \in \mathbb{R}^n$ and is the mean image of all samples, then after applying the linear transformation W^T, the scatter of the transformed feature vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ is $W^T S_T W$. In PCA, the projection *Wopt* is chosen to maximize the determinant of the total scatter matrix of the projected samples, i.e.

$$\begin{aligned} W_{opt} &= \arg\max_W |W^T S_T W| \\ &= \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \ldots & \mathbf{w}_m \end{bmatrix} \end{aligned} \tag{2.1}$$

where $\{\mathbf{w}_i \mid i = 1, 2, \ldots, m\}$ is the set of *n*-dimensional eigenvectors of S_T corresponding to the m largest eigenvalues. Since these eigenvectors have the same dimension as the original images, they are referred to as Eigenpictures in and Eigenfaces. If classification is performed using a nearest neighbor classier in the reduced feature space and m is chosen to be the number of images *N* in the training set, then the Eigenface method is equivalent to the correlation method in the previous section.

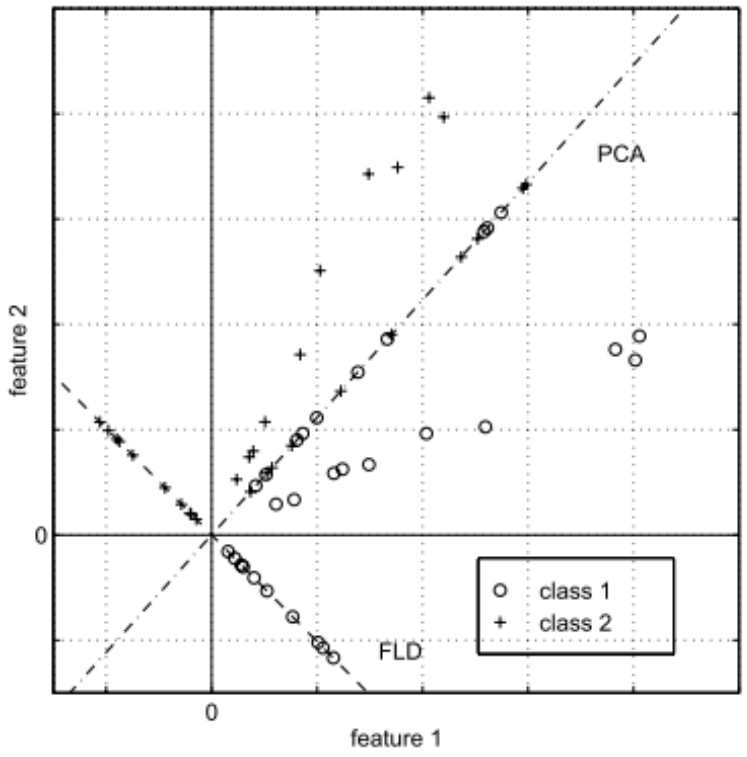
A drawback of this approach is that the scatter being maximized is due not only to the between-class scatter that is useful for classification, but also to the within-class scatter that, for classification purposes, is unwanted information. Much of the variation from one image to the next is due to illumination changes. Thus if PCA is presented with images of faces under varying illumination, the projection matrix *Wopt* will contain principal components (i.e. Eigenfaces) which retain, in the projected feature space, the variation due lighting. Consequently, the points in the projected space will not b e well clustered, and worse, the classes may be smeared together.

It has been suggested that by discarding the three most significant principal components, the variation due to lighting is reduced. The hope is that if the first principal components capture the variation due to lighting, then better clustering of projected samples is achieved by ignoring them. Yet it is unlikely that the first several principal components correspond solely to variation in lighting; as a consequence, information that is useful for discrimination may be lost.

Fisherfaces

The previous algorithm takes advantage of the fact that under admittedly idealized conditions, the variation within class lies in a linear subspace of the image space. Hence, the classes are convex and therefore linearly separable. One can perform dimensionality reduction using linear projection and still preserve linear separability. This is a strong argument in favor of using linear methods for dimensionality reduction in the face recognition problem, at least when one seeks insensitivity to lighting conditions.

Since the learning set is labeled, it makes sense to use this information to build a more reliable method for reducing the dimensionality of the feature space.



$$S_W = \sum_{i=1}^{c} \sum_{\mathbf{x}_k \in X_i} (\mathbf{x}_k - \boldsymbol{\mu}_i)(\mathbf{x}_k - \boldsymbol{\mu}_i)^T$$

Figure 18. A comparison of principal component analysis (PCA) and Fisher's linear discriminant (FLD) for a two class problem where data for each class lies near a linear subspace.

Here we argue that using class specific linear methods for dimensionality reduction and simple classiers in the reduced feature space, one gets better recognition rates than with either the Linear Subspace method or the Eigenface method. Fisher's Linear Discriminant (FLD) is an example of a class specific method, in the sense that it tries to "shape" the scatter in order to make it more reliable for classification. This method selects *W* in such a way that the ratio of the between-class scatter and the within-class scatter is maximized.

Let the between-class scatter matrix be defined as

(2.2)

and the within-class scatter matrix be defined as

(2.3)

where $\boldsymbol{\mu}_i$ is the mean image of class *Xi*, and *Ni* is the number of samples in class *Xi*. If *SW* is nonsingular, the optimal projection *Wopt* is chosen as the matrix with orthonormal columns which maximizes the ratio of the determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix of the projected samples, i.e.

$$S_B = \sum_{i=1}^{c} N_i \, (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

$$W_{opt} = \arg\max_{W} \frac{|W^T S_B W|}{|W^T S_W W|} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \dots & \mathbf{w}_m \end{bmatrix} \quad (2.4)$$

where $\{\mathbf{w}_i \mid i = 1, 2, \ldots, m\}$ is the set of generalized eigenvectors of *SB* and *SW* corresponding to the m largest generalized $\{\lambda_i \mid i = 1, 2, \ldots, m\}$ eigenvalues, i.e.

$$S_B \mathbf{w}_i = \lambda_i S_W \mathbf{w}_i \,, \qquad i = 1, 2, \ldots, m.$$

Note that there are at most c ; 1 nonzero generalized eigenvalues, and so an upper bound on misc ; 1 where c is the number of classes.

To illustrate the benefits of class specific linear projection, we constructed a low dimensional analogue to the classification problem in which the samples from each class linear a linear subspace. Figure **??** is a comparison of PCA and FLD for a two-class problem in which the samples from each class are randomly perturbed in a direction perpendicular to a linear subspace. For this example *N* = 20, *n* = 2, and *m* = 1. So the samples from each class lie near a line passing through the origin in the 2-D feature space. Both PCA and FLD have been used to project the points from 2-D down to 1-D. Comparing the two projections in the figure, PCA actually smears the classes together so that they are no longer linearly separable in the projected space. It is clear that although PCA achieves larger total scatter, FLD achieves greater between-class scatter, and consequently classification becomes easier.

In the face recognition problem one is confronted with the difficulty that the within-class scatter $S_W \in \mathbb{R}^{n \times n}$ matrix is always singular. This stems from the fact that the rank of *SW* is at most *N* – c, and in general, the number of images in the learning set *N* is much smaller than the number of pixels in each image *n*. This means that it is possible to choose the matrix *W* such that the within-class scatter of the projected samples can be made exactly zero.

In order to overcome the complication of a singular *SW*, we propose an alternative to the criterion in the above given *Wopt*. This method, which we call Fisherfaces, avoids this problem by projecting the image set to a lower dimensional space so that the resulting within-class scatter matrix *SW* is

nonsingular. This is achieved by using PCA to reduce the dimension of the feature space to *N*; c and then, applying the standard FLD defined by *Wopt* to reduce the dimension to *c* – 1. More formally, *Wopt* is given by

$$W_{opt}^T = W_{fld}^T W_{pca}^T \tag{2.5}$$

where

$$\begin{aligned} W_{pca} &= \arg\max_W |W^T S_T W| \\ W_{fld} &= \arg\max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}. \end{aligned} \tag{2.6}$$

Optimization for *Wpca* is performed over *n* x (*N*–*c*) matrices with orthonormal columns, while the optimization for *Wfld* is performed over (N–c) x *m* matrices with orthonormal columns. In computing *Wpca* we have thrown away only the smallest c principal components.

There are certainly other ways of reducing the within-class scatter while preserving between-class scatter. For example, a second method which we are currently investigating chooses W to maximize the between-class scatter of the projected samples after having first reduced the within-class scatter. Taken to an extreme, we can maximize the between-class scatter of the projected samples subject to the constraint that the within-class scatter is zero, i.e.

$$W_{opt} = \arg\max_{W \in \mathcal{W}} |W^T S_B W| \tag{2.7}$$

where *W* is the set of n x m matrices with orthonormal columns contained in the kernel of *SW*[16].

Face recognition using eigenfaces

As a general view, this algorithm extracts the relevant information of an image and encodes it as efficiently as possible. For this purpose, a collection of images from the same person is evaluated in order to obtain the variation. Mathematically, the algorithm calculates the eigenvectors of the covariance matrix of the set of face images.

Each image from the set contribute to an eigenvector, these vectors characterize the variations between the images. When we represent these eigenvectors, we call it eigenfaces. Every face can be represented as a linear combination of the eigenfaces; however, we can reduce the number of eigenfaces to the ones with greater values, so we can make it more efficient. The basic idea of the algorithm is develop a system that can compare not images themselves, but these feature weights explained before. The algorithm can be reduced to the next simple steps.

1. Acquire a database of face images, calculate the eigenfaces and determine the face space with all them. It will be necessary for further recognitions.

2. When a new image is found, calculate its set of weights.

3. Determine if the image is a face; to do so, we have to see of it is close enough to the face space.

4. Finally, it will be determined if the image corresponds to a known face of the database of not.

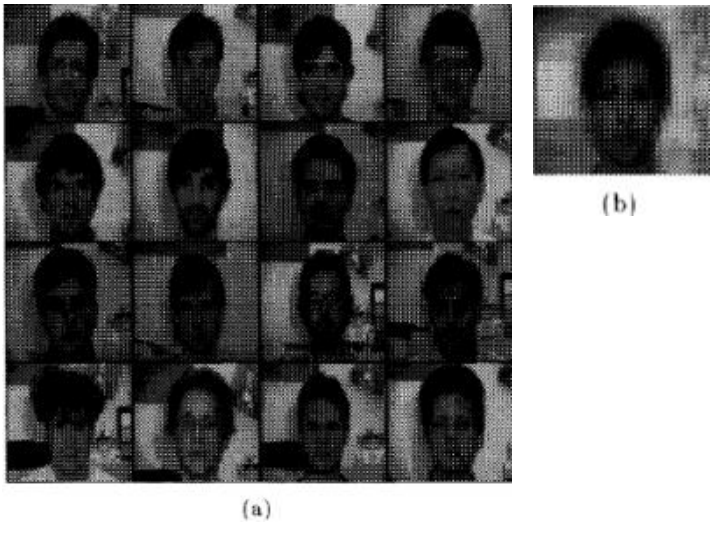


Figure 19. (a) set of face images. (b) Average of the set of images given left.

Let the training set of face $I_1, I_2, I_3 \ldots I_M$ images be . We calculate the

$$\Psi = \frac{1}{M}\sum_{N=1}^{M} I_n$$

average of the set as . In addition, the difference of each image from the average is

defined by $\Phi_i = I_i - \Psi$. We can see if Figure 19(a) a set of images and its average in Figure 19(b). Finally, we calculate the eigenvalues λk and eigenvectors uk of the $C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T$. covariance matrix.

The last step is to classify a face image. We just need to transform the new image into its eigenfaces components; i.e. its project into face space. We have to calculate the vector of $\Omega^T = [\varpi_1, \varpi_2 .. \varpi_{M'}]$, weights where $\varpi_k = u_k^T (I - \Psi)$ for *k*= 1,2... *M′*; and *M′* represents not the total eigenfaces, but the ones with greater values. The criterion to determine which the matched face image is to

(a) (b) (c)

determine the image face class k that gives the minimum Euclidean distance is the vector that describes the face image number k. We can see an example of this procedure in Figure 20. Image (a) and (b) represents the case when the input image is near face space (it is a face) and a face class (face matched). On the other hand, image (c) shows an example of an input image distant from face space (in fact, it is a flower, not a human face) and not near a known face class. We could also find an input image that is not near face space but it still is near a face class; it would be detected as a false positive and it depends on the value of threshold set to compare the Euclidean distance explained previously.

Figure 20. Three examples of input images projected on the face space.

Face recognition using line edge map

This algorithm describes a new technique based on line edge maps (LEM) to accomplish face recognition. In addition, it proposes a line matching technique to make this task possible. In opposition with other algorithms, LEM uses physiologic features from human faces to solve the problem; it mainly uses mouth, nose and eyes as the most characteristic ones.

In order to measure the similarity of human faces the face images are firstly converted into gray-level pictures. The images are encoded into binary edge maps using edge detection algorithm. This system is very similar to the way human beings perceive other people faces as it was stated in many psychological studies. The main advantage of line edge maps is the low sensitiveness to illumination changes, because it is an intermediate-level image representation derived from low-level edge map representation. The algorithm has another important improvement, it is the low memory requirements because the kind of data used. In Figure 21, there is an example of a face line edge map; it can be noticed that it keeps face features but in a very simplified level.

Figure 21. Example of face LEM.

One of the most important parts of the algorithm is the Line Segment Hausdorff Distance (LHD) described to accomplish an accurate matching of face images. This method is not oriented to calculate exact lines form different images; its main characteristic is its flexibility of size, position and orientation. Given two LEMs $M^l = \{m_1^l, m_2^l ... m_p^l, \}$ (face from the database) $T^l = \{t_1^l, t_2^l .. t_q^l\}$ and (input image to be detected); the LHD is represented by the $\overline{d}(m_i^l, t_j^l)$. The elements of this vector represent themselves three difference distance measurements: orientation distance, parallel distance and perpendicular distance respectively.

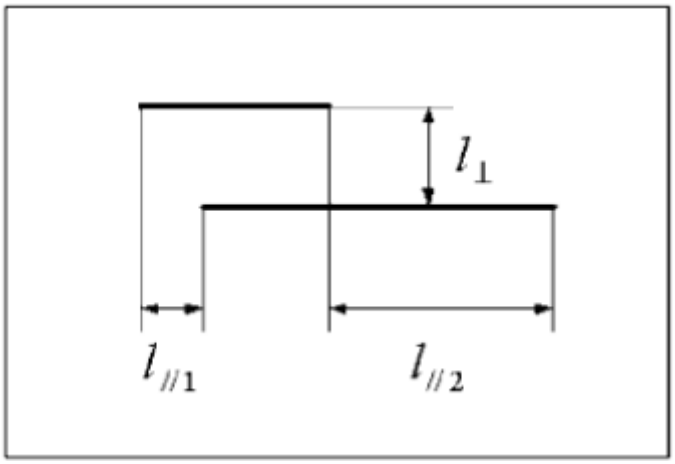


Figure 22. Practical example for calculating parallel and perpendicular distances.

$$\overline{d}(m_i^l, t_j^l) = \begin{bmatrix} d_\theta(m_i^l, t_j^l) \\ d_\parallel(m_i^l, t_j^l) \\ d_\perp(m_i^l, t_j^l) \end{bmatrix}$$

$$d_\theta(m_i^l, t_j^l) = f(\theta(m_i^l, t_j^l)) \qquad d_\parallel(m_i^l, t_j^l) = \min(l_{\parallel 1}, l_{\parallel 2}) \qquad d_\perp(m_i^l, t_j^l) = l_\perp$$

The function represents the smallest intersection angle between lines The m_i^l and t_j^l. $f(x) = x^2/W$ function f is a penalty nonlinear function that ignores smaller angles and penalizes greater ones. It can be used as the penalty function. How to calculate the parallel and perpendicular distance is shown in Figure 22. Finally, the distance between the two segments can be calculated with the next equation[17].

$$d(m_i^l, t_j^l) = \sqrt{d_\theta^2(m_i^l, t_j^l) + d_{\parallel}^2(m_i^l, t_j^l) + d_\perp^2(m_i^l, t_j^l)} \quad (2.8)$$

After having defined the distance between two lines, line segment Hausdorff distance (pLHD) is defined as

$$H_{pLHD}(M^l, T^l) = \max(h(M^l, T^l), h(T^l, M^l))$$

where $h(M^l, T^l) = \frac{1}{\sum_{m_i^l \in M^l} l_{m_i^l}} \sum_{m_i^l \in M^l} l_{m_i^l} \min_{t_x^l \in T^l} d(m_i^l, t_j^l)$ and $l_{m_i^l}$ is the length of segment m_i^l.

Comparative analysis

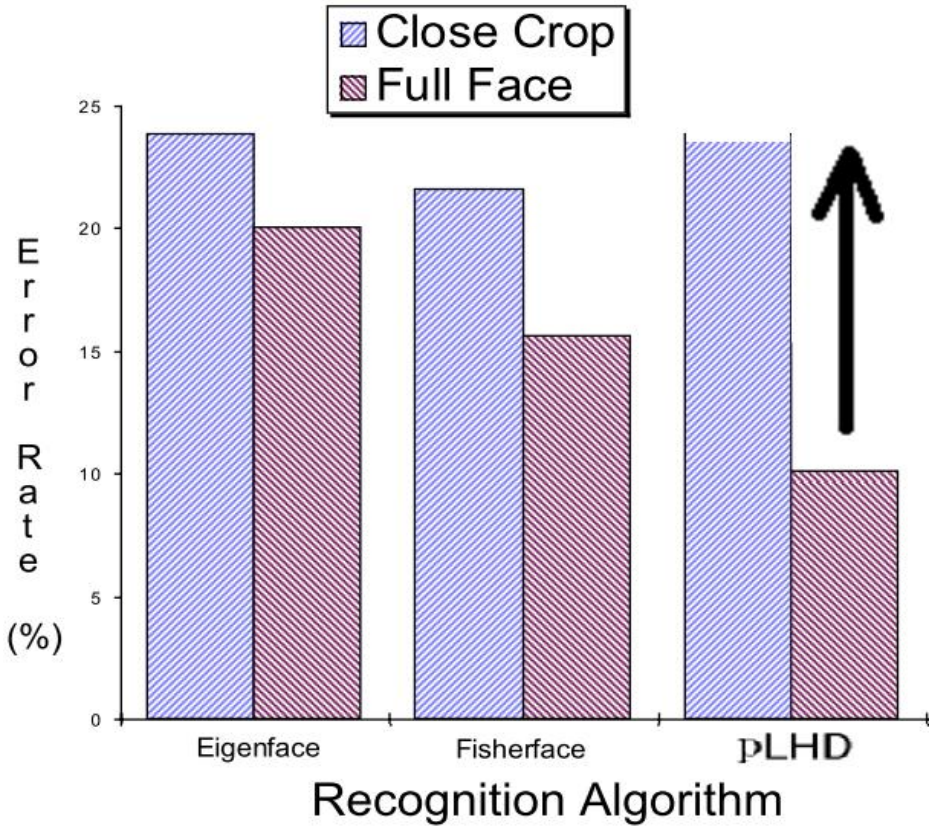


Figure 23. Chart of error rate of the recognition algorithms.

As we can witness from the above given Figure 23 the algorithm with highest error rate is Eigenface algorithm. Next algorithm Fisherface also showed good results but pLHD algorithm which is based on line Hausdorff Distance coped with the task even better. Of course pLHD is not applicable in close crop recognition but in case of full face recognition its results considerably differs from the rest of two algorithms.

Why Line Edge Map is not applicable in close crop mode?

It is because of, pLHD algorithm functions based on matching the lines of the whole Line edge map. In close crop pixel intensity of an image decreases. Respectively, edge line determination becomes nearly impossible. Thus, recognition process will fail

Now let’s look at the next table:

Method	Close Crop Error rate (%)	Full Face Error rate (%)
Eigenface	23.9	20.0
Fisherface	21.6	15.6
pLHD	n/a	10.8

Table 1. Error rate table of the recognition algorithms.

As we can see from the above given results in Figure 24 pLHD algorithm's error rate is just 10.8% and it surpasses its rival algorithms.

3. Proposed approach using unique parameter of pLHD algorithm and its comparative analysis

Based on the gained results we will focus on LHD based algorithm.

So, the main strength of pLHD is that this distance measurement is that measuring the parallel distance, we choose the minimum distance between edges. It helps when line edge is strongly detected and the other one not. It avoids shifting feature points.

However, it also has a weakness; briefly, it can confuse lines and not detect similarities that should be detected.

Therefore, in order to avoid errors, another measurement can be made. We can add a new parameter to the Hausdorff distance, which will function based on comparing the number of lines in the Line edge map images, hereinafter, exclude situations of confusing lines in LEM and detect them in proper way. It is planned that this will help to improve recognition accuracy and reduce error rates.

So, we will apply this parameter to pLHD algorithm and let's name it pLHD2.

$Xlt = Yl1, Yl2 \ldots Yln$

where, *Xlt* is a total amount of lines in LEM, *Yl1*, *Yl2*…*Yln* are edge lines in. (*Yl1*+*Yl2*)(*Yl1*-*Yl2*)=*Ela* where, ***E****la* is a value of difference or indifference between compared edge lines and it indicates the output result of edge lines comparison and it simply will not allow happening confusing situations respectively.

It sound simple but let's look at the results first:

Comparative analysis

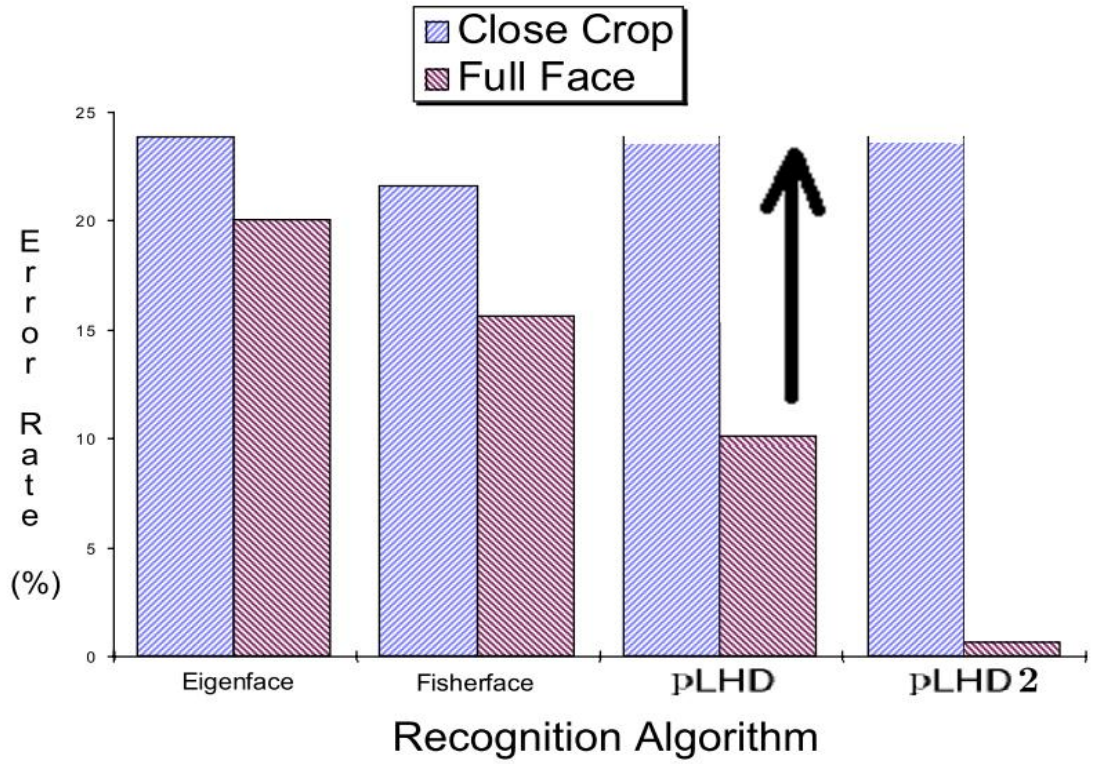


Figure 25. Error rate chart of the recognition algorithms, including pLHD2.

As you can see above in Figure 25, after integrating with unique parameter of comparing of the edge lines of LEM in order to exclude confusing situations of the missing of similarities between edge lines, the error rate indicator of pLHD algorithm dramatically decreased and it is almost at the bottom and it has only minor mistakes. It's recognition accuracy perfect as ever (see also Figure 26).

Method	Close Crop Error rate (%)	Full Face Error rate (%)
Eigenface	23.9	20.0
Fisherface	21.6	15.6
pLHD	n/a	10.8
pLHD2	n/a	0.6

Table 2. Error rate chart of the recognition algorithms, including pLHD2.

Error rate index of pLHD2 is only 0.6% and it differs from pLHD for 10.2% (fig. 26). The results came out much better that expected.

Chapter II summary

Chapter II is devoted to the investigation of the technique of Template matching, as well as existing major face recognition algorithms which are based on template matching technique. There has been conducted comprehensive review of the technical concepts as: convolution; edge detection; sum of absolute differences; metric function; euqlidean distance; cross-correlation; linear subspace etc. which promoted us to understand what doest template matching represent by itself, also, template matching algorithm has been explained both theoretically and mathematically. Furthermore, existing major face recognition algorithms as: eigenface, fisherface and pLHD have been thoroughly studied and their performances have been tested via conduction of a comparative analysis. Based on the gained results an optimal face recognition algorithm pLHD has been selected and optimized by integrating a unique proposed parameter which considerably improved its recognition accuracy. In the end, again comparative analysis has been conducted but this time including with optimized pLHD algorithm named as pLHD2 and the received results graphically demonstrated its superiority over the other recognition algorithms.

Chapter III. Face recognition software implementation

1. C# programming language in software implementation

For the development of face recognition software C# programming language has been selected. Let's briefly get acquainted with C#.

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative. C# is one of the programming languages designed for the Common Language Infrastructure. C# is built on the syntax and semantics of C++, allowing C programmers to take advantage of .NET and the common language runtime.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. The most recent version is C# 5.0, which was released on August 15, 2012.

Why this programming language have been chosen for the development of the proposed software of face recognition.

- The language, and implementations thereof, provides support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.
- The language is intended for use in developing software components suitable for deployment in distributed environments.
- Source code portability is very important, as is programmer portability, especially for those programmers already familiar with C and C++.
- Support for internationalization is very important.

- C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
- Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.

Some notable features of C# that motivated to choose it in comparison with C and C++ (and Java, where noted) are:

- C# supports strongly typed implicit variable declarations with the keyword var, and implicitly typed arrays with the keyword new followed by a collection initializer.
- Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.
- The type dynamic allows for run-time method binding, allowing for JavaScript like method calls and run-time object composition.
- C# has support for strongly-typed function pointers via the keyword delegate.
- The C# languages does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.
- Multiple inheritances are not supported, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows the programmer to implement each method depending on which interface that method is being called through, or, like Java,

allows the programmer to implement the method once and have that be the single invocation on a call through any of the class's interfaces.

- C#, unlike Java, supports operator overloading. Only the most commonly overloaded operators in C++ may be overloaded in C#.
- C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers.
- C# has explicit support for covariance and contravariance in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.

Specific functions of C# which were played an important role in the implementation of the proposed software:

Boxing and unboxing

Boxing is the operation of converting a value-type object into a value of a corresponding reference type. Boxing in C# is implicit.

Unboxing is the operation of converting a value of a reference type (previously boxed) into a value of a value type. Unboxing in C# requires an explicit type cast. A boxed object of type T can only be unboxed to a T (or a nullable T).

Example from the source-code:

```
int foo1 = 42;         // Value type.
object bar = foo1;     // foo is boxed to bar.
int foo2 = (int)bar;  // Unboxed back to value type.
```

Generics

Generics use type parameters, which make it possible to design classes and methods that do not specify the type used until the class or method is instantiated. The main advantage is that one can use generic type parameters to create classes and methods that can be used without incurring the cost of runtime casts or boxing operations.

Example from the source-code:

```
// Declare the generic class.

public class GenericList<T>
{
    void Add(T input) { }
}

class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int.
        GenericList<int> list1 = new GenericList<int>();

        // Declare a list of type string.
        GenericList<string> list2 = new GenericList<string>();

        // Declare a list of type ExampleClass.
        GenericList<ExampleClass> list3 = new
GenericList<ExampleClass>();
    }
}
```

Preprocessor

C# features "preprocessor directives based on the C preprocessor that allow programmers to define symbols, but not macros. Conditionals such as `#if`, `#endif`, and `#else` are also provided. Directives such as `#region` give hints to editors for code folding.

Example from the source-code:

```
public class Foo
{
    #region Constructors
    public Foo() {}
    public Foo(int firstParam) {}
    #endregion

    #region Procedures
    public void IntBar(int firstParam) {}
    public void StrBar(string firstParam) {}
    public void BoolBar(bool firstParam) {}
    #endregion
}
```

Libraries

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the Common Language Infrastructure (CLI), which is standardized as ECMA-335 *Common Language Infrastructure (CLI)*.

Therefore, for the implementation of the software of face recognition we compiled the following available libraries: Open-CV and Mgu-CV[18].

2. Applied libraries: Open-CV and Mgu-CV

Template matching using Open-CV:

Tasks:

- Use the OpenCV function matchTemplate to search for matches between an image patch and an input image;
- Use the OpenCV function minMaxLoc to find the maximum and minimum values (as well as their positions) in a given array.

We need two primary components:

a. *Source image (I):* The image in which we expect to find a match to the template image

b. *Template image (T):* The patch image which will be compared to the template image

Our goal is to detect the highest matching area (fig. 28):

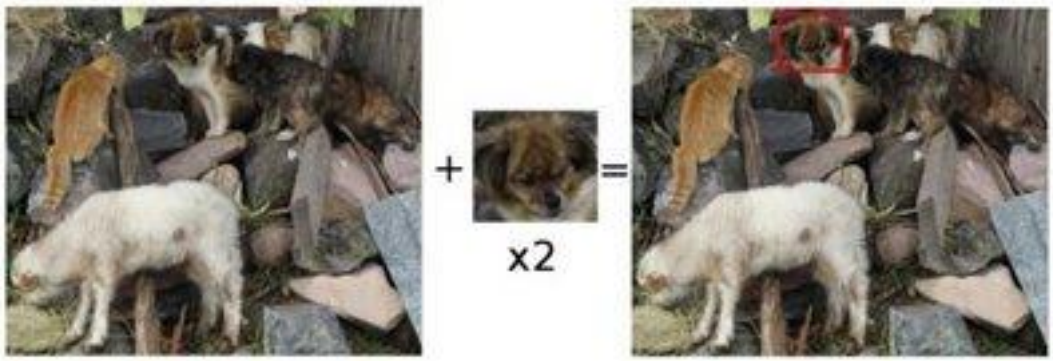


Figure 27. Highest matching area detection.

To identify the matching area, we have to *compare* the template image against the source image by sliding it:

Figure 28. Moving the patch one pixel at a time (left to right, up to down).

By *sliding*, we mean moving the patch one pixel at a time (left to right, up to down)(fig. 28). At each location, a metric is calculated so it represents how "good" or "bad" the match at that location is (or how similar the patch is to that particular area of the source image).

For each location of **T** over **I**, you *store* the metric in the *result matrix* **(R)**. Each location (x, y) in **R** contains the match metric:

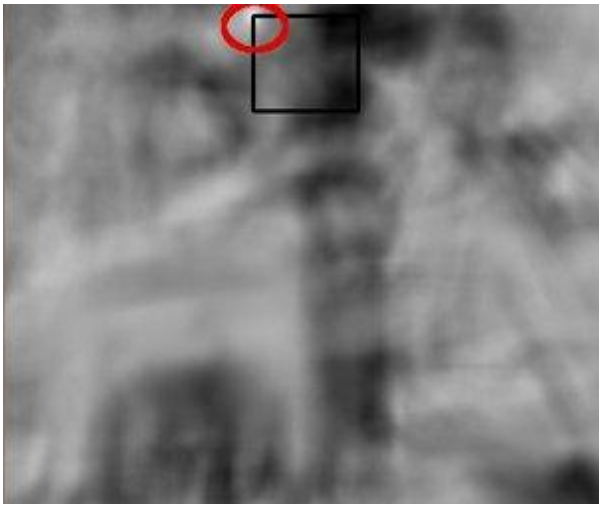


Figure 29. Result of sliding the patch

The image above (Figure 29) is the result **R** of sliding the patch with a metric **TM_CCORR_NORMED**. The brightest locations indicate the highest matches. As you can see, the location marked by the red circle is probably the one with the highest value, so that location (the rectangle formed by that point as a corner and width and height equal to the patch image) is considered the match.

In practice, we use the function minMaxLoc to locate the highest value (or lower, depending of the type of matching method) in the *R* matrix.

OpenCV implements Template matching in the function matchTemplate. The available methods are 6:

a. **method=CV_TM_SQDIFF**

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

b. **method=CV_TM_SQDIFF_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

c. **method=CV_TM_CCORR**

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

d. **method=CV_TM_CCORR_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

e. **method=CV_TM_CCOEFF**

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

where

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$$

f. **method=CV_TM_CCOEFF_NORMED**

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

What does this program do?

- Loads an input image and a image patch (*template*)
- Perform a template matching procedure by using the OpenCV function matchTemplate with any of the 6 matching methods described before. The user can choose the method by entering its selection in the Trackbar.
- Normalize the output of the matching procedure
- Localize the location with higher matching probability
- Draw a rectangle around the area corresponding to the highest match

Code at glance:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

/// Global Variables
Mat img; Mat templ; Mat result;
char* image_window = "Source Image";
char* result_window = "Result window";

int match_method;
int max_Trackbar = 5;

/// Function Headers
void MatchingMethod( int, void* );

/** @function main */
int main( int argc, char** argv )
{
  /// Load image and template
  img = imread( argv[1], 1 );
  templ = imread( argv[2], 1 );

  /// Create windows
```

```
  namedWindow( image_window, CV_WINDOW_AUTOSIZE );
  namedWindow( result_window, CV_WINDOW_AUTOSIZE );

  /// Create Trackbar
  char* trackbar_label = "Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n 2: TM
CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED";
  createTrackbar( trackbar_label, image_window, &match_method,
max_Trackbar, MatchingMethod );

  MatchingMethod( 0, 0 );

  waitKey(0);
  return 0;
}

/**
 * @function MatchingMethod
 * @brief Trackbar callback
 */
void MatchingMethod( int, void* )
{
  /// Source image to display
  Mat img_display;
  img.copyTo( img_display );

  /// Create the result matrix
  int result_cols =  img.cols - templ.cols + 1;
  int result_rows = img.rows - templ.rows + 1;

  result.create( result_cols, result_rows, CV_32FC1 );

  /// Do the Matching and Normalize
  matchTemplate( img, templ, result, match_method );
  normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );

  /// Localizing the best match with minMaxLoc
  double minVal; double maxVal; Point minLoc; Point maxLoc;
  Point matchLoc;

  minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

  /// For SQDIFF and SQDIFF_NORMED, the best matches are lower values. For
all the other methods, the higher the better
  if( match_method  == CV_TM_SQDIFF || match_method == CV_TM_SQDIFF_NORMED
)
    { matchLoc = minLoc; }
  else
    { matchLoc = maxLoc; }

  /// Show me what you got
  rectangle( img_display, matchLoc, Point( matchLoc.x + templ.cols ,
matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
  rectangle( result, matchLoc, Point( matchLoc.x + templ.cols , matchLoc.y
+ templ.rows ), Scalar::all(0), 2, 8, 0 );

  imshow( image_window, img_display );
  imshow( result_window, result );

  return;
}
```

Explanation

1. Declare some global variables, such as the image, template and result matrices, as well as the match method and the window names:

```
2.      Mat img; Mat templ; Mat result;
3.      char* image_window = "Source Image";
4.      char* result_window = "Result window";
5.
6.      int match_method;
7.      int max_Trackbar = 5;
```

8. Load the source image and template:

```
9.      img = imread( argv[1], 1 );
10.     templ = imread( argv[2], 1 );
```

11. Create the windows to show the results:

```
12.     namedWindow( image_window, CV_WINDOW_AUTOSIZE );
13.     namedWindow( result_window, CV_WINDOW_AUTOSIZE );
```

14. Create the Trackbar to enter the kind of matching method to be used. When a change is detected the callback function **MatchingMethod** is called.

```
15.     char* trackbar_label = "Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n
2: TM CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED";
16.     createTrackbar(    trackbar_label,    image_window,    &match_method,
max_Trackbar, MatchingMethod );
```

17. Wait until user exits the program.

```
18.     waitKey(0);
19.     return 0;
```

20. Let's check out the callback function. First, it makes a copy of the source image:

```
21.     Mat img_display;
22.     img.copyTo( img_display );
```

23. Next, it creates the result matrix that will store the matching results for each template location. Observe in detail the size of the result matrix (which matches all possible locations for it)

```
24.     int result_cols =  img.cols - templ.cols + 1;
25.     int result_rows = img.rows - templ.rows + 1;
```

```
26.
27.     result.create( result_cols, result_rows, CV_32FC1 );
```

28. Perform the template matching operation:

```
29.     matchTemplate( img, templ, result, match_method );
```

the arguments are naturally the input image **I**, the template **T**, the result **R** and the match_method (given by the Trackbar)

30. We normalize the results:

```
31.     normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );
```

32. We localize the minimum and maximum values in the result matrix **R** by using minMaxLoc.

```
33.     double minVal; double maxVal; Point minLoc; Point maxLoc;
34.     Point matchLoc;
35.     minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );
36.     if( match_method == CV_TM_SQDIFF || match_method == CV_TM_SQDIFF_NORMED )
37.       { matchLoc = minLoc; }
38.     else
39.       { matchLoc = maxLoc; }
```

40. Display the source image and the result matrix. Draw a rectangle around the highest possible matching area:

```
41.     rectangle( img_display, matchLoc, Point( matchLoc.x + templ.cols , matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
42.     rectangle( result, matchLoc, Point( matchLoc.x + templ.cols , matchLoc.y + templ.rows ), Scalar::all(0), 2, 8, 0 );
43.
44.     imshow( image_window, img_display );
45.     imshow( result_window, result );
```

Results

1. Testing our program with an input image such as:

Figure 30. Input image.

and a template image:

2. Generate the following result matrices (first row are the standard methods SQDIFF, CCORR and CCOEFF, second row are the same methods in its normalized version). In the first column, the darkest is the better match, for the other two columns, the brighter a location, the higher the match.

3. The right match is shown below (black rectangle around the face of the guy at the right). Notice that CCORR and CCDEFF gave erroneous best matches, however their normalized version did it right, this may be due to the fact that we are only considering the “highest match” and not the other possible high matches[23].

Figure 31. Result.

Face recognition using MguCV:

The Program is designed to use a web camera so this is essential. While the program should execute on single core machines be aware that performance may be increased by using the sequential frame processing method. The x86 source will also run on x64 machines however the x64 source is only for x64 architectures.

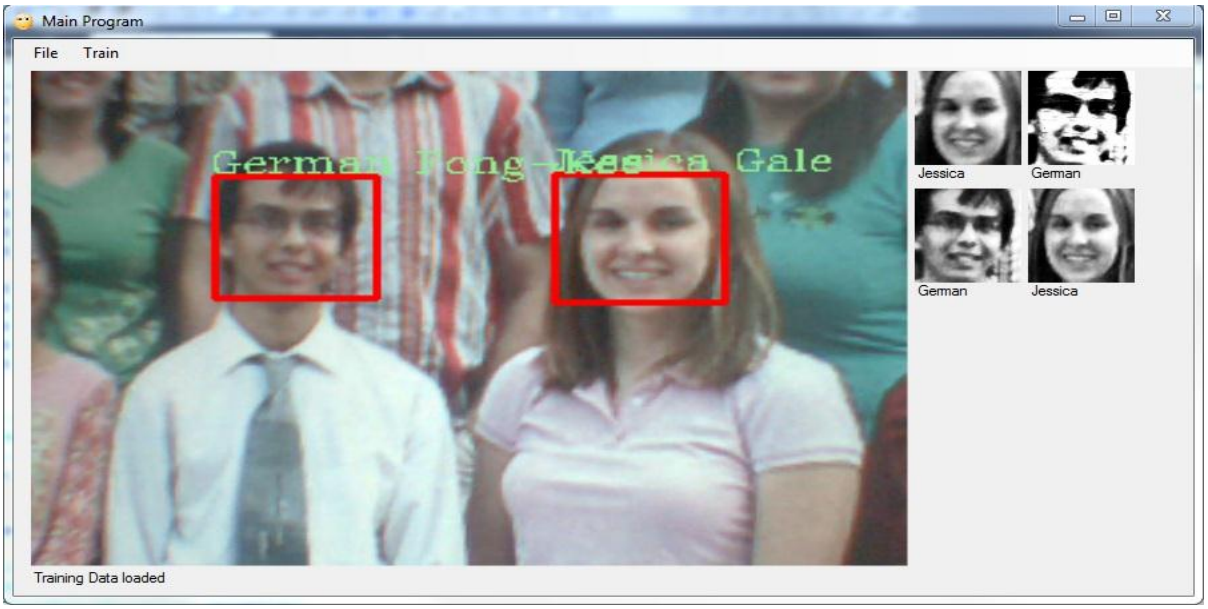


Figure 32. Recognition moment

This FaceRecognizer constructor that allows Eigen, Fisher, and pLHD classifiers to be used together. The class combines common method calls between the classifiers. The constructor for each classifier type is as follows:

- FaceRecognizer recognizer = new EigenFaceRecognizer(num_components, threshold);
- FaceRecognizer recognizer = new FisherFaceRecognizer(num_components, threshold);
- FaceRecognizer recognizer = new pLHDFaceRecognizer(radius, neighbors, grid_x, grid_y, threshold);
- FaceRecognizer recognizer = new pLHD2FaceRecognizer(radius, neighbors, grid_x, grid_y, threshold);

Eigen Classifier

The Eigen recognisor takes two variables. The 1st, is the number of components kept for this Principal Component Analysis. There's no rule how many components that should be kept for good reconstruction capabilities. It is based on your input data, so experiment with the number. OpenCV documentation suggests keeping 80 components should almost always be sufficient. The 2nd variable is designed to be a prediction threshold; this variable contains the bug as any value above this is considered as an unknown. For the Fisher and pLHD this is how unknowns are classified however with the Eigen recogniser we must use the return distance to provide our own test for unknowns. In the Eigen recogniser the larger the value returned the closer to a match we have.

To allow us to set a threshold rule later we set the threshold value to positive infinity, allowing all faces to be recognized:

```
FaceRecognizer recognizer = new EigenFaceRecognizer(80, double.PositiveInfinity);
```

We then examine the Eigen distance return after recognition, if it is above the threshold we set in the form than it is recognized if not then it's an unknown.

```
public string Recognise(Image<Gray, Byte> Input_image, int Eigen_Thresh = -1)
{
    if (_IsTrained)
    {
        FaceRecognizer.PredictionResult ER = recognizer.Predict(Input_image);

.....
        //Only use the post threshold rule if we are using an Eigen Recognizer
        //since Fisher and LBHP threshold set during the constructor will work correctly
        switch (Recognizer_Type)
        {
            case ("EMGU.CV.EigenFaceRecognizer"):
                    if (Eigen_Distance > Eigen_threshold) return Eigen_label;
                    else return "Unknown";
            case ("EMGU.CV.pLHDFaceRecognizer"):
            case ("EMGU.CV.FisherFaceRecognizer"):
            default:
                    return Eigen_label; //the threshold set in training controls unknowns
        }
    }
......
}
```

Fisher Classifier

The Fisher recogniser takes two variables as with the Eigen constructor. The 1st, is the number of components kept Linear Discriminant Analysis with the Fisherfaces criterion. It's useful to keep all components, this means the number of your training inputs. If you leave this at the default (0), set it to a value less than 0, or greater than the number of your training inputs, it will be set to the correct number (your training inputs - 1) automatically. The 2nd Variable is the threshold value for unknowns, if the resultant Eigen distance is above this value the Predict() method will return a -1 value indicating an unknown. This method works and the threshold is set to a default of 3500, change this to constrain how accurate you want results. If you change the value in the constructor the recogniser will need retraining.

```
FaceRecognizer recognizer = new FisherFaceRecognizer(0, 3500);//4000
```

As with the Eigen you can introduce your own rule as demonstrated below.

```
//NOTE: This is not within V2.4.9 of the code....
public string Recognise(Image<Gray, Byte> Input_image, int Eigen_Thresh = -1)
{
    if (_IsTrained)
    {
        FaceRecognizer.PredictionResult ER = recognizer.Predict(Input_image);

.....
            //Only use the post threshold rule if we are using an Eigen Recognizer
            //since Fisher and pLHD threshold set during the constructor will work
correctly
            switch (Recognizer_Type)
            {
                case ("EMGU.CV.EigenFaceRecognizer"):
                        if (Eigen_Distance > Eigen_threshold) return Eigen_label;
                        else return "Unknown";
                case ("EMGU.CV.FisherFaceRecognizer"):
                        //Note how the Eigen Distance must be below the threshold unlike as
above
if (Eigen_Distance < Fisher_threshold) return Eigen_label;
                        else return "Unknown";
                case ("EMGU.CV.pLHDFaceRecognizer"):
                default:
                        return Eigen_label; //the threshold set in training controls
unknowns
            }
        }
......
    }
```

pLHD Classifier

The pLHD recogniser unlike the other two takes five variables:

radius – The radius used for building the Circular Local Binary Pattern.

neighbors – The number of sample points to build a Circular Local Binary Pattern from. An value suggested by OpenCV Documentations is ‘8’ sample points. Keep in mind: the more sample points you include, the higher the computational cost.

grid_x – The number of cells in the horizontal direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.

grid_y – The number of cells in the vertical direction, 8 is a common value used in publications. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector.

threshold – The threshold applied in the prediction. If the distance to the nearest neighbour is larger than the threshold, this method returns -1.

The final variable, the threshold value works as it does in the Fisher method. If the Eigen Distance calculated is above this value the Predict() method will return a -1 value indicating an unknown. This method works and the threshold is set to a default of 100, change this to onstrain how accurate you want results. If you change the value in the constructor the recogniser will need retraining.

```
FaceRecognizer recognizer = new pLHDFaceRecognizer(1, 8, 8, 8, 100);//50
```

This is the example is provided in case you wish to add additional form controls to the threshold settings[27].

```
//NOTE: This is not within V2.4.9 of the code....
public string Recognise(Image<Gray, Byte> Input_image, int Eigen_Thresh = -1)
{
    if (_IsTrained)
    {
        FaceRecognizer.PredictionResult ER = recognizer.Predict(Input_image);

.....
            //Only use the post threshold rule if we are using an Eigen Recognizer
            //since Fisher and pLHD threshold set during the constructor will work
correctly
            switch (Recognizer_Type)
            {
                case ("EMGU.CV.EigenFaceRecognizer"):
                        if (Eigen_Distance > Eigen_threshold) return Eigen_label;
                        else return "Unknown";
                case ("EMGU.CV.pLHDFaceRecognizer"):
                        //Note how the Eigen Distance must be below the threshold unlike as
above
                        if (Eigen_Distance < pLHD_threshold) return Eigen_label;
                        else return "Unknown";
                case ("EMGU.CV.FisherFaceRecognizer"):
                default:
                        return Eigen_label; //the threshold set in training controls
unknowns
            }
    }
......
}
```

3. Software demonstration and user instructions

This software works using web-camera and the main window of the software is shown below (fig.33):

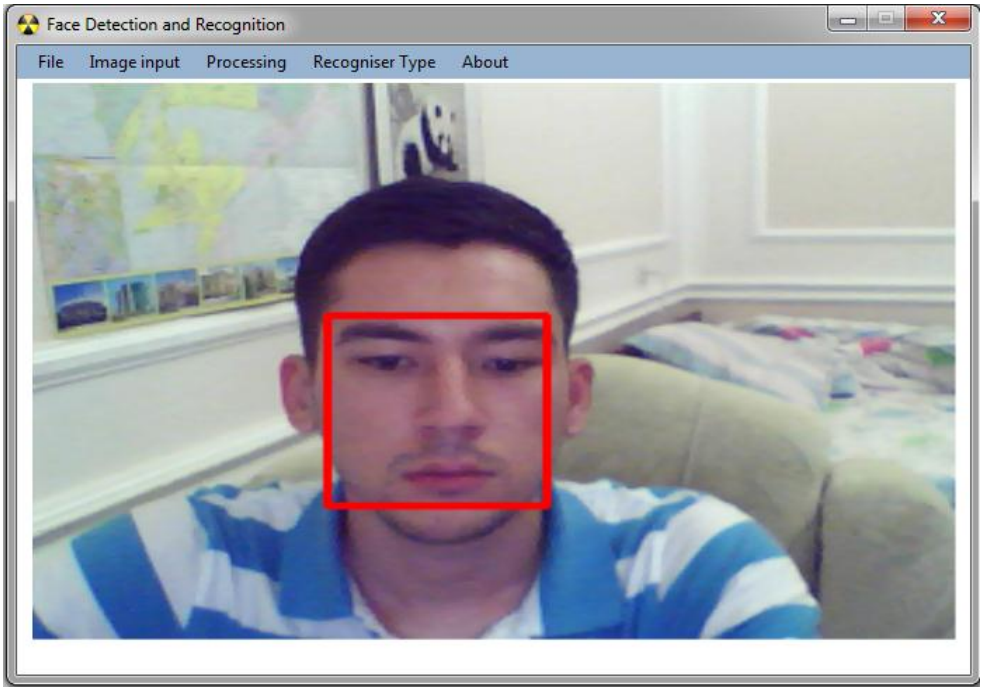


Figure 33. Main window.

The red rectangle is showing my face which means it automatically detected my face. If you click ‘Image input’ button which is at the topt left of the main window, the following window will appear(fig.33):

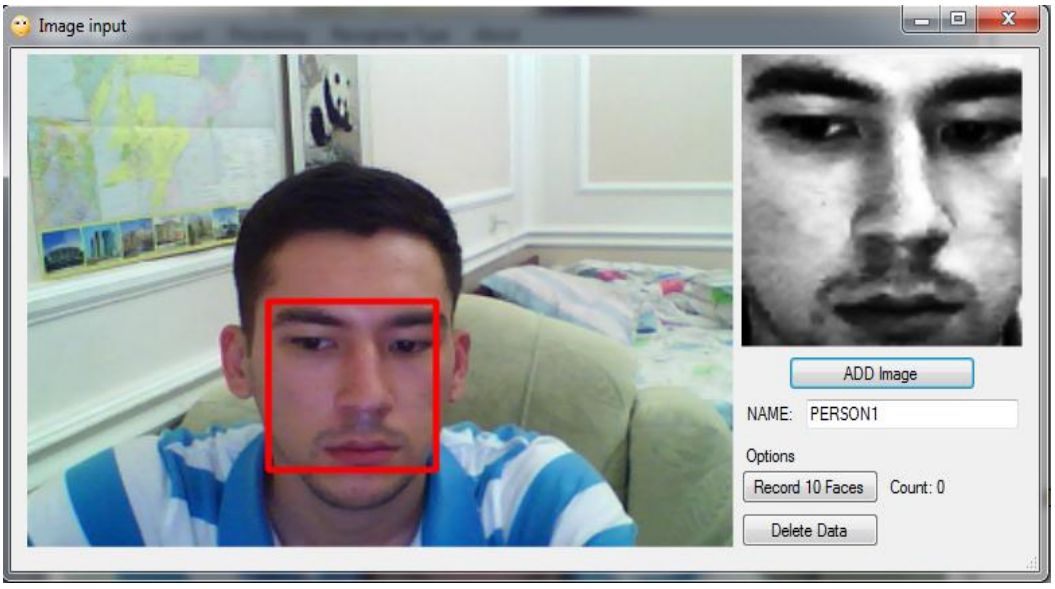


Figure 34. Image input menu.

In this window (fig.34) you will input template images. You can type the name for the input image. You can add template face captures via clicking ‘ADD Image’ button or by clicking ‘Record 10 Faces’, which will captures 10 template images at once and will store it in the database. Moreover, you can delete all the templates in the database by selecting ‘Delete Data’ button.

If you select ‘Recognizer Type’ option which is shown next(fig.35):

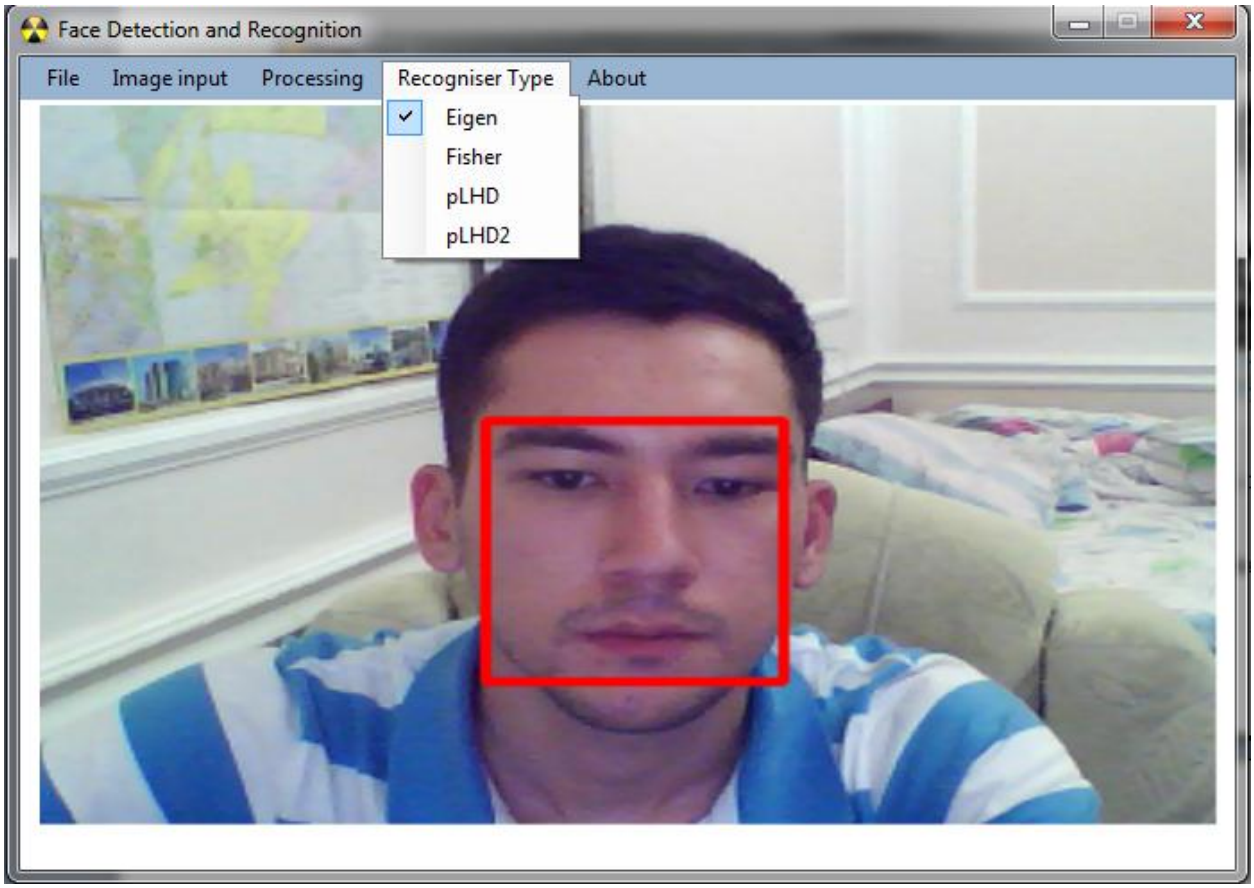


Figure 35. Selecting of recognizer type.

you will be able to select recognition algorithms as: Eigenface, Fisherface, pLHD and pLHD2 which is pLHD integrated with proposed unique parameter.

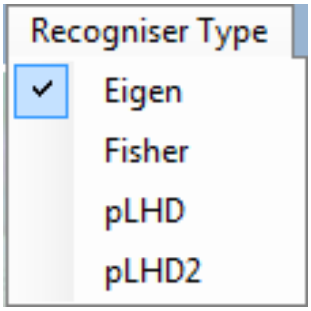


Figure 36. List Recognition algorithms.

You can find information about the author via selecting ‘About’ button which is shown in the following image:

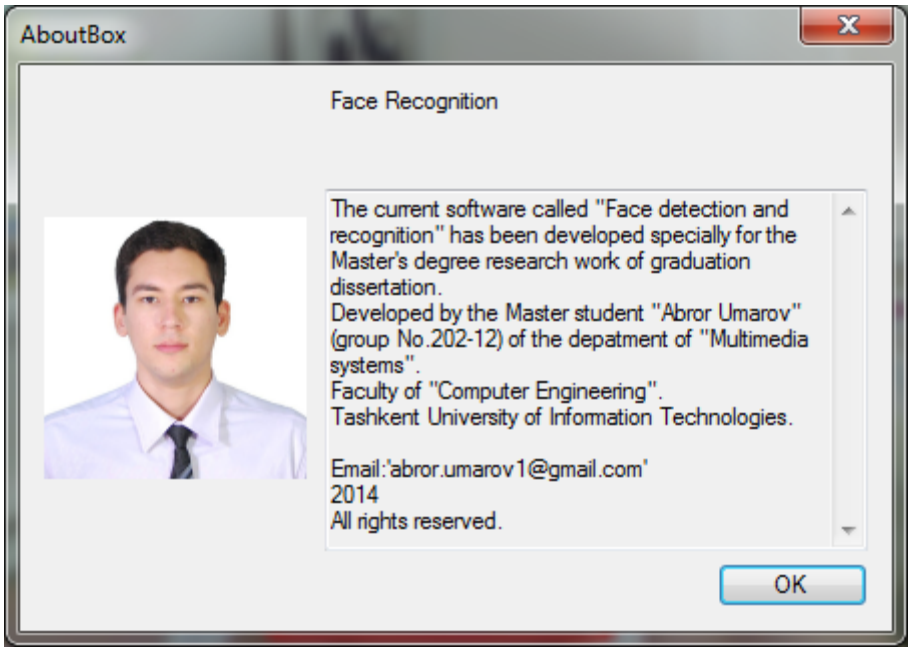


Figure 37. About author.

Let’s turn to recognition procedure using all the given algorithms.

Eigenface algorithm

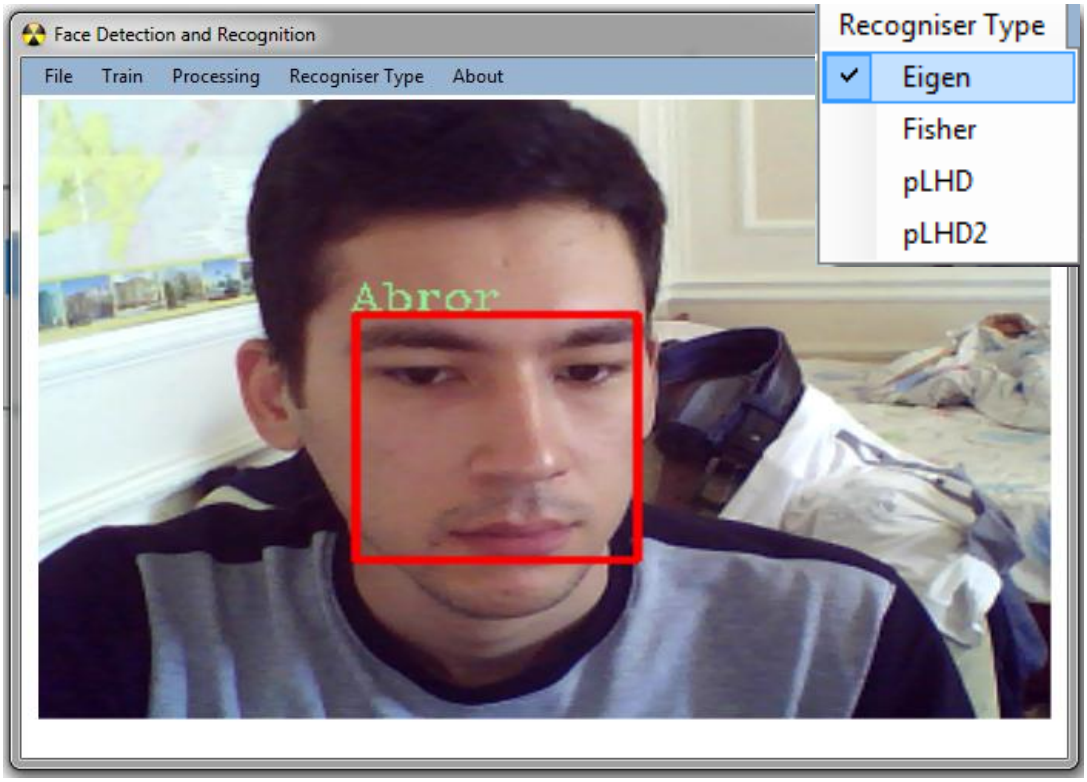


Figure 38. Recognition using Eigenface algorithm.

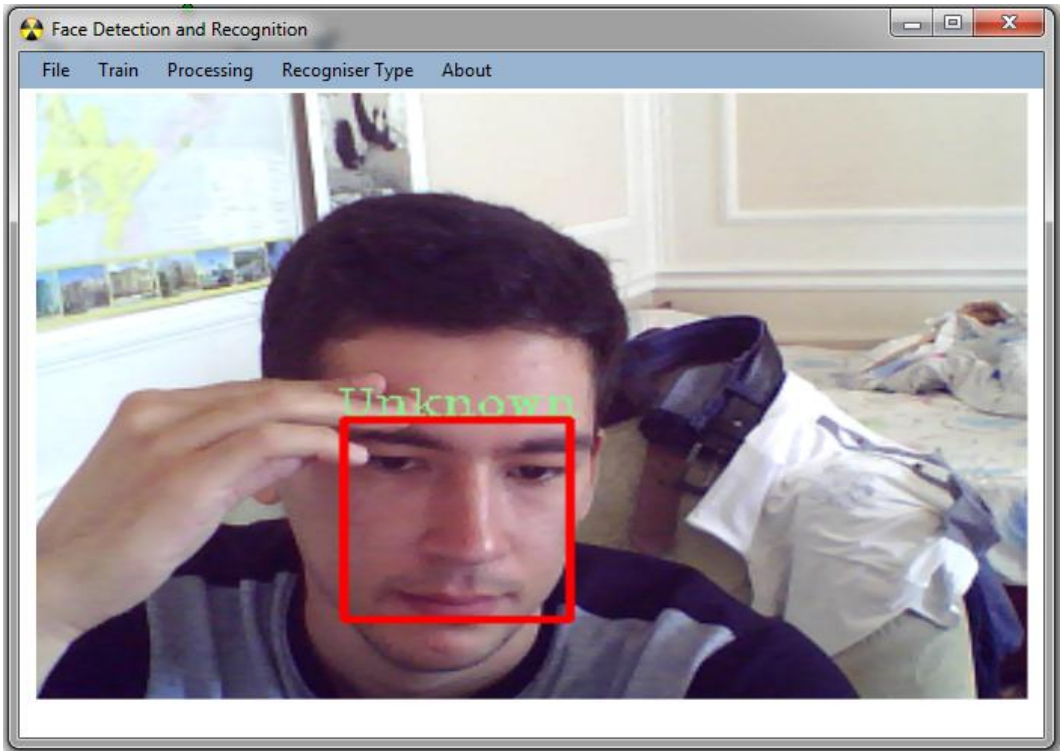


Figure 39. Recognition using Eigenface algorithm.

As you can see it recognizes well when I am looking straight but when I put my hand to my forehead it doesn't recognize (see fig.38,39).

Fisherface algorithm

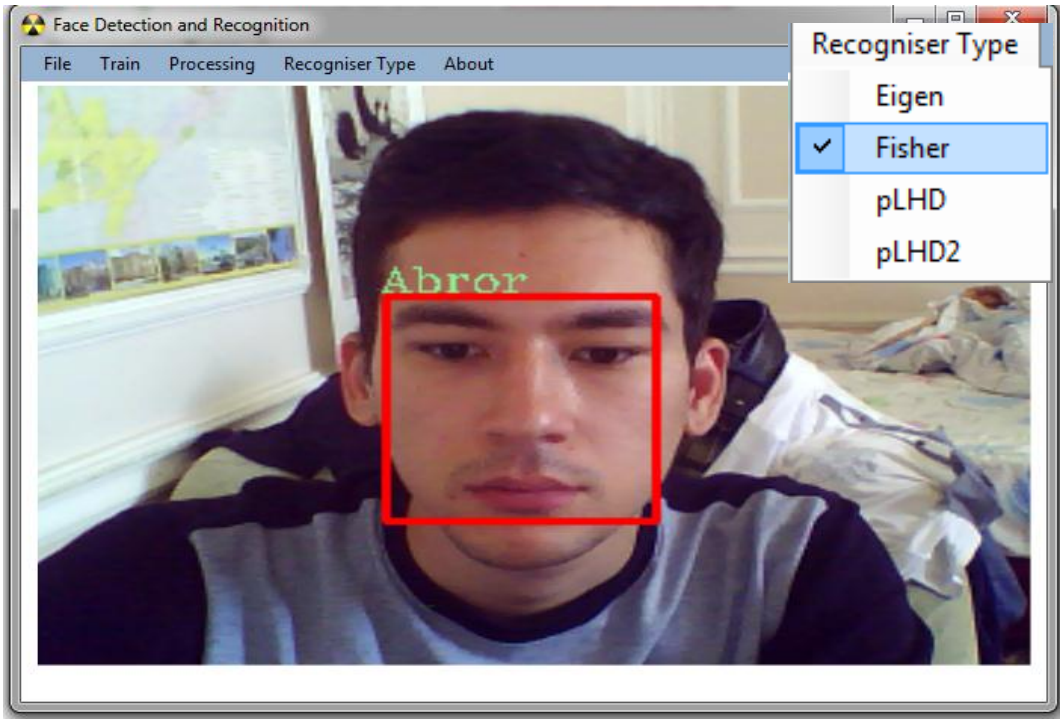


Figure 39. Recognition using Fisherface algorithm.

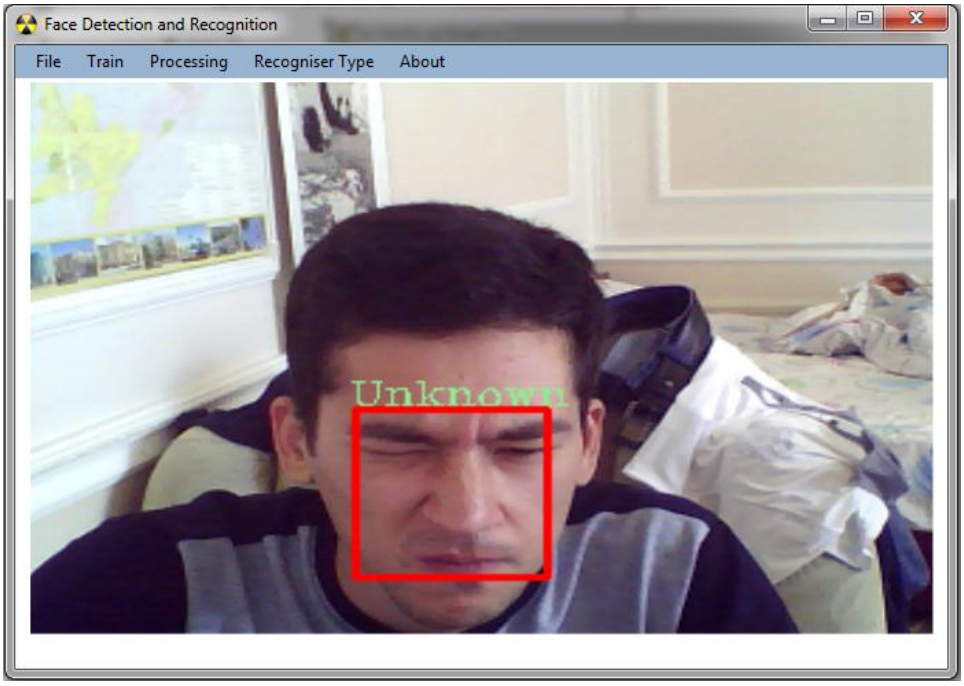


Figure 40. Recognition using Fisherface algorithm.

As you can witness Fisher is also recognizing well when I am looking straight but when I changed my facial expression it couldn't recognize (see fig.39,40).

pLHD algorithm

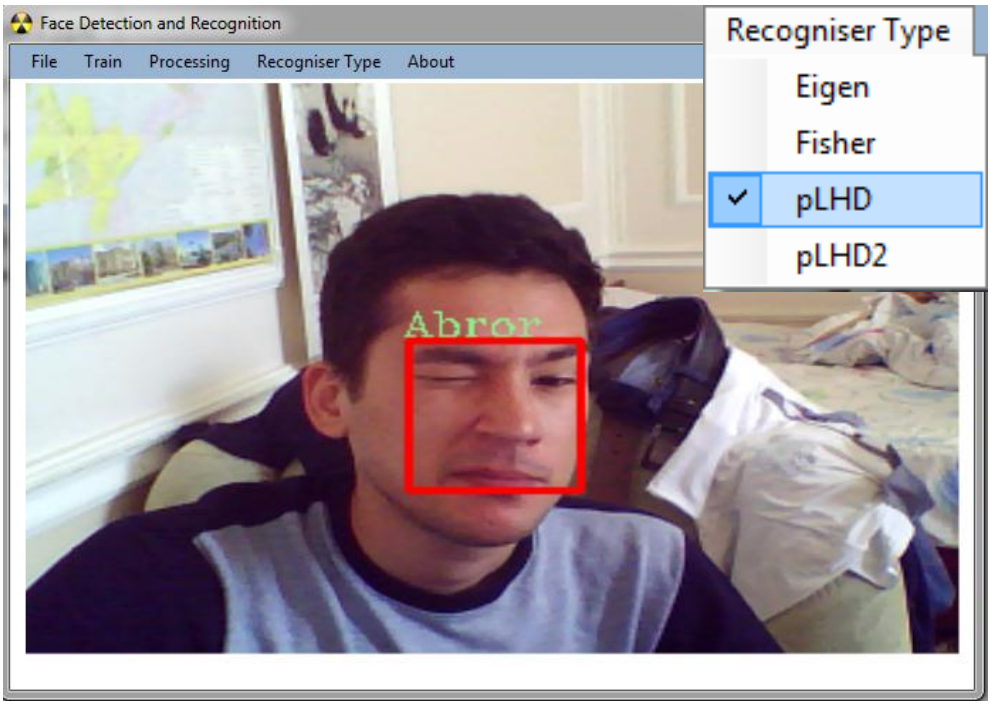


Figure 41. Recognition using pLHD algorithm.

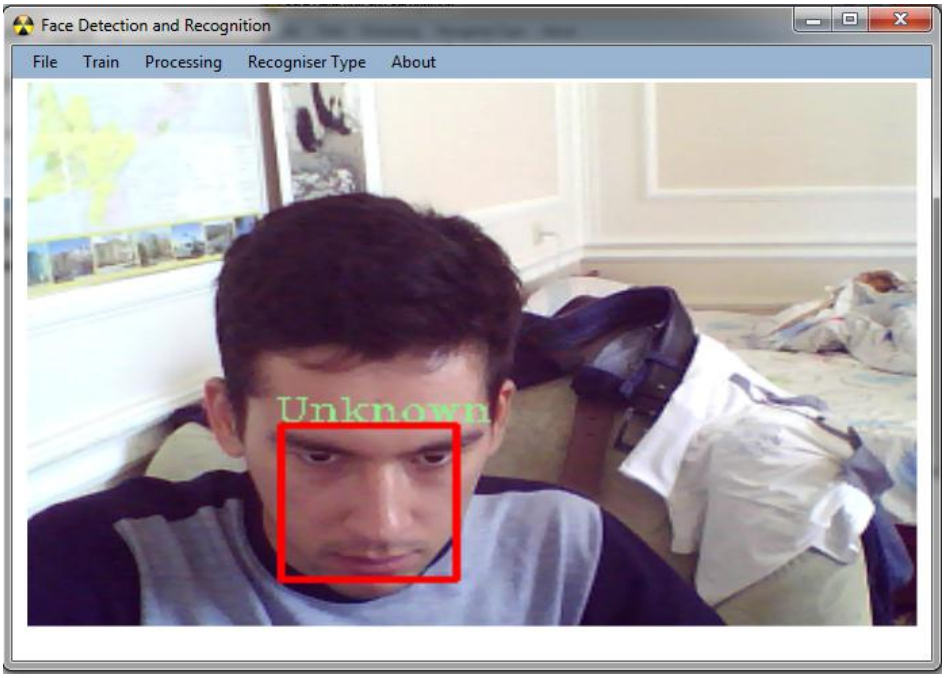


Figure 42. Recognition using pLHD algorithm.

pLHD algorithm could recognize my face even with different facial expression but couldn't recognize when I lean my head to the down side (see fig.41,42).

pLHD2 algorithm

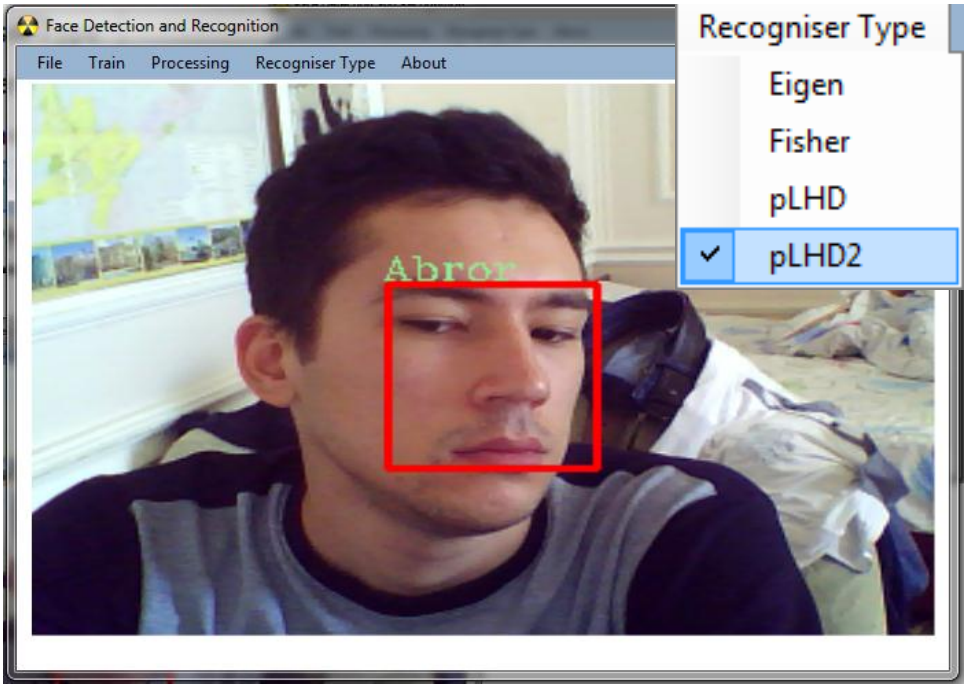


Figure 43. Recognition using pLHD2 algorithm.

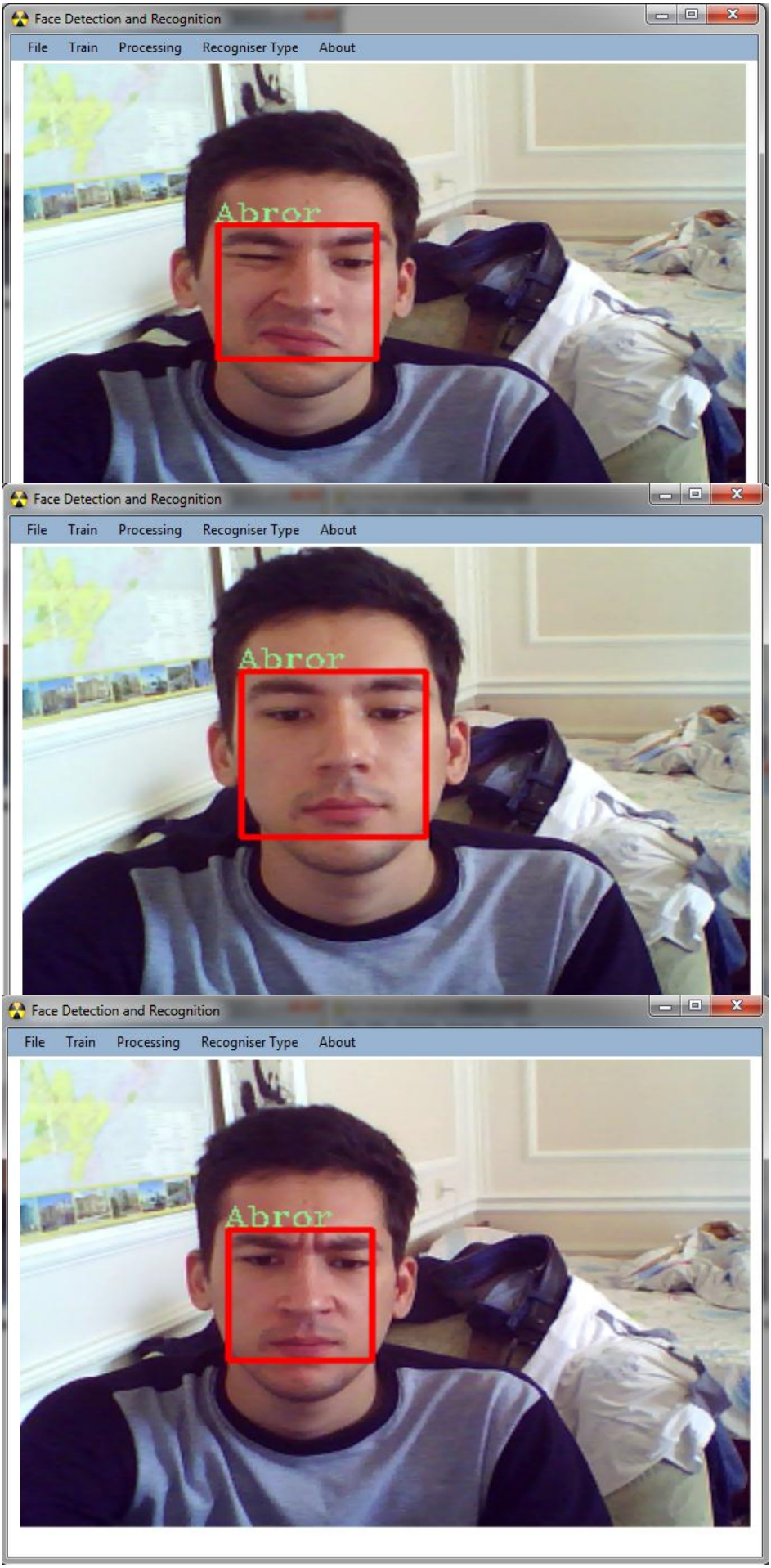


Figure 44. Recognition using pLHD2 algorithm.

As you can see pLHD2 algorithm recognizes my face even with differnt facial expressions and positions of my head (see fig.43,44).

See next for more recognition results using pLHD2 algorithm in more coplex moments and conditions:

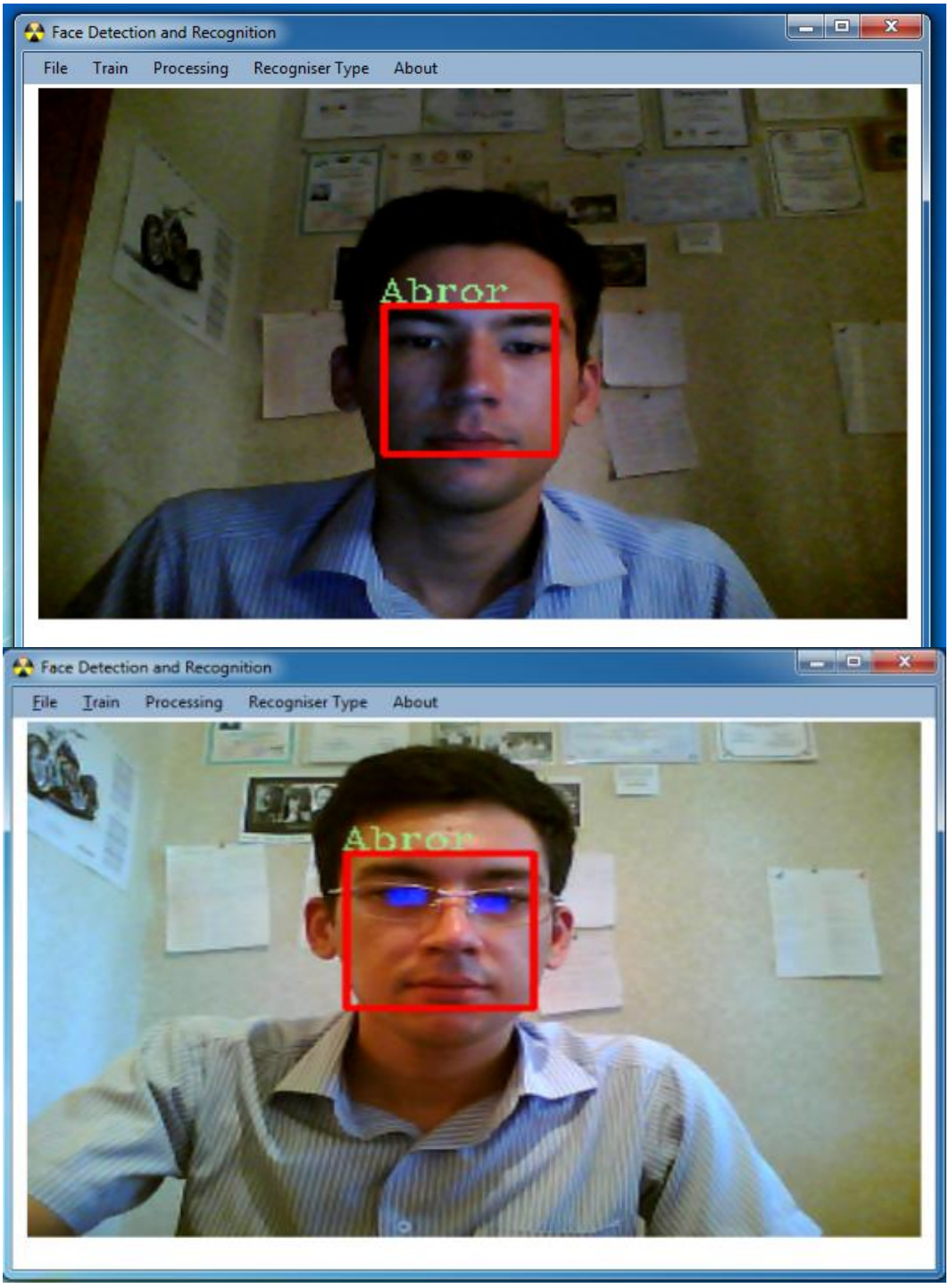


Figure 45. Miscellaneous recognitions using pLHD2 algorithm.

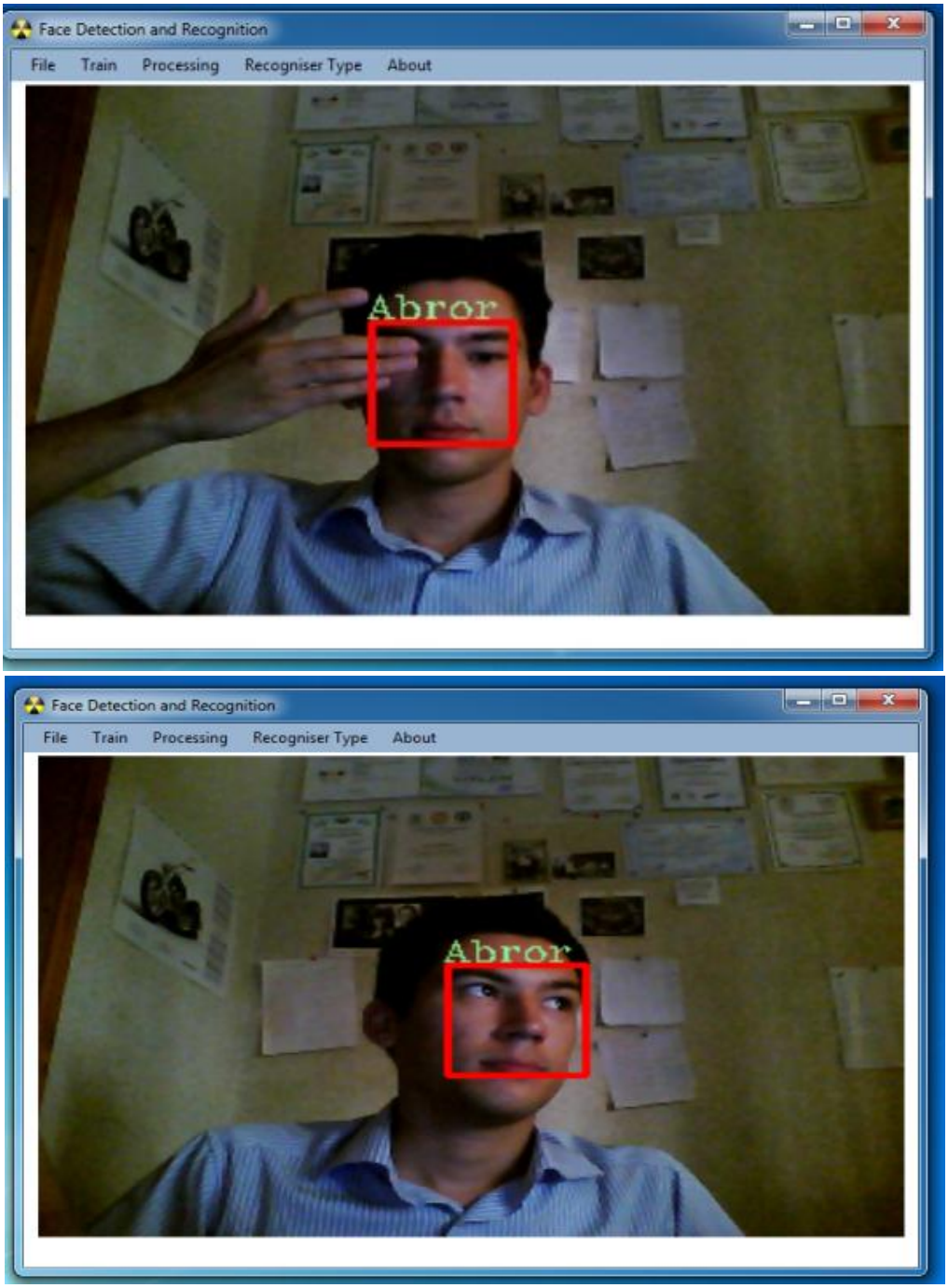


Figure 46. Miscellaneous recognitions using pLHD2 algorithm.

pLHD algorithm is still recognizing well in spite of the dark condition with bad illumination, with wearing glasses, even when I hide one of my eyes with my hand in dark condition.

As you could witness from the above demonstration, the recognition results of pLHD2 algorithm is obviously better in comparison with other major algorithms (see fig.45,46).

Chapter III summary

Chapter III is dedicated to the implementation of the software of face recognition, which functions based on the major algorithms which are overviewed in previous chapter. These algorithms are Eigenface; Fisherface; pLHD and pLHD2 respectively. In this chapter have been overviewed C# programming language and its useful features which promoted to implement the proposed software. Moreover, compiled libraries as OpenCV and EmguCV based on C# have been used for the realization of the template matching technique and combining of the major recognition algorithms in one software. In the end of third chapter, software demonstration has been thoroughly presented and user instructions have been explained.

Conclusion

In this master's dissertation main attention paid to the investigation of the methods of Image recognition using template matching algorithms based on face recognition technology. The existing recognition algorithms thoroughly have been studied. Based on the gained knowledge a comparative analysis between major algorithms has been conducted and its results have been received.

Based on the results, a unique parameter has been applied to one of the existing major image recognition algorithm. Furthermore, using unique proposed method, face recognition software has been developed and demonstrated its optimal results in comparison with other major algorithms.

References

1. The decrees of the Republic of Uzbekistan

1. The decree of the Republic of Uzbekistan “On Informatiozation” from December 11, 2003.

2. The decree of President of the Republic of Uzbekistan “On further development of computerization and implementation of the information-communication technologies” from May 30, 2002.

3. The decree of the Republic of Uzbekistan “on building of the state committee of communication, informatiozation and telecommunication technologies” from October 16, 2012.

4. The decree of President of the Republic of Uzbekistan “On further implementation of modern information-communication technologies and its development” from March 21, 20012.

2. Literatures

5. Bahram Javidi. “Image recognition and classification: algorithms, systems, and applications”. CRC Press, June 14, 2002.

6. Christopher M. Bishop. “Pattern recognition and machine learning”. Springer, October 1, 2007.

7. Richard O. Duda, Peter E. Hart, David G. Stork. Pattern Classification (Pt.1). Springer, May 7, 2006.

8. Kevin P. Murphy. “Machine learning: a probabilistic perspective”. The Mit Press, August 24, 2012.

3. Scientific technical publications

9. P.J.Phillips, R.B. McCabe and R. Chelappa, “Biometric Image Processing and Recognition”, European Signal Processing Conference, 1998.

10. Amir Faizi, “Robust Face Detection Using Template Matching Algorithm”, //University of Toronto 2008.

11. W. Zhao, R. Chelappa, A.Rosenfeld, P.J. Phillips, “Face Recognition: A Literature Survey”, //ACM Computing Surveys, 2003.

12. T. Mahalakshmi, R. Muthaiah and P. Swaminathan, “An Overview of Template Matching Technique in Image Processing”, //Research Journal of Applied Sciences, Engineering and Technology; December 15, 2012. ISSN: 2040-7467.

13. Luke Cole, David Austin, Lance Cole, “Visual Object Recognition using Template Matching”, //Australian National University 2004.

14. Bishop, Christopher M. “Pattern Recognition in Maching Learning”. //Springer. P. vii. (2006)

4. Internet links

15. http://en.wikipedia.org/wiki/Pattern_recognition

16. http://www.fidis.net/resources/deliverables/hightechid/int-d37001/doc/

17. http://www.facerecognition.it/index.asp

18. http://en.wikipedia.org/wiki/Template_matching

19. http://en.wikipedia.org/wiki/Convolution

20. http://en.wikipedia.org/wiki/Edge_detection

21. http://en.wikipedia.org/wiki/Sum_of_absolute_differences

22. http://rkb.home.cern.ch/rkb/AN16pp/node169.html#168

23. http://en.wikipedia.org/wiki/Euclidean_distance

24. http://en.wikipedia.org/wiki/Cross-correlation

25. http://en.wikipedia.org/wiki/Linear_subspace

26. http://open-cv.com

27. http://mgu-cv.com

APPLICATION

ClassifierTrain.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;
using System.Xml;
using Emgu.CV;
using Emgu.CV.Structure;

namespace Face_Recognition
{
    /// <summary>
    /// Desingned to remove the training a EigenObjectRecognizer code from the main
form
    /// </summary>
    class ClassifierTrain: IDisposable
    {

        #region Variables

        //Eigen
        //EigenObjectRecognizer recognizer;
        FaceRecognizer recognizer;

        //training variables
        List<Image<Gray, byte>> trainingImages = new List<Image<Gray,
byte>>();//Images
        //TODO: see if this can be combined in Ditionary format this will remove
support for old data
        List<string> Names_List = new List<string>(); //labels
        List<int> Names_List_ID = new List<int>();
        int ContTrain, NumLabels;
        float Eigen_Distance = 0;
        string Eigen_label;
        int Eigen_threshold = 2000;

        //Class Variables
        string Error;
        bool _IsTrained = false;

        public string Recognizer_Type = "EMGU.CV.EigenFaceRecognizer";
        #endregion

        #region Constructors
        /// <summary>
        /// Default Constructor, Looks in (Application.StartupPath + "\\TrainedFaces")
for traing data.
        /// </summary>
        public ClassifierTrain()
        {
            _IsTrained = LoadTrainingData(Application.StartupPath + "\\TrainedFaces");
        }

        /// <summary>
        /// Takes String input to a different location for training data
        /// </summary>
        /// <param name="Training_Folder"></param>
        public ClassifierTrain(string Training_Folder)
        {
```

```
            _IsTrained = LoadTrainingData(Training_Folder);
        }
        #endregion

        #region Public
        /// <summary>
        /// Retrains the recognizer witout resetting variables like recognizer type.
        /// </summary>
        /// <returns></returns>
        public bool Retrain()
        {
            return _IsTrained = LoadTrainingData(Application.StartupPath +
"\\TrainedFaces");
        }
        /// <summary>
        /// Retrains the recognizer witout resetting variables like recognizer type.
        /// Takes String input to a different location for training data.
        /// </summary>
        /// <returns></returns>
        public bool Retrain(string Training_Folder)
        {
            return _IsTrained = LoadTrainingData(Training_Folder);
        }

        /// <summary>
        /// <para>Return(True): If Training data has been located and Eigen Recogniser
has been trained</para>
        /// <para>Return(False): If NO Training data has been located of error in
training has occured</para>
        /// </summary>
        public bool IsTrained
        {
            get { return _IsTrained; }
        }

        /// <summary>
        /// Recognise a Grayscale Image using the trained Eigen Recogniser
        /// </summary>
        /// <param name="Input_image"></param>
        /// <returns></returns>
        public string Recognise(Image<Gray, byte> Input_image, int Eigen_Thresh = -1)
        {
            if (_IsTrained)
            {
                FaceRecognizer.PredictionResult ER = recognizer.Predict(Input_image);

                if (ER.Label == -1)
                {
                    Eigen_label = "Unknown";
                    Eigen_Distance = 0;
                    return Eigen_label;
                }
                else
                {
                    Eigen_label = Names_List[ER.Label];
                    Eigen_Distance = (float)ER.Distance;
                    if (Eigen_Thresh > -1) Eigen_threshold = Eigen_Thresh;

                    //Only use the post threshold rule if we are using an Eigen
Recognizer
                    //since Fisher and LBHP threshold set during the constructor will
work correctly
                    switch (Recognizer_Type)
```

```
                    {
                        case ("EMGU.CV.EigenFaceRecognizer"):
                            if (Eigen_Distance > Eigen_threshold) return Eigen_label;
                            else return "Unknown";
                        case ("EMGU.CV.PLHDFaceRecognizer"):
                        case ("EMGU.CV.FisherFaceRecognizer"):
                        default:
                            return Eigen_label; //the threshold set in training
controls unknowns
                    }



                }

            }
            else return "";
        }

        /// <summary>
        /// Sets the threshold confidence value for string Recognise(Image<Gray, byte>
Input_image) to be used.
        /// </summary>
        public int Set_Eigen_Threshold
        {
            set
            {
                //NOTE: This is still not working correctley
                //recognizer.EigenDistanceThreshold = value;
                Eigen_threshold = value;
            }
        }

        /// <summary>
        /// Returns a string containg the recognised persons name
        /// </summary>
        public string Get_Eigen_Label
        {
            get
            {
                return Eigen_label;
            }
        }

        /// <summary>
        /// Returns a float confidence value for potential false clasifications
        /// </summary>
        public float Get_Eigen_Distance
        {
            get
            {
                //get eigenDistance
                return Eigen_Distance;
            }
        }

        /// <summary>
        /// Returns a string contatining any error that has occured
        /// </summary>
        public string Get_Error
        {
            get { return Error; }
```

```
}

/// <summary>
/// Saves the trained Eigen Recogniser to specified location
/// </summary>
/// <param name="filename"></param>
public void Save_Eigen_Recogniser(string filename)
{
    recognizer.Save(filename);

    //save label data as this isn't saved with the network
    string direct = Path.GetDirectoryName(filename);
    FileStream Label_Data = File.OpenWrite(direct + "/Labels.xml");
    using (XmlWriter writer = XmlWriter.Create(Label_Data))
    {
        writer.WriteStartDocument();
        writer.WriteStartElement("Labels_For_Recognizer_sequential");
        for (int i = 0; i < Names_List.Count; i++)
        {
            writer.WriteStartElement("LABEL");
            writer.WriteElementString("POS", i.ToString());
            writer.WriteElementString("NAME", Names_List[i]);
            writer.WriteEndElement();
        }

        writer.WriteEndElement();
        writer.WriteEndDocument();
    }
    Label_Data.Close();
}

/// <summary>
/// Loads the trained Eigen Recogniser from specified location
/// </summary>
/// <param name="filename"></param>
public void Load_Eigen_Recogniser(string filename)
{
    //Lets get the recogniser type from the file extension
    string ext = Path.GetExtension(filename);
    switch (ext)
    {
        case (".PLHD"):
            Recognizer_Type = "EMGU.CV.PLHDFaceRecognizer";
            recognizer = new PLHDFaceRecognizer(1, 8, 8, 8, 100);//50
            break;
        case (".FFR"):
            Recognizer_Type = "EMGU.CV.FisherFaceRecognizer";
            recognizer = new FisherFaceRecognizer(0, 3500);//4000
            break;
        case (".EFR"):
            Recognizer_Type = "EMGU.CV.EigenFaceRecognizer";
            recognizer = new EigenFaceRecognizer(80, double.PositiveInfinity);
            break;
    }

    //introduce error checking
    recognizer.Load(filename);

    //Now load the labels
    string direct = Path.GetDirectoryName(filename);
    Names_List.Clear();
    if (File.Exists(direct + "/Labels.xml"))
    {
```

```
            FileStream filestream = File.OpenRead(direct + "/Labels.xml");
            long filelength = filestream.Length;
            byte[] xmlBytes = new byte[filelength];
            filestream.Read(xmlBytes, 0, (int)filelength);
            filestream.Close();

            MemoryStream xmlStream = new MemoryStream(xmlBytes);

            using (XmlReader xmlreader = XmlTextReader.Create(xmlStream))
            {
                while (xmlreader.Read())
                {
                    if (xmlreader.IsStartElement())
                    {
                        switch (xmlreader.Name)
                        {
                            case "NAME":
                                if (xmlreader.Read())
                                {
                                    Names_List.Add(xmlreader.Value.Trim());
                                }
                                break;
                        }
                    }
                }
            }
            ContTrain = NumLabels;
        }
        _IsTrained = true;

    }

    /// <summary>
    /// Dispose of Class call Garbage Collector
    /// </summary>
    public void Dispose()
    {
        recognizer = null;
        trainingImages = null;
        Names_List = null;
        Error = null;
        GC.Collect();
    }

    #endregion

    #region Private
    /// <summary>
    /// Loads the traing data given a (string) folder location
    /// </summary>
    /// <param name="Folder_location"></param>
    /// <returns></returns>
    private bool LoadTrainingData(string Folder_location)
    {
        if (File.Exists(Folder_location +"\\TrainedLabels.xml"))
        {
            try
            {
                //message_bar.Text = "";
                Names_List.Clear();
                Names_List_ID.Clear();
                trainingImages.Clear();
```

```
                    FileStream filestream = File.OpenRead(Folder_location +
"\\TrainedLabels.xml");
                    long filelength = filestream.Length;
                    byte[] xmlBytes = new byte[filelength];
                    filestream.Read(xmlBytes, 0, (int)filelength);
                    filestream.Close();

                    MemoryStream xmlStream = new MemoryStream(xmlBytes);

                    using (XmlReader xmlreader = XmlTextReader.Create(xmlStream))
                    {
                        while (xmlreader.Read())
                        {
                            if (xmlreader.IsStartElement())
                            {
                                switch (xmlreader.Name)
                                {
                                    case "NAME":
                                        if (xmlreader.Read())
                                        {
                                            Names_List_ID.Add(Names_List.Count); //0,
1, 2, 3....

                                            Names_List.Add(xmlreader.Value.Trim());
                                            NumLabels += 1;
                                        }
                                        break;
                                    case "FILE":
                                        if (xmlreader.Read())
                                        {
                                            //PROBLEM HERE IF TRAININGG MOVED
                                            trainingImages.Add(new Image<Gray,
byte>(Application.StartupPath + "\\TrainedFaces\\" + xmlreader.Value.Trim()));
                                        }
                                        break;
                                }
                            }
                        }
                    }
                    ContTrain = NumLabels;

                    if (trainingImages.ToArray().Length != 0)
                    {



                        //-------------------------------------------------------------
-----------------------------------------------------------------------



                        switch (Recognizer_Type)
                        {
                            case ("EMGU.CV.PLHDFaceRecognizer"):
                                recognizer = new PLHDFaceRecognizer(1, 8, 8, 8,
100);//50
                                break;
                            case ("EMGU.CV.FisherFaceRecognizer"):
                                recognizer = new FisherFaceRecognizer(0, 3500);//4000
                                break;
                            case("EMGU.CV.EigenFaceRecognizer"):
                            default:
                                recognizer = new EigenFaceRecognizer(80,
double.PositiveInfinity);
```

```
                                break;
                            }

                            recognizer.Train(trainingImages.ToArray(),
Names_List_ID.ToArray());
                            // Recognizer_Type = recognizer.GetType();
                            // string v = recognizer.ToString();
//EMGU.CV.FisherFaceRecognizer || EMGU.CV.EigenFaceRecognizer ||
EMGU.CV.PLHDFaceRecognizer

                            return true;
                        }
                        else return false;
                    }
                    catch (Exception ex)
                    {
                        Error = ex.ToString();
                        return false;
                    }
                }
                else return false;
            }

            #endregion
        }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using Emgu.CV.UI;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;

using System.IO;
using System.Xml;
using System.Runtime.InteropServices;
using System.Threading;
using System.Security.Principal;
using System.Threading.Tasks;
using Microsoft.Win32.SafeHandles;

namespace Face_Recognition
{
    public partial class MainForm : Form
    {
        #region variables
        Image<Bgr, Byte> currentFrame; //current image aquired from webcam for display
        Image<Gray, byte> result, TrainedFace = null; //used to store the result image
and trained face
        Image<Gray, byte> gray_frame = null; //grayscale current image aquired from
webcam for processing

        Capture grabber; //This is our capture variable
```

```
        public CascadeClassifier Face = new CascadeClassifier(Application.StartupPath
+ "/Cascades/haarcascade_frontalface_default.xml");//Our face detection method

        MCvFont font = new MCvFont(FONT.CV_FONT_HERSHEY_COMPLEX, 0.5, 0.5); //Our
fount for writing within the frame

        int NumLabels;

        //Classifier with default training location
        ClassifierTrain Eigen_Recog = new ClassifierTrain();

        #endregion

        public MainForm()
        {
            InitializeComponent();

            //Load of previus trainned faces and labels for each image

            if (Eigen_Recog.IsTrained)
            {
                message_bar.Text = "Training Data loaded";
            }
            else
            {
                message_bar.Text = "No training data found, please train program using
Train menu option";
            }
            initialise_capture();

        }

        //Open training form and pass this
        private void trainToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Stop Camera
            stop_capture();

            //OpenForm
            TrainingForm TF = new TrainingForm(this);
            TF.ShowDialog();
        }
        public void retrain()
        {

            Eigen_Recog = new ClassifierTrain();
            if (Eigen_Recog.IsTrained)
            {
                message_bar.Text = "Training Data loaded";
            }
            else
            {
                message_bar.Text = "No training data found, please train program using
Train menu option";
            }
        }

        //Camera Start Stop
        public void initialise_capture()
        {
            grabber = new Capture();
            grabber.QueryFrame();
            //Initialize the FrameGraber event
```

```
            if (parrellelToolStripMenuItem.Checked)
            {
                Application.Idle += new EventHandler(FrameGrabber_Parrellel);
            }
            else
            {
                Application.Idle += new EventHandler(FrameGrabber_Standard);
            }
        }
        private void stop_capture()
        {
            if (parrellelToolStripMenuItem.Checked)
            {
                Application.Idle -= new EventHandler(FrameGrabber_Parrellel);
            }
            else
            {
                Application.Idle -= new EventHandler(FrameGrabber_Standard);
            }
            if(grabber!= null)
            {
            grabber.Dispose();
            }
        }

        //Process Frame
        void FrameGrabber_Standard(object sender, EventArgs e)
        {
            //Get the current frame form capture device
            currentFrame = grabber.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);

            //Convert it to Grayscale
            if (currentFrame != null)
            {
                gray_frame = currentFrame.Convert<Gray, Byte>();

                //Face Detector
                Rectangle[] facesDetected = Face.DetectMultiScale(gray_frame, 1.2, 10,
new Size(50, 50), Size.Empty);

                //Action for each element detected
                for (int i = 0; i < facesDetected.Length; i++)// (Rectangle face_found
in facesDetected)
                {
                    //This will focus in on the face from the haar results its not
perfect but it will remove a majoriy
                    //of the background noise
                    facesDetected[i].X += (int)(facesDetected[i].Height * 0.15);
                    facesDetected[i].Y += (int)(facesDetected[i].Width * 0.22);
                    facesDetected[i].Height -= (int)(facesDetected[i].Height * 0.3);
                    facesDetected[i].Width -= (int)(facesDetected[i].Width * 0.35);

                    result = currentFrame.Copy(facesDetected[i]).Convert<Gray,
byte>().Resize(130, 130, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                    result._EqualizeHist();
                    //draw the face detected in the 0th (gray) channel with blue color
                    currentFrame.Draw(facesDetected[i], new Bgr(Color.Blue), 2);

                    if (Eigen_Recog.IsTrained)
                    {
                        string name = Eigen_Recog.Recognise(result);
                        int match_value = (int)Eigen_Recog.Get_Eigen_Distance;
```

```
                        //Draw the label for each face detected and recognized
                        currentFrame.Draw(name + " ", ref font, new
Point(facesDetected[i].X - 2, facesDetected[i].Y - 2), new Bgr(Color.LightGreen));
                        ADD_Face_Found(result, name, match_value);
                    }
                }
                //Show the faces procesed and recognized
                image_PICBX.Image = currentFrame.ToBitmap();
            }
        }

        void FrameGrabber_Parrellel(object sender, EventArgs e)
        {
            //Get the current frame form capture device
            currentFrame = grabber.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);

            //Convert it to Grayscale
            //Clear_Faces_Found();

            if (currentFrame != null)
            {
                gray_frame = currentFrame.Convert<Gray, Byte>();
                //Face Detector
                Rectangle[] facesDetected = Face.DetectMultiScale(gray_frame, 1.2, 10,
new Size(50, 50), Size.Empty);
             //   CircleF eyeDetected = eye
                //Action for each element detected
                Parallel.For(0, facesDetected.Length, i =>
                    {
                        try
                        {
                            facesDetected[i].X += (int)(facesDetected[i].Height *
0.15);
                            facesDetected[i].Y += (int)(facesDetected[i].Width *
0.22);
                            facesDetected[i].Height -= (int)(facesDetected[i].Height *
0.3);
                            facesDetected[i].Width -= (int)(facesDetected[i].Width *
0.35);

                            result = currentFrame.Copy(facesDetected[i]).Convert<Gray,
byte>().Resize(100, 100, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
                            result._EqualizeHist();
                            //draw the face detected in the 0th (gray) channel with
blue color
                            currentFrame.Draw(facesDetected[i], new Bgr(Color.Red),
2);

                            if (Eigen_Recog.IsTrained)
                            {
                                string name = Eigen_Recog.Recognise(result);
                                int match_value = (int)Eigen_Recog.Get_Eigen_Distance;

                                //Draw the label for each face detected and recognized
                                currentFrame.Draw(name + " ", ref font, new
Point(facesDetected[i].X - 2, facesDetected[i].Y - 2), new Bgr(Color.LightGreen));
                              //  ADD_Face_Found(result, name, match_value);

                            }

                        }
```

```
                        catch
                        {
                            //do nothing as parrellel loop buggy
                            //No action as the error is useless, it is simply an error
in
                            //no data being there to process and this occurss
sporadically
                        }
                    });
                //Show the faces procesed and recognized
                image_PICBX.Image = currentFrame.ToBitmap();
            }
        }

        //ADD Picture box and label to a panel for each face
        int faces_count = 0;
        int faces_panel_Y = 0;
        int faces_panel_X = 0;

        void Clear_Faces_Found()
        {
            this.Faces_Found_Panel.Controls.Clear();
            faces_count = 0;
            faces_panel_Y = 0;
            faces_panel_X = 0;
        }
        void ADD_Face_Found(Image<Gray, Byte> img_found, string name_person, int
match_value)
        {
            PictureBox PI = new PictureBox();
            PI.Location = new Point(faces_panel_X, faces_panel_Y);
            PI.Height = 80;
            PI.Width = 80;
            PI.SizeMode = PictureBoxSizeMode.StretchImage;
            PI.Image = img_found.ToBitmap();
            Label LB = new Label();
            LB.Text = name_person + " " + match_value.ToString();
            LB.Location = new Point(faces_panel_X, faces_panel_Y + 80);
            //LB.Width = 80;
            LB.Height = 15;

            this.Faces_Found_Panel.Controls.Add(PI);
            this.Faces_Found_Panel.Controls.Add(LB);
            faces_count++;
            if (faces_count == 2)
            {
                faces_panel_X = 0;
                faces_panel_Y += 100;
                faces_count = 0;
            }
            else faces_panel_X += 85;

            if (Faces_Found_Panel.Controls.Count > 10)
            {
                Clear_Faces_Found();
            }

        }

        //Menu Opeartions
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Dispose();
```

```
        }
        private void singleToolStripMenuItem_Click(object sender, EventArgs e)
        {
            parrellelToolStripMenuItem.Checked = false;
            singleToolStripMenuItem.Checked = true;
            Application.Idle -= new EventHandler(FrameGrabber_Parrellel);
            Application.Idle += new EventHandler(FrameGrabber_Standard);
        }
        private void parrellelToolStripMenuItem_Click(object sender, EventArgs e)
        {
            parrellelToolStripMenuItem.Checked = true;
            singleToolStripMenuItem.Checked = false;
            Application.Idle -= new EventHandler(FrameGrabber_Standard);
            Application.Idle += new EventHandler(FrameGrabber_Parrellel);

        }
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog SF = new SaveFileDialog();
            //As there is no identification in files to recogniser type we will set
the extension ofthe file to tell us
            switch (Eigen_Recog.Recognizer_Type)
            {
                case ("EMGU.CV.PLHDFaceRecognizer"):
                    SF.Filter = "PLHDFaceRecognizer File (*.PLHD)|*.PLHD";
                    break;
                case ("EMGU.CV.FisherFaceRecognizer"):
                    SF.Filter = "FisherFaceRecognizer File (*.FFR)|*.FFR";
                    break;
                case ("EMGU.CV.EigenFaceRecognizer"):
                    SF.Filter = "EigenFaceRecognizer File (*.EFR)|*.EFR";
                    break;
            }
            if (SF.ShowDialog() == DialogResult.OK)
            {
                Eigen_Recog.Save_Eigen_Recogniser(SF.FileName);
            }
        }
        private void loadToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog OF = new OpenFileDialog();
            OF.Filter = "EigenFaceRecognizer File (*.EFR)|*.EFR|PLHDFaceRecognizer
File (*.PLHD)|*.PLHD|FisherFaceRecognizer File (*.FFR)|*.FFR";
            if (OF.ShowDialog() == DialogResult.OK)
            {
                Eigen_Recog.Load_Eigen_Recogniser(OF.FileName);
            }
        }

        //Unknow Eigen face calibration
        private void Eigne_threshold_txtbxChanged(object sender, EventArgs e)
        {
            try
            {
                Eigen_Recog.Set_Eigen_Threshold =
Math.Abs(Convert.ToInt32(Eigne_threshold_txtbx.Text));
                message_bar.Text = "Eigen Threshold Set";
            }
            catch
            {
                message_bar.Text = "Error in Threshold input please use int";
            }
        }
```

```
        private void eigenToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Uncheck other menu items
            fisherToolStripMenuItem.Checked = false;
            pLHDToolStripMenuItem.Checked = false;
            pLHDToolStripMenuItem.Checked = false;

            Eigen_Recog.Recognizer_Type = "EMGU.CV.EigenFaceRecognizer";
            Eigen_Recog.Retrain();
        }

        private void fisherToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Uncheck other menu items
            pLHDToolStripMenuItem.Checked = false;
            eigenToolStripMenuItem.Checked = false;
            pLHDToolStripMenuItem.Checked = false;

            Eigen_Recog.Recognizer_Type = "EMGU.CV.FisherFaceRecognizer";
            Eigen_Recog.Retrain();
        }

        private void pLHDToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Uncheck other menu items
            fisherToolStripMenuItem.Checked = false;
            eigenToolStripMenuItem.Checked = false;
            pLHDToolStripMenuItem.Checked = false;

            Eigen_Recog.Recognizer_Type = "EMGU.CV.EigenFaceRecognizer";
            Eigen_Recog.Retrain();

        }

        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutBox sd = new AboutBox();
            sd.ShowDialog();
        }

        private void pLHDToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Uncheck other menu items
            fisherToolStripMenuItem.Checked = false;
            eigenToolStripMenuItem.Checked = false;
            pLHDToolStripMenuItem.Checked = false;

            Eigen_Recog.Recognizer_Type = "EMGU.CV.PLHDFaceRecognizer";
            Eigen_Recog.Retrain();
        }
    }
}
```