AN EFFICIENT HYBRID REAL TIME FACE RECOGNITION ALGORITHM IN JAVA ENVIRONMENT

M. A. Abdou*, M. H. Fathy**
*Informatics Research Institute, City for Scientific Research and Technology Applications (SRTA-City), **EGYPT**
Electrical Engineering Department, Pharos University in Alexandria, Alexandria, **EGYPT

ABSTRACT

Image processing on mobile smart phones is a new and exciting field with many challenges due to limited hardware and connectivity problems. Android based mobile phones are now becoming the core of many applications. This paper develops a real time face recognition application model for smart phones. This introduced model uses a hybrid skin color-eigen face detection method and an interest point localization for feature matching. The paper is coded in JAVA programming language to fulfill Android smart phones. Results are shown and compared with existing open source techniques for verification. The aim is to maintain real time measures with high recognition rate. Applications range from security to people with disabilities adaptation.

Keywords: Face recognition, Real time, Eigenface, JAVA Code, Interest Point Detection, OpenCV.

INTRODUCTION

The conventional architecture of a computer vision system should take into considerations essential modules. First, a sensor that comprises a digital camera based in a CCD or CMOS device. Second, an acquisition interface for data transfer such as specific frame grabbers, Camera Link, IEEE1394, USB, or the more recent GigE standard is necessary. The processing unit comes as a mandatory unit with different OS options. Finally user interface and other communication peripherals: screen, mouse, and keyboard for the input Mobile is selected according to the user environment.

Image Processing in Mobile Phones

Smartphone could be seen as a combination of personal digital assistant (PDA) and simple mobile phone. Since the launch of the Android operating system (OS) - developed by Google and is based upon the Linux kernel and GNU software- in 2007 [1], mobile development tools has been high in demand [2]. Every day, the hardware of smartphones is increasing in processing power and storage capacity, i.e. which means reducing the gap with desktop computers. On the other hand, the popularity of operating systems oriented to third party applications (Android, iOS, etc.) brings to developers access to SDKs and NDKs for native applications development (e.g. C++, Java,…) [3]. Furthermore, attention from academic and industrial forums is increasing due to rapid growth of application markets such as Apple Store or Android Market [4]. Thus, computer vision applications and toolkits are profiting from this revolution; leading to numerous possibilities to implement advanced applications based on portability and sensors' integration. One of these fields is face recognition that we are concerned with in our research [5].

Image Acquisition refers to the capturing of image data by a camera and transforming the image into data to be processed. Over the last decade, image acquisition, visualization, and computer vision capabilities have been integrated in mobile phones. Since the computational power of most of these devices was limited, the area is challenging. However, this situation is changing with the emerging market of smartphones. Once the image data is acquired, Pre-Processing often includes rendering the acquired data to a format that can be handled by a set of algorithms for different image processing applications such as feature extraction, as shown in figure (1). Feature extraction is used to make spatial decisions or notify a user of an event such as an augmented reality device. Extracted features are tracked over time to render some temporal statistics to make decisions based on existing data base, such as in early warning or surveillance devices.

Application that uses camera usually involves an image processing method such as Gaussian, Median, Mean Laplacian, Sobel filter and others. Developers who have basic knowledge about image processing can write their own codes to apply those image processing methods in their application but for the one who does not have any basic about image processing will face a lot of difficulties creating their applications. Developers usually prefer to import libraries in their work.

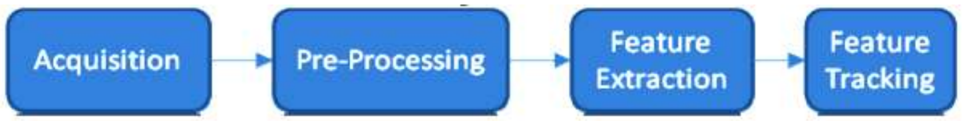


Figure (1): Mobile image processing block diagram.

Operating System and Software Technology

Due to its recent growth in popularity with varying hardware manufacturers such as HTC, Motorola, and Samsung, the Android operating system (OS) is preferable for smart phone applications. The Android OS is supported and a part of the Open Handset Alliance; positions key manufacturers, cellular providers and the Android OS in a collaborative environment which has caused large growth since October 2008 when the first Android mobile phone was released. Recently, Android has reached great success in mobile operating system especially in smartphones and tablets. Due to these circumstances, Android developers are introducing new applications daily to satisfy the needs of the Smartphone users. Android application developers are so much interested to interface hardware such as camera, sensors, compass, Bluetooth, and Wi-Fi … into their application.

To build image processing algorithms on a smartphone, several constrains are considered. Camera phone problems such as illumination, occlusion and pose variations are of great interest. Limiting factors such as reduced computing power, low quality cameras, and limited memory should also be discussed [3]. Although many efforts have been applied to develop robust face recognition algorithms, relatively few works have focused on adapting the technology to the domain of mobile computing. In some systems, mobile devices are used only to sense video and the processing tasks are performed on a server via a wireless network. More recent works show a trend to make all the computations in the mobile device [5].

In this work, Java software technology is chosen as an Android based programming language to reach a successful face recognition system. There are many reasons for using Java. we propose a simple, but robust and efficient face recognition algorithm for implementation in mobile devices. Java has significant advantages over other languages and environments that make it suitable for just about any programming task: Java is easy to learn, easy to use, easy to write, and easy to compile. Furthermore, Java is object-oriented, i.e. it allows user to create modular programs and reusable code. One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Face Recognition Techniques

Video signals constitute a rich source of visual information. They are made of a sequence of images, called frames, taken at regular time intervals (specified as the frame rate) and showing a scene in motion. With the advent of powerful smart phones, it is now possible to perform advanced visual analysis on video sequences, and sometime at rates close to, or even faster than, the actual video frame rate. Once the individual frames of a video sequence have been extracted, the different image processing functions can be applied to each of them. A Robust Face Recognition algorithm should satisfy the following criteria:

- Performs in different lighting conditions
- Supports low resolution image
- Provides training and testing results in real time

Video tracking process could be summarized: first feature points in an initial frame are detected. Second, tracking these points in the next frame is managed; this is a search for these points in this new frame. Obviously, since we are dealing with a video sequence, there is a good chance that the object on which the feature points are found has moved (the motion can also be due to camera motion). Therefore, you must search around a point's previous location in order to find its new location in the next frame. This is what accomplishes the cv::calcOpticalFlowPyrLK function. You input two consecutive frames and a vector of feature points in the first image, the function returns a vector of new point locations. To track points over a complete sequence, you repeat this process from frame to frame. Note that as you follow the points across the sequence, you will unavoidably lose track of some of them such that the number of tracked feature points will gradually reduces. Therefore, it could be a good idea to detect new features from time to time.

In [6], authors discussed a number of limitations on an eigenface approach system used for face recognition that used Princple Component Analysis (PCA). The face recognition results were acceptable however the proposed method could not be considered as a real system and was not robust for illumination variations. Other researchers [7] presented an illumination robust face recognition system comprising new PCA feature and a fusion approach integrating two half-face images for varied consumer applications. The algorithm could be summarized in three steps. First, face images were subdivided into two sub-images to minimize illumination effect. Second, matching scores were obtained from every sub-image, and finally combined using a weighted-summation operation to get a fused-score to identify an unknown user. The system was not extended to video signals. A. Wagner and A. Ganesh [8] proposed a system for recognizing human faces from images taken underneath practical conditions, i.e. with enough illumination variation. The system succeeded to solve this problem for still images.

System Model

The proposed framework starts by creating a video signal application in JAVA environment. The face recognition combines a hybrid skin color - eigen face algorithm for face detection and an interest points localization for feature extraction and face recognition. The whole system is implemented using Java to fit with Android OS and mobile platform. Figure (2) shows the block diagram of the proposed system.

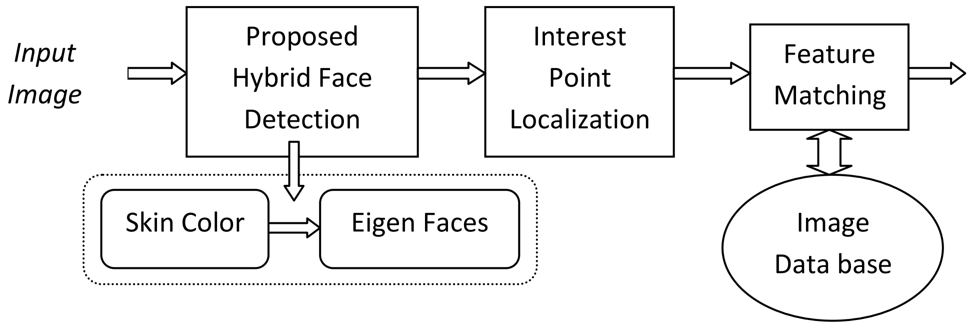


Figure (2): Block diagram of the proposed hybrid recognition system.

Building life camera application in JAVA environment

In our proposed mobile application, performing face detection needs a preprocessing sub-algorithm using JAVA code for building a camera application [9]. The general steps for creating this custom camera interface are as follows:

- Detecting and accessing camera hardware at run time and then accessing it -if present.
- Creating a camera Preview Class (it is a Surface View that can display the live image data coming from a camera, so users can frame and capture a picture or video
- Building a Preview Layout (it considers the container for the camera preview class that incorporates the preview and the user interface controls you want)
- Setup listeners for capture (connecting listeners for your interface controls to start video capture in response to user actions, such as pressing a button)
- Capturing and Saving Files (once the preview class and the view layout have been built, you are ready to capture video signal)
- The camera output signal must release properly for use by other applications, as it is shared on the whole mobile device

Proposed Hybrid Face Detection Algorithm

Skin color face detection suffers false detection due to backgrounds with color components near face color. Furthermore, multi-faces let most algorithms to be in trouble. The proposed hybrid face detection system uses a pre-filter to select skin color pixels' then eigenface technique similar to that proposed in [10] is applied on the pre-filtered image.

Someone could think that using dual face detection methods increases execution time and then moves towards offline algorithms. The experiments within this work proves that using this hybrid system makes eigen faces works properly and on less number of pixels which means less computation complexity. As a general view, this algorithm extracts the relevant information of an image and encodes it as efficiently as possible. Mathematically, the algorithm calculates the eigenvectors of the covariance matrix of the set of face images [11].

The eigen values of a square matrix are simple constant numbers that represent the whole matrix in an eigen vectors equation:

$$Ax = \lambda x \qquad (1),$$

where x is the eigen vector and λ is the eigen value.

For every eigen vector there is only one associated eigen value. In order to calculate the eigen vectors of a square matrix, we should start with calculating the eigen values as follows:

$$Ax - \lambda x = 0$$
$$(A - \lambda I)x = 0 \quad (2)$$
$$|A - \lambda I| = 0$$

We will start by calculating the eigen values from equation (2), then we will use equation (1) to obtain the related eigen vectors.

The eigen vectors corresponding to non-zero eigen values of the covariance matrix produce an orthonormal basis for the subspace image. The eigen vactors are sorted in descending order according to their corresponding eigen values. The eigen vector related to the greatest eigen value usually reflects the maximum variance within the video frame image. On the other hand, the one related to the smallest eigen value corresponds to the minimum variance [9].

Each image from the set contribute to an eigenvector, this vector characterizes 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 frames themselves but these feature weights.

Android supports a wide array of camera features you can control with your camera application, such as picture format, flash mode, focus settings, face detection and many more. To use the face detection feature in our JAVA camera application built previously, general steps are required:

- Start face detection support on your mobile device (as this feature are not supported on all devices)
- Create a face detection listener (in order to be notified of any respond to the detection of a face)
- Add the face detection listener to your camera object
- Start face detection after preview

PROPOSED FACE RECOGNITION MODEL

Interest Point Detection

Many computer vision tasks are based on the detection of interest points. Using such local features can improve the robustness of different processing algorithms. Algorithms of Interest Point Detection have been shown to be well suited for feature extraction. Furthermore interest point detection is used to reduce the data flow and consequently the computational costs, as all consecutive processing steps such as characterizing the points' neighborhood and matching them are solely applied to the interest points. There are two criteria for evaluating interest points: repeatability and information content. *Repeatability* compares the geometrical stability of the detected interest points between different images; *Information content* is a measure of the distinctiveness of an interest point. Numerous interest point detectors have been proposed in the literature. Harris Corners [12], SIFT Difference-of-Gaussian (DoG) [13] keypoints, Maximally Stable Extremal Regions (MSER) [14], Hessian Affine [15], FAST [16] and Hessianblobs [17] are some examples.

The algorithm used for interest point detection in our work uses a method extracted from Fast Hessian detector from the Speed up Robust Feature (SURF) [18] and applies Java programming through BoofCV. *BoofCV* is an open source Java library for real-time computer vision and robotics applications that has many classes to be formatted and programmed according to user's orientation. The algorithm used could be summarized as follows:

1- Calculate the integral image of the filtered face:

$$I_{\sum}(X) = \sum_{i=0}^{i\le x}\sum_{j=0}^{j\le x} I(i,j) \qquad (3)$$

2- Calculate Hessian Matrix for Image, to detect blob-like structure at locations where the determination is maximum:

$$\mathrm{H}(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \qquad (4)$$

Where $L_{mn}(x,\sigma)$ is the Gaussian second order derivative ($\frac{\partial^2 g(\sigma)}{\partial m \partial n}$) of the image I.

3- Scale spaces are usually implemented as an image pyramids, scale space is divided into octaves each octave represent a series of a filter response.

4- Localize the interest points in the images over scales.

5- Apply the algorithm over BoofCV library.

Feature Matching

Each interest point has a location, scale, and orientation associated with it. Interest point locations are needed in the geometric verification step to validate potential candidate matches. Furthermore image database (images representing faces) are also applied to the same interest points detection algorithm to extract interest points and corresponding features in an offline execution algorithm. For efficiently indexing all the local features of detected interest points, and assuming a small image database (50 faces), approximate nearest neighbor (ANN) search of SIFT descriptors with a best-bin-first strategy was applied. The features codebook is trained by k-means clustering. During a query, scoring the database images can be made fast by using an inverted file index associated with the codebook.

RESULTS AND DISCUSSION

Figures (3 and 4) show the face detection proposed algorithm and the corresponding interest points' detection method applied to two different images. The first contains two faces inserted from the web, while the second contains one face acquired from the laptop camera. Both images show good face localization. The second figure shows that the proposed method has succeeded to localize face despite bad illumination. Figure (5) shows several images acquired via mobile CAM and the result of face detection and recognition from the implemented database. To measure the robustness and effectiveness of the proposed method, two measures are considered: execution time and recognition error.

The data base (DB) used was organized as follows: 2 females and 13 males face images; 6, 8, and 10 different illumination conditions cases for each person was selected. This yields a DB of 90, 120, and 150 face images respectively.

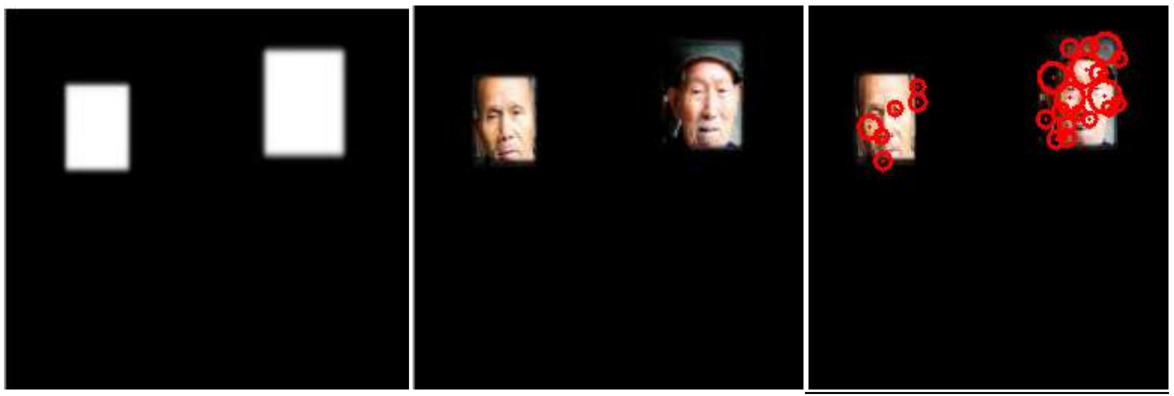


Figure (3): Image from the web: (a) Original image – (b) Face detection filter – (c) Face filtered – (d) Interest points detection

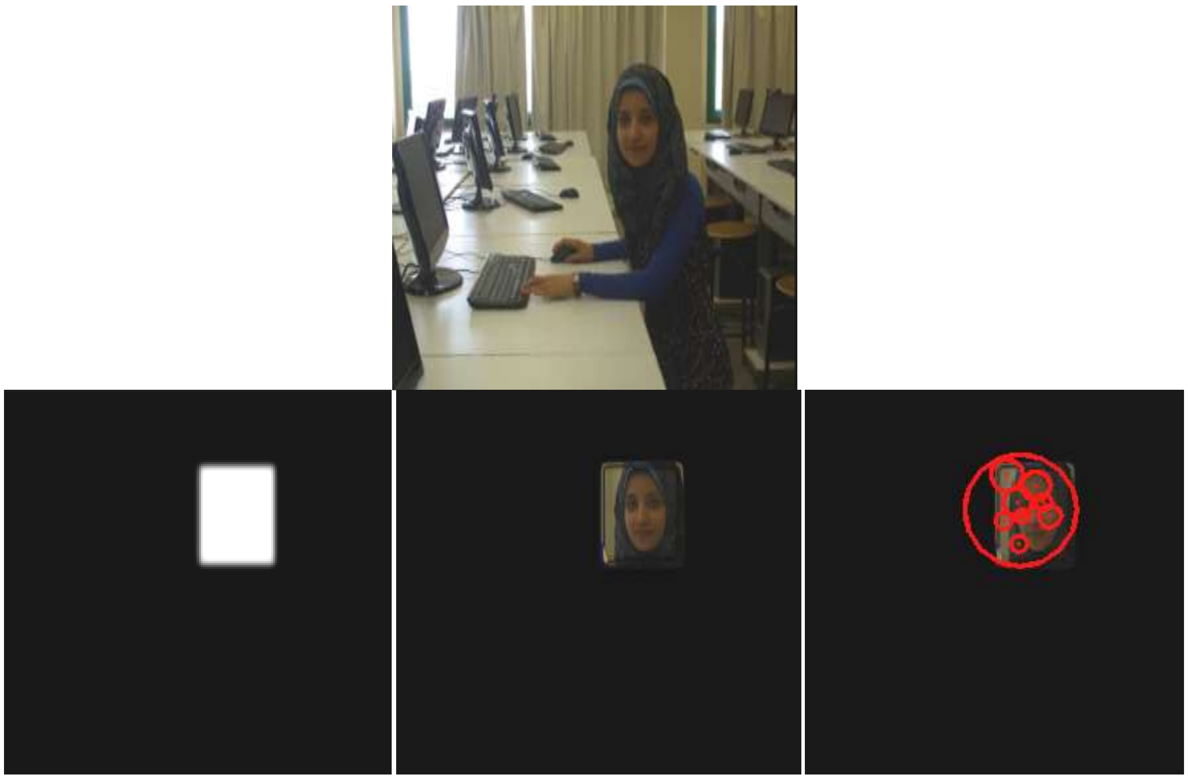
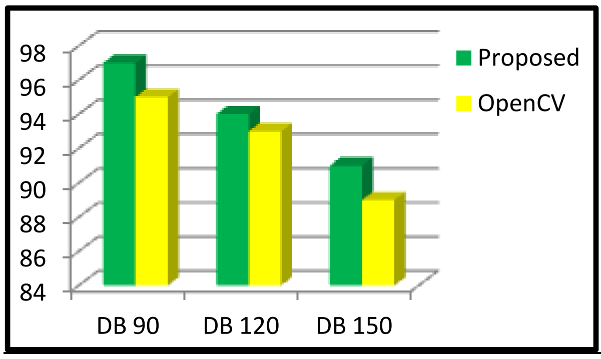
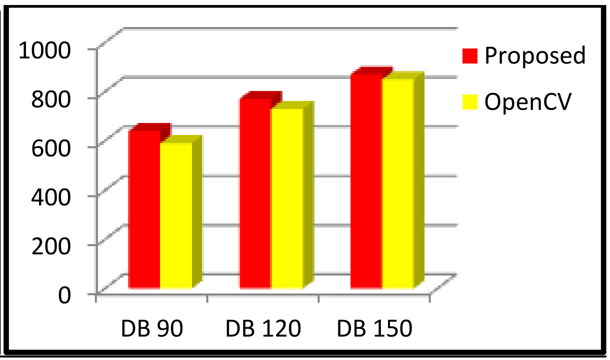


Figure (4): Image from mobile CAM: (a) Original image – (b) Face detection filter – (c) Face filtered – (d) Interest points detection

The OpenCV face recognizer [19] is an open source C++ code with many advantages. To start our comparison and maintain similar hardware configurations, our proposed Java code and the OpenCV code were both applied to the same H/W configurations. We have used a smart phone-virtual machine simulator to let the laptop behaves as a smart phone. Figure (6) shows the detection-recognition rate as a percentage and the detection-recognition time (measured in ms). From this figure, as the DB size increases the recognition rate decreases since the system enters more interest points and features matching solutions. Furthermore, as the DB increases the execution time also increases.

Figure (5): Recognition Results.





(a) Detection-Recognition Rate Comparison (b) Detection-Recognition Time in ms
Figure (6): Comparison between the proposed algorithm and the OpenCV [19].

The proposed method shows better recognition rate when compared with that in [19] but the execution time has been increased. We should notice that both algorithms have been implemented with different codes, Java and C++. The difference in the execution time is not remarkable (ms) since we achieved better recognition rates.

CONCLUSION

Illumination variations that happen on face images reduce the performance of face recognition systems in different environments. Face recognition is a challenging and important application for different environments. Obtained results show that the proposed hybrid face detection method that uses both skin color filter and eigen faces gives better results than using eigen faces only (as developed in the OpenCV). Recognition system that fits with Android smart phones uses interest points' detection for better features matching. The system shows effective detection-recognition rate when compared with existing open source algorithms in different DB sizes. Furthermore, the execution time is acceptable for real time applications.

REFERENCES

[1] Industry leaders announce open platform for mobile devices, November 2007.
[2] K. Owen, An Executive Summary of Research in Android & Integrated Development Environments, April 2011.
[3] OpenCV - Android Website, http://opencv.willowgarage.com/wiki/Android
[4] A. Holzer and J. Ondrus, "Mobile application market: A developer's perspective," Telematics and informatics, vol. 28, no. 1, pp. 22-31, Feb. 2011

[5] E. V. Fernandez, H. G. Pardo, D. G. Jimenez, L. P. Freire, "Built-in Face Recognition for Smart Photo Sharing in Mobile Devices," IEEE International Conference on Multimedia , July 2011.
[6] K. Kim, "Face Recognition using principal component analysis", USA, June 2000.
[7] S. H. Lee, D. J. Kim, J. H. Cho, "Illumination-Robust Face Recognition System Based on Differential Components," IEEE Transactions on Consumer Electronics, Vol. 58 (3), 2012.
[8] A. Wagner, A. Ganesh, "Toward a Practical Face Recognition System: Robust Alignment and Illumination by Sparse Representation," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 34 (2), 2012.
[9] "Introduction to java programming," Y.D. Liang, Prentice Hall, 2011.
[10] M. Easwaran, B. Poorna, "A Novel Framework for Face Recognition in Real Time Environments," International Journal of Scientific and Research, Vol. 3 (8), 2013.
[11] M.A. Turk, A.P. Pentland, "Face Recognition Using Eigenfaces". Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3-6 June 1991, Maui, Hawaii, USA, pp. 586-591.
[12] C. Harris and M. Stephens, "A combined corner and edge detector," in Proc. of 4th Alvey Vision Conference, 1988.
[13] D. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol. 60, no. 2, pp. 91 110, 2004.
[14] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in Proc. of British Machine Vision Conference (BMVC), Cardiff, Wales, UK, September 2002.
[15] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, "A Comparison of Affine Region Detectors," International Journal on Computer Vision, vol. 65, no. 1-2, pp. 43–72, 2005.
[16] E. Rosten and T. Drummond, "Machine Learning for High Speed Corner Detection," in Proc. of Euproean Conference on Computer Vision (ECCV), Graz, Austria, May 2006.
[17] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in Proc. of European Conference on Computer Vision (ECCV), Graz, Austria, May 2006.
[18] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "Speeded-up robust feature," Computer Vision and Image Understanding, vol. 110, no. 3, pp. 346–359, 2008.
[19] http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_api.html, FaceRecognizer API, OpenCV.