VOL. 10, NO. 23, DECEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

ENHANCED FACE RECOGNITION METHOD PERFORMANCE ON ANDROID vs WINDOWS PLATFORM

Mohammed Hayyan ALSIBA[1], Hadi Bin Manap[1] and Abdul Adam Bin Abdullah[2]
[1]Faculty of Engineering Technology, University Malaysia Pahang, Gambang, Kuantan, Malaysia
[2]Automotive Engineering Centre, University Malaysia Pahang, Pekan, Malaysia
E-Mail: mhdhayyan@gmail.com

ABSTRACT

Android is becoming one of the most popular operating systems on smartphones, tablet computers and similar mobile devices. With the quick development in mobile device specifications, it is worthy to think about mobile devices as current or - at least - near future replacement of personal computers. This paper presents an enhanced face recognition method. The method is tested on two different platforms using Windows and Android operating systems. This is done to evaluate the method and to compare the platforms. The platforms are compared according to two factors: development simplicity and performance. The target is evaluating the possibility of replacing personal computers using Windows operating system by mobile devices using Android operating system. Face recognition has been chosen because of the relatively high computing cost of image processing and pattern recognition applications comparing with other applications. The experiment results show acceptable performance of the method on Android platform.

Keywords: face recognition, OpenCV, android, image processing, pattern recognition.

INTRODUCTION

Face recognition studies are very popular among image processing and pattern recognition researchers. The wide range of applications include: biometrics, Information security, access control, law enforcement, smart cards and surveillance system [1]. One of the most important merits of using face is that it is as any individual's biological traits cannot be misplaced, forgotten, stolen or forged.

Face recognition is not easy if compared to other dissimilar object recognition. The difficulty occurs because faces appear to be roughly alike and the differences among them are quite difficult to be recognized using traditional pattern recognition techniques [2]. There are many factors that cause the appearance of the face to vary. Examples include: age, facial expression, facial paraphernalia, ethnicity and gender. Other factors are related to image environment, like night vision, uncontrolled illumination, pose, scale and imaging parameters (e.g., resolution, focus, imaging, noise, etc.) Studies showed that age variations, illumination variations and pose variations are three major problems plaguing current face recognition systems [2, 3, 4, and 5].

Applying face recognition algorithms in real-time is dependent on the algorithm itself and the device or platform to be applied on. Android is an operating system (OS) based on the Linux kernel and currently developed by Google for mobile devices. Android was unveiled in 2007. It is designed primarily for touchscreen mobile devices such as smartphones and tablet computers. It is also used in game consoles, digital cameras, regular PCs, and other electronics. According to surveys of 2015, Android has the largest installed base of all operating systems [6]. Android apps are usually developed in Java programming language using the Android software development kit (SDK). Android SDK includes a set of development tools, like debugger, libraries, a handset emulator, documentation, sample code, and tutorials. Until around the end of 2014, the officially supported IDE was Eclipse using the Android Development Tools (ADT) Plugin. At the beginning of 2015, "Android Studio" became the official IDE [7]. Actually, developers are free to use any text editor to edit Java and XML files, then use command line tools to create, build and debug Android applications. The necessary requirements are only Java Development Kit and Apache Ant. In this paper we use Eclipse with ADT Plugin.

OpenCV is an open source computer vision and machine learning software library. It provides C++, C, Python and Java interfaces and it is designed to be run in multiplatform: Linux, Mac, Android and Windows OS. OpenCV provides three methods of face recognition: Eigenfaces, Fisherfaces and Local Binary Patterns Histograms (LBPH). Eigenfaces and Fisherfaces find a mathematical description of the most dominant features of the training set as a whole.

In this paper we use an enhanced version of the LBPH algorithm which analyzes each face in the training set separately and independently. The LBPH algorithm, which is implemented by OpenCV, is combined with Histogram of Oriented Gradients (HOG) method to enhance the face recognition. The new method is tested on two different platforms for evaluation and comparison purposes. To compare the platforms we focus on two factors: development simplicity and performance. To evaluate development simplicity, the procedure needed by developers for applying the method on both platforms is compared. To evaluate performance, the recognition statistics and processing time for applying the method on both platforms is compared too.

LITRATURE REVIEW

Many researchers have worked on face recognition. Useful information can be found in review

papers like [3]. Results of most of the papers show that robust face recognition in uncontrolled illumination environment is still one of the unsolved challenges. There are two main approaches for pattern recognition in general: geometric feature-based descriptors and appearance-based descriptors. Geometric descriptors can be hard to extract reliably under variations in facial appearance and difficult lighting conditions. On the other hand, appearance-based descriptors tend to blur out small details. Most of the researchers use methods derived from Local Binary Patterns (LBP) or Edge Orientation Histogram (EOH) for Feature extraction [4, 5].

Very few researchers applied their algorithms on Android platforms using Android apps. Recently, the use of process virtual machine (VM) Dalvik, which was replaced later by Android Runtime (ART) [7], allows the Android products especially the industrial products to increase quickly. The VM allows developers to ignore the difference of hardware. This means that applications can be run on all devices using Android OS without any modification. Because of his improvement in the developing environment, Android platform is started to be used by researchers to build scientific applications. Consequently, many computer vision algorithms might be implemented on the android based devices. Face detection is a fundamental human-computer interaction technique to be applied. But researchers in some publications like [8], stated that "Due to the performance limit on the mobile devices, complex face detection process must be avoided in order to generate a fast detection". They proposed a method to enhance the detection rates and reduce the running time by applying ordered priority orientation algorithm.

Another problem is the power limitations that mobile devices exhibit. Researchers suggested cloud computing as a solution by making computations offline on the "cloud". Thus reducing the power consumption on the device and allowing more elaborate and accurate algorithms to be performed on the server [9].

The Authors in [8, 9] did not mention any information about the specification of the devices and systems used. It is expected that the devices used are not of high performance specifications.

In [10] the authors designed and implemented a real-time object recognition system based on Android mobile device to assist car drivers for safety driving. It depends on lane line detection and safety distances monitoring with the vehicles in front. The fps was 0.46 when they used HTC hero mobile phone. They mentioned that if their system is running on HTC Incredible mobile phone, the fps could achieve 2.06. HTC hero was released in 2009 and it has a 528 MHz ARM 11 CPU and 288 MB RAM [11]. The HTC Incredible was released in 2011 with 1 GHz Scorpion CPU and 768 MB RAM [11]. From the experimental results, they concluded that the application can be applied on the mobile phones easily and can work smoothly. Even though the application is not as complex and time costing as face recognition applications, but the mobile devices are improving very quickly and having increasing number of sensors and processing power.

In [12], authors implemented an improved algorithm of visual background extractor (Vibe) based on Android platform with OpenCV. Experiment results on Nexus7 device, showed that their algorithm is applicable in real-time with acceptable performance. Nexus7 has Quad-core 1.2 GHz Cortex-A9 CPU and 1 GB RAM [11]. The paper concluded that although mobile platform performance of motion detection still has a large disparity with personal computer, but with this work, the computer vision algorithm will transplant into the Android system more easily, and with the developing of mobile platform and higher performance provided in the future, the performance of computer vision algorithm on Android device will become as on computers. Their application is built with Java code and it calls algorithm written in C/C++ code to be compiled with Android NDK by using Java Native Interface (JNI).

FACE RECOGNITION METHODOLOGY

In this research we propose a new face recognition method using enhanced LBPH algorithm. OpenCV 2.4 Provides a newer FaceRecognizer class for face recognition. The currently available algorithms are Eigenfaces, Fisherfaces, and Local Binary Patterns Histograms LBPH. We built our method by modifying the OpenCV version of LBPH.

LBPH itself is a modified version of LBP which is one of the best performing texture descriptors and widely used in various applications. LBP was first described in 1994 [16, 17]. The algorithm considers the central pixel value of a 3 x 3 neighborhood as a threshold value. Then it labels the result of applying the threshold on the surrounding pixels as a binary number (0 or 1). Figure-1 shows an example of applying the algorithm. The binary numbers are then converted to a local value by converting it to a decimal number according to the weights in Figure-2. The histogram of the labels can then be used as a texture descriptor.

Ahonen T. and Pietikäinen M. [18] suggested to extend the basic histogram into a spatially enhanced histogram which encodes both the appearance and the spatial relations of (m) facial regions. The histogram is computed independently within each of the (m) regions generating (m) histograms. The spatially enhanced histogram gives a description of the face on three different levels of locality: The LBP labels for the histogram contain information about the patterns on a pixel-level, the labels are summed over a small region to produce information on a regional level, and the regional histograms are concatenated to build a global description of the face [18]. It is not necessary to have rectangular regions and not even necessary to have the same size or shape of each region. Human face can be considered as a composition of multi-patterns. LBP can be one of the best algorithms for face recognition if the facial image is divided into local regions and texture descriptors are extracted from each region independently.

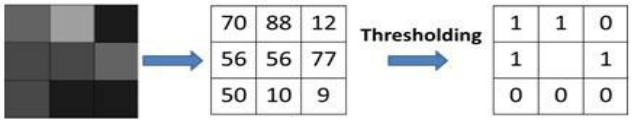


Figure-1. Example of applying LBP on a 3 x 3 neighborhood pixels.

1	2	4
8		16
32	64	128

Figure-2. Binary number multiplication weight.

Following the procedure of Heisele *et al*. [19] we can automatically detect each region in the image instead of detecting the face and then using a fixed division into regions. Some facial features, such as eyes, play more important roles in recognition. Therefore, regions can be weighted based on the importance of the information they contain. For example, the weighted Chi square distance can be defined as in Equation (1) [18]:

$$\chi_w^2(\mathbf{x}, \boldsymbol{\xi}) = \sum_{j,i} w_j \frac{(x_{i,j} - \xi_{i,j})^2}{x_{i,j} + \xi_{i,j}}, \quad (1)$$

where x and ξ are the normalized enhanced histograms to be compared, indices i and j refer to i^{th} bin in the histogram corresponding to the j^{th} local region and ω_j is the weight for region j.

In this paper we suggest adding Histogram of Oriented Gradients (HOG) descriptors [20] to extend the histograms. HOG descriptors showed a significant enhancement for other applications like in [21]. HOG features are calculated by Equation (2):

$$\forall x, y : g_x(x, y) = I(x+1, y) - I(x-1, y)$$
$$\forall x, y : g_y(x, y) = I(x, y+1) - I(x, y-1) \quad (2)$$

where $g_x(x, y)$ and $g_y(x, y)$ denotes the x and y components of the image gradient, respectively.

The magnitude $m(x, y)$ and orientation $\theta(x, y)$ of the image gradient are calculated by Equation (3):

$$m(x, y) = \sqrt{g_x(x, y)^2 + g_y(x, y)^2}$$
$$\theta(x, y) = \tan^{-1}(g_y(x, y) / g_x(x, y)) \quad (3)$$

Unsigned orientation of the image gradient suggested by Dalal *et al*. [20] is used as in Equation (4):

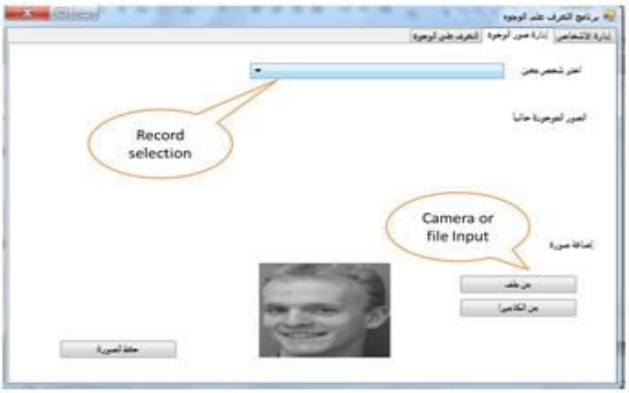
$$\breve{\theta}(x, y) = \begin{cases} \theta(x, y) + \pi & \text{if } \theta(x, y) < 0 \\ \theta(x, y) & \text{otherwise} \end{cases} \quad (4)$$

PLATFORM SPECIFICATIONS

To compare the development simplicity and performance we tested our enhanced LPBH algorithm on two different platforms:

- Windows7 OS and visual studio IDE platform: We applied the program on HP – probook with Intel core i5 CPU and 4GB RAM. Figure-3 (a), (b) and (c) show the GUI interface of the application. The application provides three interfacing forms. The first one is for building the database information of faces to be recognized. The second form is for assigning face images with the saved database records. The third form is the testing form where the application matches an input face image with the images in the database to recognize the person. The program allows two ways for inputting face images: a camera or a stored image on the hard disk.

(a)



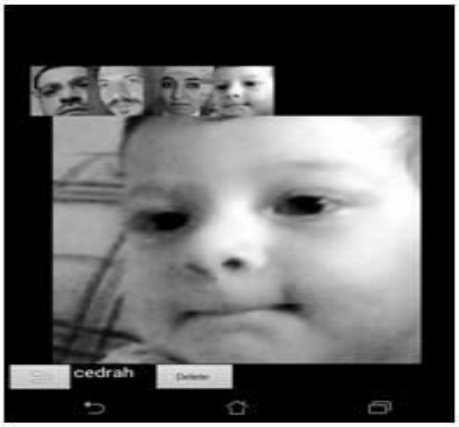
(b)

(c)

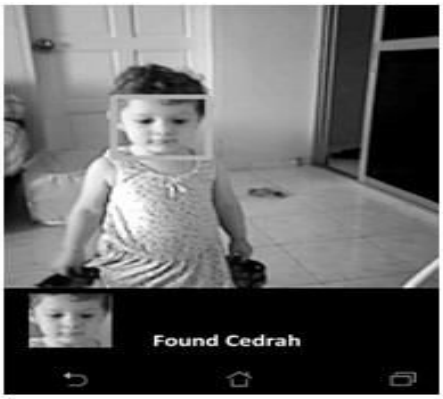
Figure-3. The GUI interface of the application developed for Windows 7 OS using visual studio IDE platform.

- Android 4.4.2 OS and Eclipse IDE with Android SDK platform: We applied the program on Asus fonepad 7 with Intel Atom Z2520 CPU and 2GB RAM. Figure-4 (a), (b) and (c) show the GUI interface of the Android app. The app provides three interfacing screens. The first one is for database building, where user can capture a face image and assign a name to it. The second is to show the database of faces and their assigned names. The third screen is for testing, where the application matches an input face image with the images in the database to recognize the person.

(a)



(b)



(c)

Figure-4. The GUI interface of the android app developed for Android 4.4.2 OS using eclipse IDE platform.

DEVELOPMENT SIMPLICITY COMPARISON

In this section we compare the procedure needed by developers to build a program for applying the method on both platforms.

Using OpenCV library on Windows OS

OpenCV has more than 2500 algorithms. The algorithms can be used for many applications like face detection, objects recognition, tracking, 3D models of objects extracting and eye movement following. The easiest way to use OpenCV under Windows OS is to install it from the official website [13]. The website provides a full documentation on how to install and use OpenCV in different platforms [14]. To use the OpenCV library there are two options [15]: Installation by Using the Pre-built Libraries and Installation by Making Own Libraries from the Source Files. We used the first option because it is easier to complete. After having the OpenCV directory that contains the OpenCV header files plus binaries, environment variables should be added to the systems path. The OpenCV libraries are distributed on the Microsoft Windows OS in a Dynamic Linked Libraries (DLL). Therefore, the content of the library are loaded at runtime. Another advantage is that many programs can use the same library file at the same time.

In this paper Visual Studio Integrated Development Environment (IDE) is used. To build an application with OpenCV two things should be done: Show to the compiler the header files and tell the linker from where to get the functions or data structures of OpenCV, when they are needed. The linker in Windows OS needs to know where is the DLL to search for the data structure or function at the runtime. This information is stored inside ".lib" files which are import libraries. To pass on all this information to the Visual Studio IDE we can do it globally to allow all projects to get this information. This is done in global property sheet, which is automatically added to every created project, by adding the include directories using the environment variable OPENCV_DIR [14].

In case of using Eclipse, only the CDT plugin for C/C++ is needed. Then it is important to tell OpenCV where the OpenCV headers and libraries are. This is done in Project properties by adding the path to the folder where OpenCV was installed to "Include paths(-l)" in GCC "C++ Compiler" / "Includes", and the path to where the openCV libraries reside to "Library search path (-L)" in "GCC C++ Linker". Then in "Libraries(-l)" we add the needed OpenCV libraries [14].

In general, configuring OpenCV on Windows, Mac or Linux OS using any C++ IDE is not a complicated task. This is because OpenCV is written in C++ and its primary interface is in C++.

Using OpenCV library as a native android module

As mentioned previously, OpenCV is written natively in C++. But since version 2.4.4 it started to supports Java. Furthermore, it has a template interface which works smoothly with Standard Template Library (STL) containers. The Android way is writing all code in Java [7]. Even though OpenCV started to supports Java, it is not enough and it is needed to go to a native level and write part of the application in C/C++. Furthermore, sometimes it is useful to reuse a functionality which is written in C++ and uses OpenCV without having to rewrite the C++ code in Java.

To develop an application with OpenCV on Android OS using JNI, the following software should be installed:

1. JDK (Java Development Kit) and JRE (Java SE Runtime Environment).
2. Android SDK (Software Development Kit) and the following components: Android SDK Tools, revision 20 or newer and SDK Platform Android 3.0 (API 11). These components are downloaded using Android SDK Manager.
3. Eclipse IDE.
4. Native Development Kit (NDK).
5. Android Development Tools (ADT) and C/C++ Development Tooling (CDT) plugins for Eclipse.
6. OpenCV4Android Software Development Kit (SDK) which enables development of Android applications with use of OpenCV library.
7. It is good to have Android Virtual Device (AVD) which is an emulator configuration for modelling actual devices. It allows developers to define hardware and software options to be emulated by the Android Emulator. AVD can be created using AVD Manager.
8. OpenCV Manager to be installed on the device that will use the OpenCV application. OpenCV Manager is an Android service for managing OpenCV library binaries on end users devices.

CONCLUSIONS

As we can see the application in Android OS using OpenCv is a mixed-programming which involves C/C++ and Java. The native code is written with JNI and Android NDK is used to compile it. This makes the configuration of OpenCV on Android OS more complicated comparing with other OSs. The detailed configuration steps can be found on: http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/android_dev_intro.html (Last visited on August, 2015).

PERFORMANCE COMPARISON

Experiment details

Same recognition algorithm is applied on both platforms. The captured images are processed as following: Firstly the image is tested to detect a face. Face detection is done automatically using Paul Viola and Michael Jones Haar feature-based cascade classifiers [22]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. The algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. OpenCV comes with a trainer as well as detector [14]. Moreover, OpenCV already contains many pre-trained classifiers for face, eyes and smile. The pre-training data is provided as XML files which are stored in opencv/data/haarcascades/ folder. We only need to load the required XML classifiers. Then we load our input image (or video) in grayscale mode. The function "detectMultiScale()" is then used to find a face. If faces are found, the function returns the positions of detected faces as "Rect(x,y,w,h)". Once these locations are obtained, a Region Of Interest (ROI) for the face is created. Eye detection can be applied on this ROI if needed.

After detecting a face in the captured image, the face is extracted to a new cropped image. The extracted face image is then normalized to 120 x 120 pixels. The enhanced LBPH algorithm described in section 3 is then applied to classify the input face to the closest face in the database which was built by the user. In both platforms, we tested the algorithm on different 4 people with various gender and age as shown in Figure-4 (b). Testing experiment was done in real-time by capturing 50 different inputs for the faces randomly and in random illumination situations.

Results

Table-1 shows the result of applying the original LBPH method implemented by OpenCV. Table-2 shows the result of applying the proposed enhanced LBPH method.

Table-1. Recognition results on both platforms using normal LBPH.

Platform	Windows	Android
Testing data	50	50
Faces detected	48	47
Recognized correctly	39	37
Misclassified	9	10
Face detection average time	300 ms	330 ms
Complete recognition average time	400 ms	440 ms
Precision	81.2%	78.7%
Recall	95.1%	92.5%
F-measure	87.6%	85%

Table-2. Recognition results on both platforms using enhanced LBPH.

Platform	Windows	Android
Testing data	50	50
Faces detected	48	47
Recognized correctly	42	40
Misclassified	6	7
Face detection average time	300 ms	330 ms
Complete recognition average time	420 ms	470 ms
Precision	87.5%	85.1%
Recall	95.4%	93%
F-measure	91.3%	88.8%

By comparing the results in Table-1 and Table-2, we can see that the enhanced algorithm has improved the number of the correctly recognized faces from 39 to 42 on Windows platform and from 37 to 40 on Android platform. Precision, Recall and F-measure values are improved for both platforms. The number of the detected faces is not changing because there is no modification in detecting algorithm.

In both tables a comparison between results for both platforms is presented. As we can see, the experiment results show that the performance of the method on Android system is reaching acceptable levels. Precision, Recall and F-measure values are very close on both platforms. Moreover, there is no big difference in processing time although the specification of the windows device is better than the specification of the android device.

CONCLUSION AND FUTURE WORK

An enhanced LPBH algorithm was proposed in this paper. The algorithm was tested on two different platforms using Windows and Android OS. This was done to evaluate the method and to compare the platforms. The platforms were compared according to development simplicity and performance.

The application in android using OpenCv is a mixed-programming which involves C/C++ and Java. The native code is written with JNI and Android NDK is used to compile it. This makes the configuration of OpenCV on Android OS more complicated comparing with other OSs.

As for performance, according to the experiment results, the effectiveness of the method based on Android system was verified. Actually, image processing and pattern recognition applications performance on smartphone platforms are still slower than the performance on personal computers. But with the quick development of mobile platform, it is expected that higher performance devices will be designed in the near future. It is expected that the performance of computer vision algorithms on Android OS devices will become as efficient as on personal computers. Therefore, it is worthy to think about smartphones devices as current or - at least - near future replacement of personal computers.

Future work includes improvement of classification methods to improve the recognition rate and reduce the processing time. Actually, with the current recognition rate on both platforms, the algorithm is still cannot be considered as robust algorithm for critical applications like security applications for example.

REFERENCES

[1] Divyarajsinh N. P. and Brijesh B. M. 2013. Face Recognition Methods & Applications. International Journal of Computer Technology & Applications. Vol. 4, No. 1, pp. 84-86.

[2] Jafri R. and Arabnia H. R. 2009. A Survey of Face Recognition Techniques. Journal of Information Processing Systems. Vol. 5, No. 2, pp. 41-68.

[3] Zhao W., Chellappa R., Phillips P.J. and Rosenfeld A. 2003. Face recognition: A literature survey. ACM computing surveys Vol. 34, No. 4, pp. 399–485.

[4] Valstar M. F., Jiang B., Méhu M., Pantic M. and Scherer K. 2011. The first facial expression recognition and analysis challenge. In: IEEE International Conference of Automatic Face & Gesture Recognition (FG 2011). Canada. pp.921 - 926.

[5] Cheng J., Deng Y., Meng H. and Wang Z. 2013. A facial expression based continuous emotional state monitoring system with GPU acceleration. In: IEEE International Conference of Automatic Face and Gesture Recognition (FG 2013). China. p-6.

[6] Manjoo F. 2015. A Murky Road Ahead for Android, Despite Market Dominance. The New York Times. Available online on: (last visited August, 2015) http://www.nytimes.com/2015/05/28/technology/personaltech/a-murky-road-ahead-for-android-despite-market-dominance.html

[7] Android M Developer Preview: Online: http://developer.android.com/ Last visited on August, 2015.

[8] Guanghui M., Weiliang M., Shibiao X., and Xiaopeng Z. 2013. Rotational Invariant Face Detection On a Mobile Device. In International Conference on Virtual Reality and Visualization. China. pp. 229 - 232.

[9] Ferzli R. and Khalife I. 2011. Mobile cloud computing educational tool for image/video processing algorithms. In: Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE). USA. pp. 529 – 533.

[10] Chang K. and Wang K. 2012. Design and Implementation of Traffic Safety Guardian System for Android Based on OpenCV. In: IEEE International Conference on Connected Vehicles and Expo. China. pp. 288 - 289.

[11] GSMArena: online: http://www.gsmarena.com/ Last visited on August, 2015.

[12] Wenkai W. *et al.* 2013, An Improved Method of Vibe for Motion Detection Based on Android System. In: IEEE International Conference on Robotics and Biomimetics (ROBIO). China. pp. 2436 – 2440.

[13] OpenCV: online: http://opencv.org/ Last visited on August, 2015.

[14] OpenCV 2.4.11.0 documentation: online: http://docs.opencv.org/ Last visited on August, 2015.

[15] Bradski G. and Kaehler A. 2008. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly. USA.

[16] Ojala T., Pietikäinen M. and Harwood D. 1994. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In the 12th IAPR International Conference on Pattern Recognition (ICPR 1994). pp. 582 - 585.

[17] Ojala T., Pietikäinen M. and Harwood D. 1996. A Comparative Study of Texture Measures with Classification Based on Feature Distributions. Pattern Recognition. Vol. 29, pp. 51-59.

[18] Ahonen T. and Pietikäinen M. 2006. Face Description with Local Binary Patterns: Application to Face Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 28, No. 12, pp. 2037 - 2041.

[19] Heisele B., Ho P., Wu J., and Poggio T. 2003. Face Recognition: Component-Based versus Global Approaches. Compter Vision and Image Understanding. Vol. 91, No. 1-2, pp. 6-21.

[20] Dalal N. and Triggs B. 2005. Histograms of oriented gradients for human detection. In International Conference on Computer Vision & Pattern Recognition CVPR2005. San Diego, USA. pp. 886–893.

[21] Alsibai M. H. and Hirai Y. 2012. Recognition of Blue Traffic Signs Enhanced By Aggregating Fragmentarily Segmented Sign Images. In 13th IASTED Int. Conf. on Signal and Image Processing. USA pp. 27- 34.

[22] Viola P. and Jones M. 2001. Rapid object detection using a boosted cascade of simple features. In the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001. Vol. 1. pp. I-511 - I-518.