EVALUATION OF FACE RECOGNITION ALGORITHMS ON EMBEDDED SYSTEMS

by

Ahmet ÖZDİL

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Melikşah University

in partial fulfillment of the requirements for the degree of

Master of Science

In

Electrical and Computer Engineering

January 2015
Kayseri, Turkey

This is to certify that I have read the thesis titled “Evaluation of Face Recognition Algorithms on Embedded Systems” by Ahmet ÖZDİL and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Electrical and Computer Engineering, the Graduate Institute of Science and Engineering, Melikşah University.

Jan 13, 2015 — Asst. Prof. Metin Mete ÖZBİLEN
Supervisor

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Jan 13, 2015 — Prof. Dr. Murat UZAM
Head of Department

Examining Committee Members

Title and Name		Approved
Asst. Prof. Metin Mete ÖZBİLEN	Jan 13, 2015	
Asst. Prof. Kadir Aşkın PEKER	Jan 13, 2015	
Asst. Prof.. Rıfat KURBAN	Jan 13, 2015	

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Science and Engineering.

Doç Dr. Ahmet UYAR
Substitute Director

January 2015.

EVALUATION OF FACE RECOGNITION ALGORITHMS ON EMBEDDED SYSTEMS

Ahmet ÖZDİL

M.S. Thesis - Electrical and Computer Engineering
January 2015

Supervisor: Yrd. Doç. Dr. Metin Mete ÖZBİLEN

Co-Supervisor: Yrd. Doç. Dr. Kadir Aşkın PEKER

ABSTRACT

The identification of a face from a video or image source is a study of computer vision known as face detection and recognition. By the development of computing power, face detection and recognition has become popular in recent years. Face recognition has a wide range of applications including: biometrics, content-based image retrieval systems, photography and video processing.

Computing and visual sensing technologies in today's world has reached to a state that is inexpensive, reliable and accurate. Embedded systems are used in daily life much more than that we know. Home appliances, furniture, buildings, cars, etc. all of them are served with an embedded system or in some cases more than one. Since embedded systems are small and can be easily applicable to nearly everywhere, adding face recognition capabilities makes them more convenient. Many embedded systems(which have about 1GHz or mode clock rate and at least 256MB of capacity) integrated with CMOS camera can be used in face detection systems. Due to the heavy computing process, more developed versions–of embedded systems should be used in face recognition.

In this study, the performance of embedded systems on face recognition is explored. Two different ARM-based embedded systems and an x86 based system are compared in different performance metrics. A video file is used as image source. Firstly, face detection is implemented on all of three systems and their timing performances are measured. Three different face recognition algorithms: Fisherfaces, Eigenface, and Local Binary Pattern Histograms (LBPH) are studied. The timing performances and accuracies of algorithms are compared in order to find out how the embedded systems perform face recognition. How they perform face recognition and how to improve their performance are also studied.

Keywords: Face Recognition, Face Detection, Fisherfaces, Eigenfaces, LBPH, Embedded System, Pandaboard, BeagleBone Black

GÖMÜLÜ SİSTEMLERDE YÜZ TANIMA ALGORİTMALARI İNCELEMESİ

Ahmet ÖZDİL

Yüksek Lisans Tezi – Elektrik ve Bilgisayar Mühendisliği
Ocak 2015

Tez Yöneticisi: Yrd.Doç.Dr. Metin Mete ÖZBİLEN

Ortak Tez Yöneticisi: Yrd.Doç.Dr. Kadir Aşkın PEKER

ÖZ

Bir video veya resim kaynağından alınan resimlerin tanımlanması, yüz bulma ve yüz tanıma olarak bilinen bilgisayarla görü çalışmalarıdır. Son yıllarda bilgisayarların hesaplama gücünün artmasıyla yüz bulma ve tanıma daha popüler hale gelmekte. Bu durum göz önünde bulundurulduğunda, yüz tanıma birçok kolaylık getirmekte. Çok geniş uygulama alanları vardır: biyometri, içerik tabanlı görüntü alma sistemleri, fotoğrafçılık ve video işleme

Günümüz dünyasında bilgisayarla hesaplama ve görsel algılama teknolojileri öyle bir seviyeye ulaştı ki, ucuz, güvenilir ve kesin çözümler mümkün olmakta. Gömülü sistemler günlük yaşamda bilinenden daha fazla kullanılmakta. Beyaz eşya, hatta mobilyalar, binalar, arabalar vb. hepsi bir gömülü sistemler servis vermektedir, hatta bazı durumlarda birden fazla gömülü system bulunabilmektedir. Gömülü sistemler küçük ve neredeyse her yere kolaylıkla uygulanabilir olduğu için, yüz tanıma becerileri eklenmesi onların daha kullanışlı olmasını sağlar. Birçok gömülü sistem yüz bulmada kullanılabilecek CMOS kamera ile arz edilmektedir ve ağır hesaplamalardan dolayı, yüz tanıma için daha gelişmiş versiyonları kullanılabilir.

Bu çalışmada, gömülü sistemlerin yüz tanımada nasıl performans gösterdikleri araştırıldı. İki farklı arm-tabanlı gömülü sistem ve bir tane intel-tabanlı sistem performans ölçümleri ile karşılaştırıldı. Bir video dosyası resim kaynağı olarak kullanıldı. İlk olarak bu üç sistemde yüz bulma çalışması uygulandı ve zaman performansları ölçüldü. Sonra üç farklı yüz tanıma algoritması, Fisherfaces, Eigenfaces ve Local Binary Pattern Histograms(LBPH), seçildi. Videodan alınmış resimlerden bir veri tabanı oluşturuldu ve oluşturulan veri tabanı üzerinde seçilen üç algoritma, daha önce belirtilen üç sistem üzerinde çalıştırıldı. Zamanlama performansı ve algoritmaların doğruluk oranları karşılaştırıldı ve yüz tanıma söz konusu olunca gömülü sistemlerin nasıl davrandığı belirlenmeye çalışıldı. Nasıl bir performans sergiledikleri ve performansın nasıl artırılabileceği araştırıldı.

Anahtar Kelimeler: Yüz Tanıma, Yüz Bulma, Fisherfaces, Eigenfaces, LBPH, Gömülü Sistem, Pandaboard, BeagleBone Black

To my beloved family

ACKNOWLEDGEMENT

This project is granted by The Ministry of Science, Industry and Technology Turkey, with grant number 0370.STZ.2013-2.

I express sincere appreciation to Asst. Prof. Metin Mete ÖZBİLEN and Asst. Prof. Kadir Aşkın PEKER for their guidance and insight throughout the research.

I express my thanks and appreciation to my family for their understanding, motivation, and patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYSMBOLS AND ABBREVIATIONS

SYMBOL/ABBREVIATION	
API	Application Programming Interface
ARM	Acorn RISC Machine
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
DDR2	Double Data Rate Type 2
DDR3	Double Data Rate Type 3
DDR3L	Double Data Rate Type 3 Low Power
DRAM	Dynamic Random Access Memory
DSI	Display Serial Interface
DVI-D	Digital Visual Interface Only Digital
EDR	Enhanced Data Rate
eMMC	Embedded Multimedia Card
EMTs	Emergency Medical Technicians
ELBP	Extended Local Binary Patterns
ER	Emergency Room
FAT32	File Allocation Table 32 Bits
GB	Giga Byte
GCC	Gnome C Compiler
GHz	Giga Hertz
GPIO	General Purpose Input Output
HD	High Definition

HDMI	High Definition Multimedia Interface
IDE	Integrated Development Environment
JTAG	Joint Test Action Group
LBP	Local Binary Patterns
LBPH	Local Binary Pattern Histograms
LCD	Liquid-Crystal Dispaly
LDA	Linear Discriminant Analysis
MB	Mega Byte
MMC	Multimedia Card
OMAP	Open Multimedia Applications Platform
OpenCV	Open Computer Vision
OpenGL	Open Graphics Library
OpenVG	Open Vector Graphics
OS	Operating System
PC	Personal Computer
PCA	Principal Component Analysis
PIC	Peripheral Interface Controller
PRU	Programmable Runtime Unit
RGB	Red Green Blue
RJ45	Registered Jack 45
SD	Secure Digital
SSH	Secure Shell
TV	Television
UART	Universal Asynchronous Receiver Transmitter
U.S.	United States
USB	Universal Serial Bus

CHAPTER 1

INTRODUCTION

Face recognition is one of the most important and hardest problems in computer vision. In recent years, it is becoming more and more important; a certain level of success is reached but under free lighting and exposure conditions face recognition is still an open research area for development.

Applications of face recognition are used or planned to be used in daily life from homeland security to healthcare. Some examples of face recognition already applied to security field are as follows:

The German Federal Police use a face recognition system to allow voluntary subscribers to pass fully automated border controls at Frankfurt Rhein-Main international airport to ease the passing of people [1]. U.S. Department of State operates one of the largest face recognition systems in the world with over 75 million photographs that is actively used for visa processing [1]. Mexican government employed face recognition software to prevent voter fraud in the 2000 presidential election [1]. Face recognition can find many application subjects in security field, because its identification is very handy.

Passwords are the most important elements of the computer security, yet they can be easily forgotten. Instead of password, using face recognition makes the security of computers more convenient. Face recognition systems are also incorporated into unlocking mobile devices. The Mobile OS android market is working on face recognition and integrating it into their cell phones. They have created an application called VisidonApplock. This application allows you to put a face recognition lock on any of your applications which increases the safety of private apps and contents [1].

Another field that face recognition can be utilized is to organize digital images. Photographing is very easy due to today’s large capacity digital cameras. They can easily produce large number of images with large sizes. Therefore organizing digital images is a hard work. Google's Picasa digital image organizer has a built in face recognition system starting from version 3.5 and onwards. It can associate faces with persons, so that queries can be run on pictures to return all pictures with a specific group of people together. Picasaweb.com has also been providing a similar feature to its users [1]. Sony's Picture Motion Browser (PMB) analyses photo, associates photos with identical faces so that they can be tagged consequently, and distinguishes between photos from one person to many and nobody [1]. Apple's iPhoto image organizer has a feature named Faces which allows users to associate names to faces in a series of photographs. The software suggests other matching faces and locates it in the photo library and progressively refines its recognition, according to the user's acceptance, rejection or renaming of the suggested face [1].

Face can tell many things about health condition. Face recognition comes with the opportunity for EMTs and ER doctors to access related information instantly, even if the patient is unconscious. Medical records such as pharmaceutical allergies, family contact information and other accessible patient data records that can save an emergency responder time and thus save a patient’s life [2].

1.1 FACE RECOGNITION

Face authentication, face recognition, and face clustering are different from one another, but they are all called as part of face recognition problems. All of these problems try to measure the similarity between two images. Especially in security field, a certain level of success is gained, but there is more to do.

Face recognition process starts by face detection, which is to find face areas in an image. There are different approaches to face detection depending on elements such as:

shape, texture, and color. The most popular approach is Viola-Jones detector, which uses boosting technique on rectangular attributes. This detector is the first choice in many applications because of its speed and success, which uses only grey scale images. Viola-Jones detector has good results on front looking faces, but its success decreases when face turned more than a certain degree. [3]

Color based face detection techniques are not affected much from turned face.?[4] In color based detection techniques, skin color models or classifiers which are trained by the samples including different races, skin colors and lightening conditions are being developed, after pre-detection usually invalid detections are eliminated according to blob’s shape and size.

The second step in face recognition process is usually to detect facial features such as eyes, mouth, and nose. Scaling and aligning according to key points on face, such as eyes and mouth, are important for accurate and precise comparison of faces.

There are many different approaches to face recognition. Traditional methods use two dimensional (2D), 24x24 pixels sized examples, but new face recognition methods use 3D or high definition (HD) approaches.[5]

Feature transformations are utilized commonly in traditional face recognition methods. Grey-scale image is considered as the feature vector and certain feature transformations (PCA, LDA, etc.) are applied to the grey-scale image. One of the traditional methods is Eigenfaces. Eigen-faces or eigen-vectors are obtained with applying PCA (Principal Component Analysis) to a set of face images. These eigen-face images constitute a basis vector set for face dataset. These faces, which are expressed as the basis vector type, are matched by calculating the similitude-distances, using known distance methods.[6] There is another method named Fisherfaces, where LDA (Linear Discriminant Analysis) is used as feature transformation and processes are similar to Eigenfaces.[7] Also, different recognition methods are developed, which use facial features such as eyes, nose and mouth separately[8], to be more independent from pose.

These methods in [8][26][27] increase complexity and affect the robustness and stability of system negatively. When Viola-Jones detection algorithm is extended to detect turned faces, the error rate increases dramatically.

1.2 EMBEDDED SYSTEMS

Computer vision algorithms need capacity for both processing and huge storage for implementing on computers. It is urgent to optimize the computer vision algorithms for both processing and storage to fulfill the real time requirements[9]. Since face detection is a computationally resource and time consuming task, an embedded solution would let different kinds of cheap devices to be used in many applications. [10].

Nowadays, embedded system processors are developed faster than PC processor, even they can accomplish many PC based tasks. Todays embedded systems are generally equipped with processors and peripherals based on PIC and ARM architecture Especially with the development of smart phones, embedded systems are equipped with a display processor which has multiple cores for high performance. All of these improvements boosted embedded system popularity and lead it to be used in many different application areas. For example face recognition in smart phones is really popular. It has been worked on application level security. Smart phones that have front camera let this task to be easily deployed. [11]

CHAPTER 2
FACE RECOGNITION ALGORITHMS

2.1 EIGENFACES ALGORITHM

Dimensionality reduction is important in image processing, due to big image data. Image data consists of at least 2-dimensional matrix in computer science. Principal Component Analysis (PCA) transform is used to reduce dimension in eigenfaces algorithm. For example; if a face image is represented in g-dimensional space, PCA transform aims to get an h-dimensional subspace, which answers maximum variance in the g-dimensional space and where h is too small according to g[12].

Eigenfaces algorithm consists of two steps; first training and second recognition. In training step, a set of subspace images is calculated, and in second step, a new image is projected to subspace and compared with the training sets, to find the closest face set to assign[6].

2.1.1 Principal Component Analysis (PCA)

Principal Component Analysis is used to reduce the components, and select principal ones. So PCA tries to find the appropriate dimension to reduce dataset. In multidimensional data, PCA can find a linear relation. PCA steps:

- Find 1st dimension which the data varies most.
- Find 2nd dimension which the data varies most and is perpendicular to 1st dimension.
- Continue finding the dimensions which the data varies most and is perpendicular to all previous dimensions.

These dimensions are called as Principal Components. They cause the data deconstruct into eigenvectors and eigenvalues. Eigenvector is the direction of the component. Eigenvalue is the number that tells the variance of that eigenvector.

Dimensionality reduction is performed by projecting the data points on the eigenvector as shown in Figure 1.

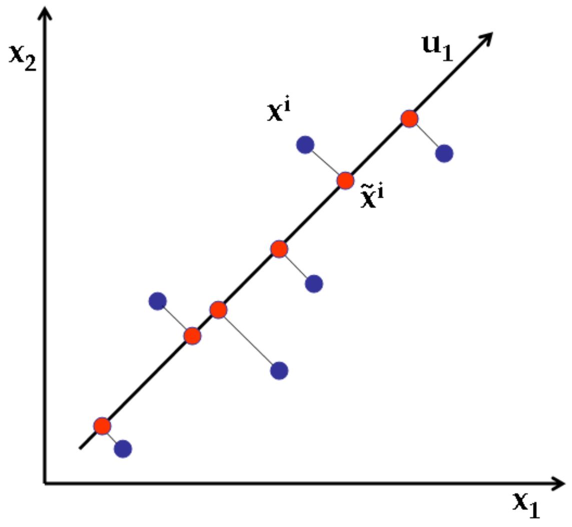


Figure 1 PCA Projection

2.1.2 Calculating Eigenfaces

A grayscale image I(x,y) is N by N 2-dimensional array can be used as N^2 vector. This means that the image is represented in N^2-space with a point.[6]

Working on this huge dimensional space is not feasible. Image can be represented in lower dimensional space. This can be done with PCA as described above. PCA determines the vectors that represent the image in the subspace. This subspace is called as “face space”. N by N images represented by N^2 length vectors, and these vectors are linear combination of original images. These vectors are the eigenvectors forming the covariance matrix of original face images. They are called as “Eigenfaces” since they bear resemblance to human faces as it can be seen in Figure 2[6].

Mean centered images are calculated by subtracting the normalized training images from the calculated mean image. If W is the matrix of mean centered training images W_i (i = 1, 2, ... L) and L is the number of training images, covariance matrix D is calculated from W as in Equation 2.1[12].

$$D = WW^T \quad (2.1)$$

To reduce the size of covariance matrix D, $D = W^TW$ can be used instead. Eigenvectors e_i and their corresponding eigenvalues λ_i are obtained from covariance matrix D.

$$z_i = E^T w_i \ (i = 1,2,\dots,L) \quad (2.2)$$

In the Equation 2.2, E is the matrix containing the eigenvectors and their largest eigenvalues, z_i represents the new feature vector of the new lower dimensional subspace [12].

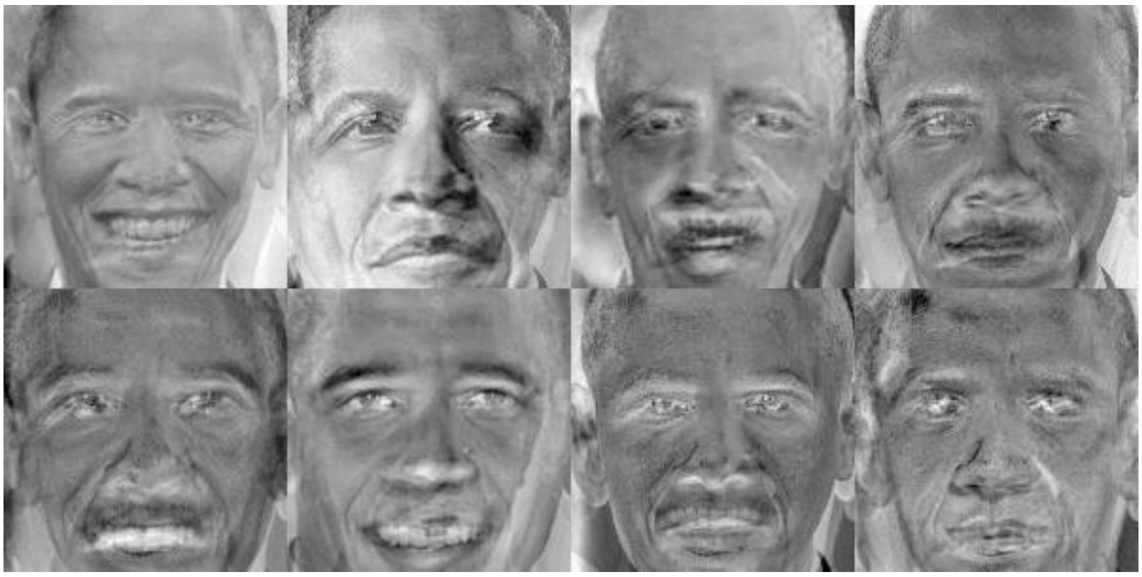


Figure 2 Eigenfaces Examples [17]

There is a negative aspect of this method, that it tries to maximize inter class scattering and also intra class scattering. Inter class scattering is good for classification while intra class scattering is not. In face recognition, if there is a variance of illumination, this increases intra class scattering very high, even classes seems stained, and causes low classification[13]. If there is illumination variance, performance decreases, and classes may smear together.

2.2 FISHERFACES ALGORITHM

When it comes to illumination variance, inter class difference comes forth for success rate. In different lighting conditions, classes smear together they cannot be separated. Linear Discriminant Analysis (LDA) or fisherfaces aims to increase inter class differences, not data representation. This means that LDA calculates a scatter for classifying the data more accurate. Therefore, class discrimination feature vectors are calculated as follows.[12]

$$S_w = \sum_{j=1}^{R} \sum_{i=1}^{M_j} (x_i^j - \mu_j)(x_i^j - \mu_j)^T \quad (2.3)$$

$$S_b = \sum_{j=1}^{R} (\mu_j - \mu)(\mu_j - \mu)^T \quad (2.4)$$

The formulas above are intra class (Equation 2.3) and inter class (Equation 2.4) scatter matrices respectively. The indices, *j* is class and *i* is image number. μ_j is the mean of class *j*, and μ is mean of all classes. M_j shows the number images in class *j*, and *R* is the number of classes. S_b is maximized while S_w is minimized for the classification to be done [12].

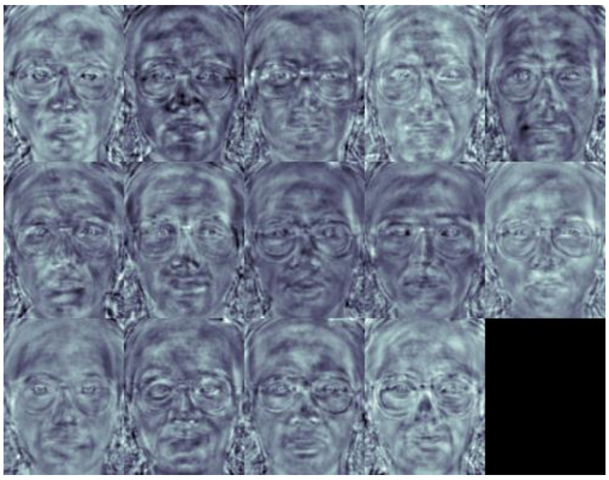


Figure 3 Fisherfaces Examples [16]

In Figure 3, fisherfaces of a dataset is shown.

2.3 LOCAL BINARY PATTERNS (LBP)

Local Binary Patterns (LBP) was first presented by Ojala et al. in [14] to use in texture description. The basic method, labels each pixel with decimal values called LBPs or LBP codes, to describe the local structure around pixel. As illustrated in Figure 4, value of the center pixel is subtracted from the 8 neighbor pixels' values, if the result is negative the binary value is 0, otherwise 1. The calculation starts from the pixel at the top left corner of the 8-neighborhood and continues in clockwise direction. After calculating with all neighbors, an eight digit binary value is produced. When this binary value is converted to decimal, the LBP code of the pixel is generated, and placed to the coordinates of pixel in matrix [15].

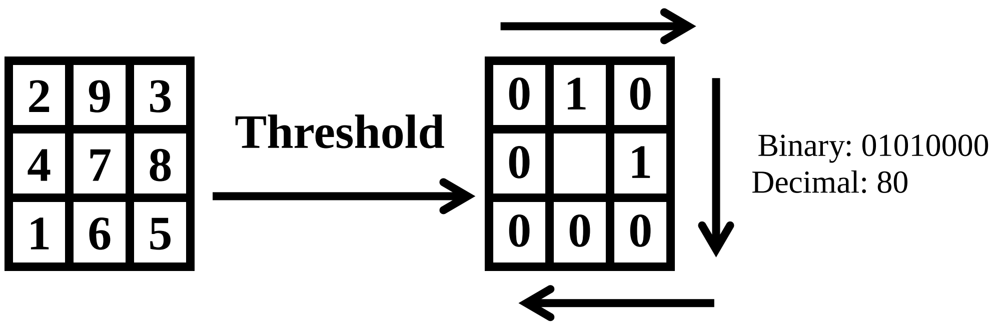


Figure 4 Basic LBP Operator

In Figure 4, basic algorithm of LBP is presented. There is a drawback of LBP which uses 8-neighborhood (3x3) that cannot cover large-scale structures. To take into account texture of different size structures, the method is generalized. In [18] Ojala et al. revised the method to be flexible for any radius and any number of sampling points and named the new method as Extended LBP(ELBP). Figure 5 shows different examples of ELBP operator, and P represents the number of neighbors and R represents the radius of a circle on which neighbors are located.

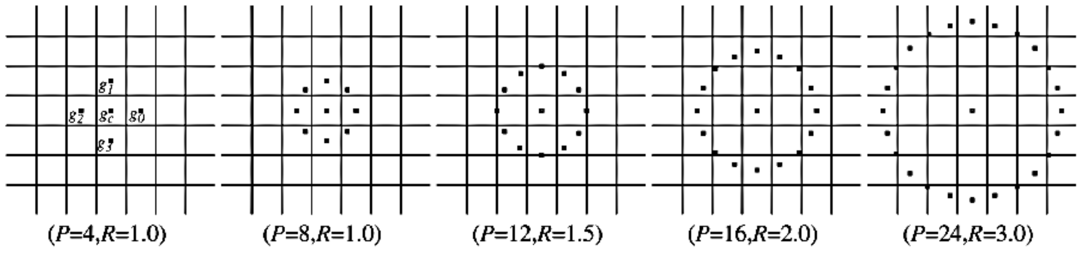


Figure 5 ELBP Operator Examples [18]

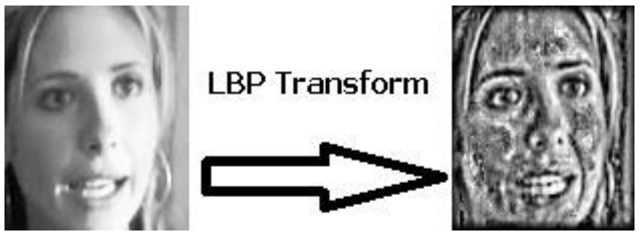


Figure 6 LBP Transform of a Face Image

The histograms of LBP are used for face recognition since LBP histograms contain information about the distribution of local micro patterns. Because the face image is too big for LBP calculation, dividing the image into small regions is proposed in [19]. Some parts of face (like eyes, mouth) contain more information for face recognition. Yang et al. proposes to train and allocate different weights for face parts, by their information covering and then concatenating them end to end to build up global description of face. This helps to collect local pattern information with spatial details of the whole image.

To decide whether two face images are belong to the same person, the images' histograms are compared. By default nearest neighbor approach is used to find the right class of the new image..

CHAPTER 3
EMBEDDED SYSTEM

3.1 INTRODUCTION

Embedded systems have different architecture than PCs and limited resources such as storage, processing capacity, etc. Thus, software developing for embedded systems has its characteristic difficulties. Today’s developing embedded systems has PC environment capabilities, but every embedded systems has its own software environment, constraints, and supported libraries. Therefore embedded software development itself becomes a research and development area. Embedded systems can vary much on memory, i/o units, hardware properties such as device interfaces, and related software components. Developing on embedded system, learning these properties and applying the software is a problem. It is a difficult problem to realize complex algorithms on embedded systems such as image processing and face recognition.

Today, Google presents the most commonly used embedded development environment with Android platform. Android platform has a rich application interface, but there are some performance constraints due to multivariate hardware platform support. Linux platforms can fully utilize the hardware if they are specially compiled for a specific hardware. Commonly used embedded systems have more than one compiled Linux systems. A development tool “gc++” is used for code development in Linux environment. Android platform uses Java as a development environment and Google provides more than one development tool. While developing code on embedded systems, usually the code is developed on PC and then copied to embedded system with an interface. Also Google provides a restricted emulator that works on PC. Therefore, there is a need for powerful PC system that will run embedded system emulator with the development environment of embedded systems for PC.

3.2 BEAGLEBONE BLACK

BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists.[20] The Beaglebone Black has some differences from the regular version by providing an onboard micro HDMI port, 512MB of DDR3L DRAM, 4GB onboard flash memory, an AM3358 processor at 1GHz, and making JTAG optional with a user supplied header. The features are listed below:

1. Processor: AM335x 1GHz ARM® Cortex-A8
2. 512MB DDR3 RAM
3. 4GB 8-bit eMMC on-board flash storage
4. 3D graphics accelerator
5. NEON floating-point accelerator
6. 2x PRU 32-bit microcontrollers
7. Connectivity
    a. USB client for power & communications
    b. USB host
    c. Ethernet
    d. HDMI
    e. 2x 46 pin headers
8. Software Compatibility
    a. Debian
    b. Android
    c. Ubuntu
    d. Cloud9 IDE on Node.js w/ BoneScript library

In Figure 7, BagleBone Black is presented. It can be seen in Figure 7 that it is credit card sized(8.6x4.8 cm). This makes it useful and easy to implement for many projects.[21] Necessary programs(like OpenCV, gcc…) are pre-installed on BeagleBone Black. This make it easy to implement code on BeagleBone Black.

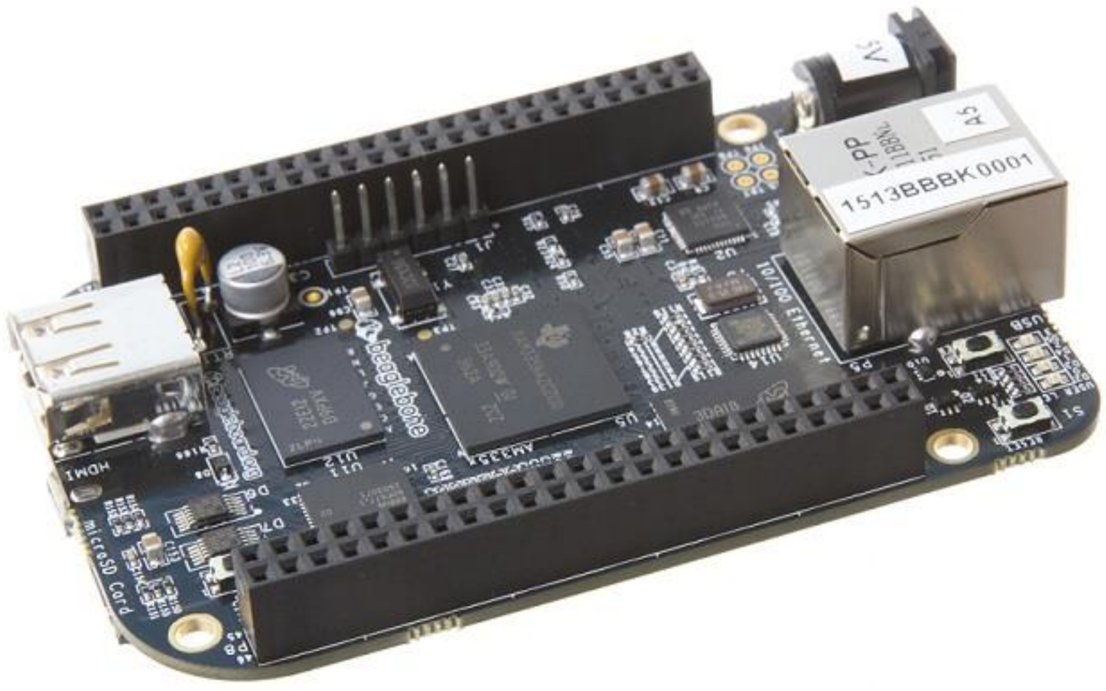


Figure 7 BeagleBone Black

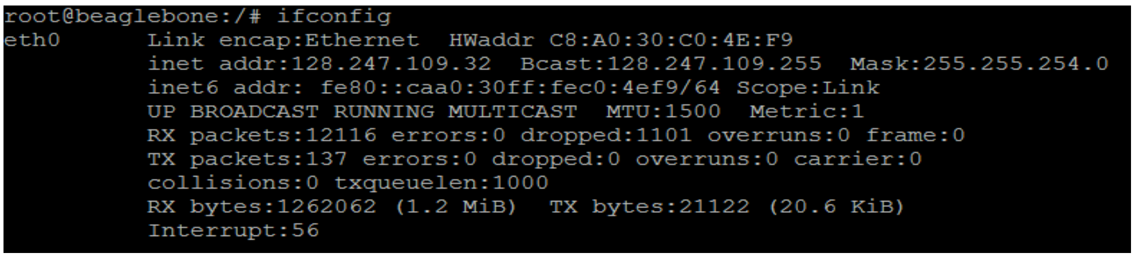
3.2.1 Connecting BeagleBone Black

Since g++ compiler and OpenCV library are pre-installed on BeagleBone Black, only thing should be known is how to connect to BeagleBone Black.

The board can be used as a PC, when a mouse and a keyboard is connected from USB port and a monitor is connected from mini-HDMI port. After attaching all devices, BeagleBone Black can be powered on and if the board recognizes the devices, it boots its built-in OS. This is rather not a preferred way since BeagleBone Black’s capacity is limited. Running an operating system with a graphical interface is very resource consuming task.

Instead of running a graphical operating system, just connecting with a terminal to BeagleBone Black over a medium is an easier way. BeagleBone Black lets three different connection as [21]:

1. Serial connection:
    a. Putty is installed from http://sourceforge.net/projects/putty.mirror/
    b. BeagleBone Black is connected from serial port to PC over USB interface.
    c. In Putty’s “Connection Type” menu, “Serial” type is selected and the COM Port number of connected BeagleBone Black is entered
    d. On the left menu, under “Connection”, in “Serial” tab below settings entered:
        - Speed(baud): 115200
        - Data Bits: 8
        - Stop bits: 1
        - Parity: None
        - Flow Control None
2. SSH using an Ethernet cable:
    a. Connecting to BeagleBone Black with SSH over Ethernet needs to know BeagleBone Black’s IP address. There are different ways to learn and assign a static IP address of BeagleBone Black; accessing the board over serial connection, or booting the graphical OS on the board, and assigning a static IP address by router.
    b. To assign a static IP address to BeagleBone Black these steps should be followed:
        - In terminal execute “ifconfig” code, the output of the code as follows:



```
root@beaglebone:/# ifconfig
eth0      Link encap:Ethernet  HWaddr C8:A0:30:C0:4E:F9
          inet addr:128.247.109.32  Bcast:128.247.109.255  Mask:255.255.254.0
          inet6 addr: fe80::caa0:30ff:fec0:4ef9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12116 errors:0 dropped:1101 overruns:0 frame:0
          TX packets:137 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1262062 (1.2 MiB)  TX bytes:21122 (20.6 KiB)
          Interrupt:56
```

Figure 8 IP Configuration of BeagleBone Black

- Execute “cat /etc/network/interfaces” code to see if static IP or DHCP configuration enabled. “iface eth0 inet dhcp” line means DHCP service is enabled. “iface eth0 inet static” line means static IP is enabled.

- Static IP address can be assigned by editing “/etc/network” file. The lines below should be added to “network” file. The address and netmask can be found with “ifconfig” command at the first step.

```
iface eth0 inet static
address xxx.xxx.xxx.xx
netmask xxx.xxx.xxx.x
```

After rebooting BeagleBone Black these setting should take effect. Then BeagleBone Black can be accessible over Ethernet.

3. SSH using USB cable:
   a. This is the easiest way to connect BeagleBone Black. The only requirement is to connect board to PC with a USB cable. Then BeagleBone Black powers on automatically. After power on process is completed, the board is accessible with SSH from “192.168.7.2” IP address. “root” user name with no password is used to login SSH.
   b. After SSH terminal is ready, on BeagleBone Black, the only thing to do is to develop the code. The code can be written on BeagleBone Black from terminal by using programs like vi, vim, etc. An easier way is to prepare the code on PC and compile it on BeagleBone Black.

Compiling code on BeagleBone Black can be done with the following code:

```
$g++ -o hello_opencv hello_opencv.cpp `pkg-config --cflags --libs opencv`
```

In the code above, a c++ source file named as “hello_opencv.cpp” is being compiled by g++ compiler with using “opencv” library.

3.3 PANDABOARD

Pandaboard is an Open OMAP™ 4 mobile software development platform.[22] The PandaBoard is a low-power, low-cost single-board computer development platform based on the Texas Instruments OMAP4430 system on a chip (SoC). The PandaBoard ES is a newer version based on the OMAP4460 SoC, with the CPU and GPU running at higher clock rates. The Pandaboard ES is used in this thesis as it has more processing capacity and flexibility.

3.3.1 Pandaborad ES Technical Specifications

Processing:

1. Core Logic OMAP4460 applications processor. Dual-core ARM® Cortex™-A9 MPCore™ with Symmetric Multiprocessing (SMP) at 1.2 GHz each. Allows for 150% performance increase over previous ARM Cortex-A8 cores.
2. Full HD (1080p) multi-standard video encode/decode
3. Imagination Technologies' POWERVR™ SGX540 graphics core supporting all major API's including OpenGL® ES v2.0, OpenGL ES v1.1, OpenVG v1.1 and EGL v1.3 and delivering 2x sustained performance compared to the previous SGX530 core
4. Low power audio

Display:

1. HDMI v1.3 Connector (Type A) to drive HD displays
2. DVI-D Connector (can drive a 2nd display, simultaneous display; requires HDMI to DVI-D adapter)
3. LCD expansion header
4. DSI Support

Memory:

1. 1 GB low power DDR2 RAM
2. Full size SD/MMC card cage with support for High-Speed & High-Capacity SD cards

Audio:

1. 3.5" Audio in/out
2. HDMI Audio out

3. Stereo audio input support

Connectivity:

1. Onboard 10/100 Ethernet
2. 802.11 b/g/n (based on WiLink™ 6.0)
3. Bluetooth® v2.1 + EDR (based on WiLink™ 6.0)

Expansion:

1. 1x USB 2.0 High-Speed On-the-go port
2. 2x USB 2.0 High-Speed host ports
3. General purpose expansion header (I2C, GPMC, USB, MMC, DSS, ETM)
4. Camera expansion header
5. LCD signal expansion using a single set of resistor banks

Debug:

1. JTAG
2. UART/RS-232
3. 2 status LEDs (configurable)
4. 1 GPIO Button
5. Sysboot switch available on board

Dimensions:

1. Height: 4.5" (114.3 mm)
2. Width: 4.0" (101.6 mm)
3. Weight: 2.88 oz (81.5 grams)

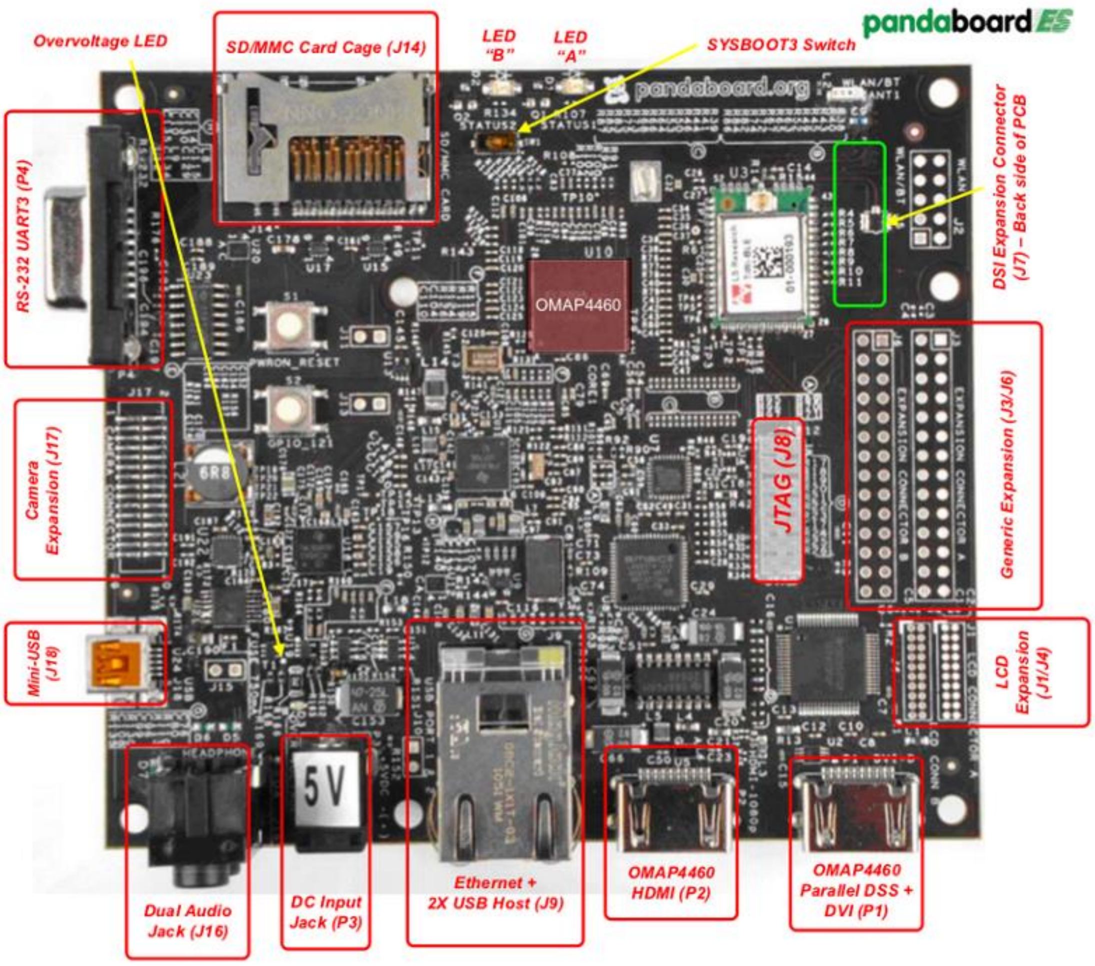


Figure 9 Pandaboard [22]

3.3.2 Installing Linux Pandaboard ES

Pandaboard is shipped without a working OS on it. So a special kernel compilation is needed. Currently available compilations are not working properly on Pandaboard ES. That's because there are some driver changes in new version and no pre-compiled Linux kernels contains that changes. This gives a rise to a new kernel compilation. Compiling a new kernel is very complex and open to any mistakes that can fail compilation. Two different kernel compilations are merged by working parts, to discard compilation troubles. Ubuntu 12.04 is installed on a microSD card and boot folder of installation is changed with manufacturer's working one.

The basic steps to install pre-built Ubuntu for Pandaboard is as follows[23]:

1. Ubuntu 12.04 Precise Pangolin Ubuntu Core is downloaded from: http://cdimage.ubuntu.com/ubuntu-core/releases/12.04/release/
2. The downloaded image is written on an empty SD(microSD) card. To write the image to the SD card, the card must be un-mounted like /dev/sde. Then this code is executed:

   ```
   zcat ./ubuntu-12.04-preinstalled-desktop-armhf+omap4.img.gz |sudo dd bs=4M of=/dev/sde ; sudo sync
   ```
3. In this installation, there are some problems with “boot” folder. The folder is replaced with manufacturers working one. This helped overcome incompatibility problems.

3.3.3 Installing OpenCV on Pandaboard ES

Installing OpenCV library on Pandaboard ES as follows[24]:

Install arm-linux-gcc cross compiler.

“arm-linux-gcc-4.3.2.tgz” file is downloaded from: http://www.friendlyarm.net/, and extracted into “/usr/local/arm/4.3.2/…”

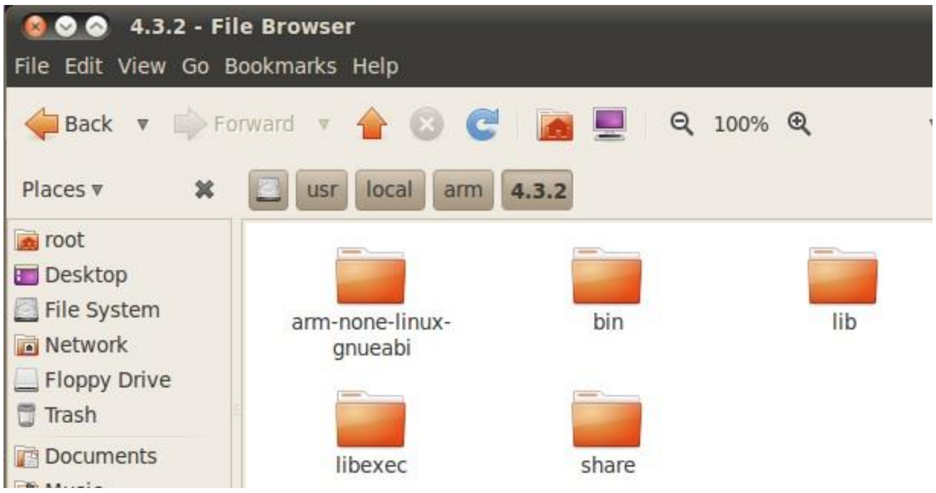


Figure 10 OpenCV Installation 1

In terminal the following code executed to add the install directory of the arm-linux-gcc to system bash so that the arm-linux-gcc/arm-linux-g++ command can be recognized:

```
# vi /etc/bash.bashrc
```

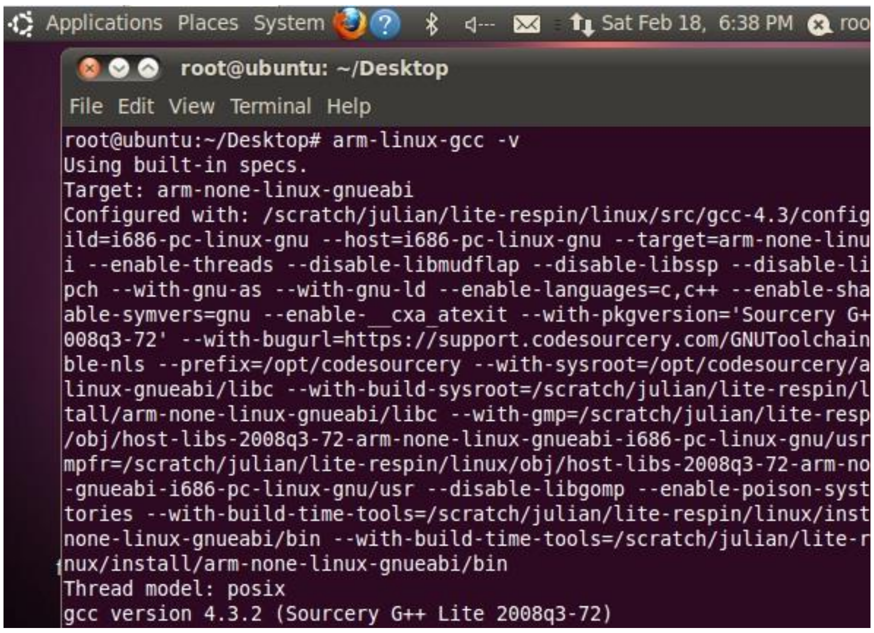
This code is added to the last line of bashrc file:

```
export PATH=$PATH:/usr/local/arm/4.3.2/bin/
```

The arm-linux-gcc is tested if included in the default path, in the terminal:

```
# arm-linux-gcc -v
```

The result of the above code is shown in Figure 11, which explains the compiler is ready for use:



```
root@ubuntu:~/Desktop# arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/gcc-4.3/config
ild=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linu
i --enable-threads --disable-libmudflap --disable-libssp --disable-li
pch --with-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-sha
able-symvers=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G+
008q3-72' --with-bugurl=https://support.codesourcery.com/GNUToolchain
ble-nls --prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/a
linux-gnueabi/libc --with-build-sysroot=/scratch/julian/lite-respin/l
tall/arm-none-linux-gnueabi/libc --with-gmp=/scratch/julian/lite-resp
/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr
mpfr=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-no
-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-syst
tories --with-build-time-tools=/scratch/julian/lite-respin/linux/inst
none-linux-gnueabi/bin --with-build-time-tools=/scratch/julian/lite-r
nux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
```

Figure 11 OpenCV Installation 2

Obtaining OpenCV source code

Current version of OpenCV library at the time of writing is a version 2.0.0. OpenCV source code can be downloaded with web browser from OpenCV-SourceForge.net(http://sourceforge.net/projects/opencvlibrary/), or wget command to acquire a source code directly on the command line:

```
#wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/opencv-2.4.9.zip
# tar xvjf opencv-2.4.9.zip
# cd root/Desktop/OpenCV-2.4.9/
~/Desktop/OpenCV-2.4.9# mkdir output
```

Compilation and installation of OpenCV binaries

```
~/Desktop/OpenCV-2.4.9#export PKG_CONFIG_PATH=/ usr/local/arm/4.3.2/arm-none-linux gnueabi/libc/armv4t/usr/lib/pkgconfig
~/Desktop/OpenCV-2.4.9#./configure
--prefix=/root/Desktop/armOpenCV/output --host=arm-linux CPPFLAGS=-I/root/Desktop/ OpenCV-2.4.9/output/include LDFLAGS=-L/root/Desktop/ OpenCV-2.4.9/output/lib --with-python=no --with-ffmpeg=no --with-imageio=no --with-gtk=no --with-carbon=no --with-unicap=no --with-gthread=no
~/Desktop/OpenCV-2.4.9#make –j3
~/Desktop/OpenCV-2.4.9#make install
```

4 folders namely **bin, include, lib, share** are created in **/root/Desktop/OpenCV-2.4.9/output**

Copying OpenCV library above to target:

```
mkdir /usr/include
mkdir /usr/lib
```

Copy all the files from /root/Desktop/OpenCV-2.4.9/output/include/opencv of host to /usr/include of target and /root/Desktop/OpenCV-2.4.9/output /lib of host to /usr/lib of target.

This last step makes OpenCV ready to use.

3.3.4 Using Pandaboard ES

Pandaboard ES is more powerful than BeagleBone Black. So it can be used with a graphical operating system, which is called desktop mode. Recommended setup for desktop mode of Pandaboard is as follows:

1. 5V power supply
2. SD card 4GB or more
3. USB to Serial adapter or Serial cable
4. Ubuntu (10.04 or above) host PC is recommended but any Linux debian-based system can work
5. RJ45 Ethernet cable
6. USB cable (Mini-A to Type-A USB)
7. DVI (HDMI Type-A to DVI-D) or HDMI (Type-A to Type-A)
8. USB keyboard

9. USB mouse
10. Display with HDMI or DVI support

After all setup is done, Pandaboard is ready to be used. There is a point to consider which is heat. The board should be placed as it can run without facing high temperatures. If some cooling mechanism is installed, Pandaboard can have a longer life.

CHAPTER 4
PC ENVIRONMENT

4.1 INTRODUCTION

Developing code on an embedded system is not an easy task due to its limited capacity of processing and memory and of course the heating problem. So that a PC system is used for developing the necessary code for face recognition. After a bug-free code is composed, it is moved to the embedded systems for compiling and testing.

The development environment in PC is composed with OpenCV library and Microsoft Visual C++ 2010 Express.

Microsoft Visual C++ 2010 Express has a standard installation procedure, but OpenCV must be identified by system.

4.2 INSTALLING OpenCV ON WINDOWS

OpenCV can be downloaded from the link in [25](the latest version 2.4.9). There are two different ways of installing OpenCV on windows system. The first way is to build the library from source files. The second and easier way is to download and install pre-built library. If no problem occurs while using pre-built library, this way should be selected. If some problems occur while installing from pre-built library, the first way should be selected, for a healthy installation.

After installing the OpenCV library on windows system, two settings should be done. First step is to add OpenCV to system path and second is to configure Microsoft Visual C++.

For the first setting, two environment variables should be set and OpenCV path should be added to the system path. To do this in “Environment Variables” editor, under “System Variables” table, click on the “New” button and set name as “OPENCV_DIR” and value as “C:\opencv\build\x86\vc10” and click “OK”. The second variable will be the version of OpenCV with the same procedure as previous. The name of variable is “OPENCV_VER” and its value is “249”. Then add the code below to the end of the “Path” variable and click “OK”:

```
;%OPENCV_DIR%
```

In a Microsoft Visual C++ project, some settings should be done to show how the OpenCV library will be used to the IDE. There are two ways of doing this, global and local. The global one takes effect to all projects, thus it is not recommended especially if multiple projects are being developed in same IDE. The local way steps are described below.

Every project has its own rule packet to build separately from others. In the rule packets every information is stored for IDE to build the project. There is two build modes, “Release” and “Debug”. These two modes are different from each other by their characteristics. The debug mode is more comprehensive since it enables to find bugs inside code. The release mode is more strict to make the application faster and smaller on disk. That’s why these modes have different rule packets. Rule packets named as project properties and can be modified in “Property Manager” Microsoft Visual Studio. Once a rule packet is generated, it can be used for same kind of projects.

In “Property Manager” a new project property sheet is added by right clicking to “Debug” and selecting “Add New Project Property Sheet” and it is named as “OpenCV_Debug”, as shown in Figure 12.

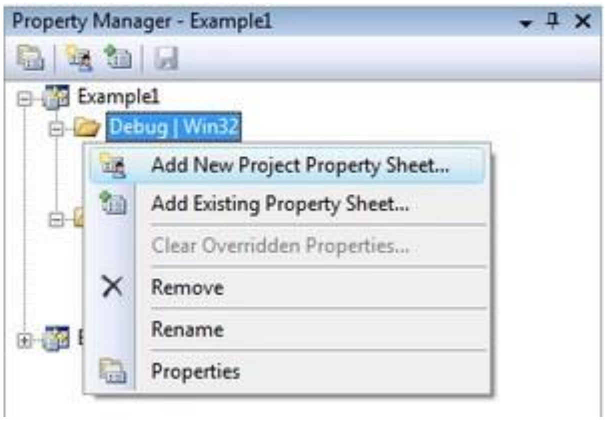
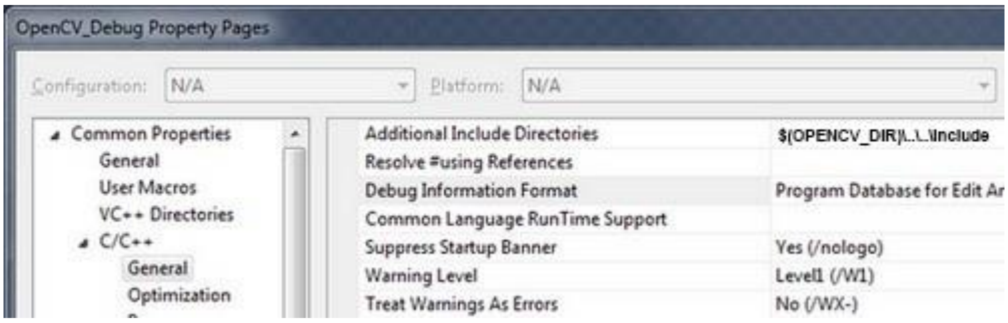

Property Manager - Example1
Example1
Debug | Win32
Add New Project Property Sheet...
Add Existing Property Sheet...
Clear Overridden Properties...
Remove
Rename
Properties

Figure 12 Adding New Property Sheet

The properties menu of “OpenCV_Debug” property sheet is opened and in “C/C++” group’s “General” tab, in the “Additional Include Directories” line “$(OPENCV_DIR)\..\..\include” is added, as shown in Figure 13 below.



OpenCV_Debug Property Pages

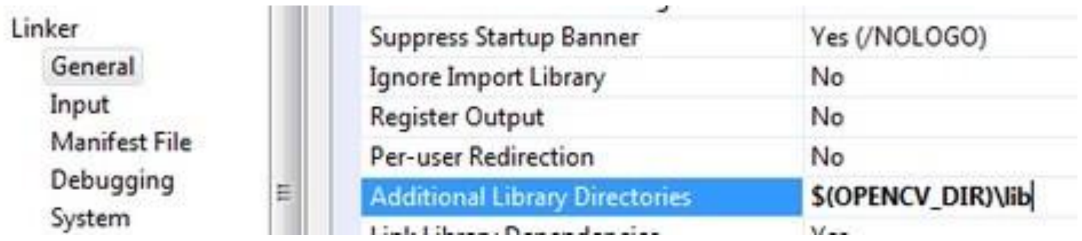
Configuration: N/A   Platform: N/A

Additional Include Directories	$(OPENCV_DIR)\..\..\include
Resolve #using References	
Debug Information Format	Program Database for Edit Ar
Common Language RunTime Support	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level1 (/W1)
Treat Warnings As Errors	No (/WX-)

Figure 13 Additional Include Directories

The environment variable “OPENCV_DIR” is used to let adding the property sheet to a different project easily. Because the environment variable is changed with its value during runtime.

Then in “Linker” group’s “General” tab, in the “Additional Library Directories” line “$(OPENCV_DIR)\lib” is added as shown in Figure 14.



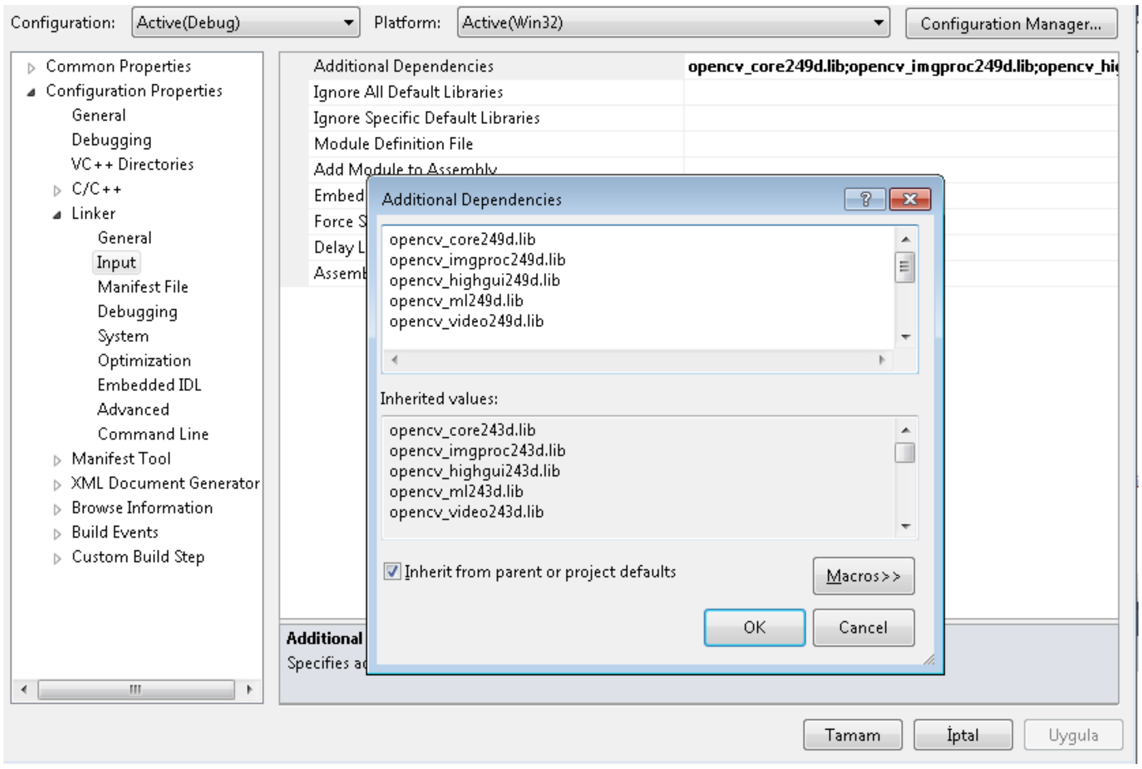
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	$(OPENCV_DIR)\lib

Figure 14 Additional Library Directories

After this, again in “Linker” group, but this time in “Input” tab, in the “Additional Dependencies” line necessary modules are added. Below a few samples of necessary modules are listed and Figure 15 show how the listed necessary modules can be added to project:

1. opencv_core249d.lib
2. opencv_imgproc249d.lib

3. opencv_highgui249d.lib
4. opencv_ml249d.lib
5. opencv_video249d.lib
6. opencv_features2d249d.lib
7. opencv_calib3d249d.lib
8. opencv_objdetect249d.lib
9. opencv_contrib249d.lib
10. opencv_legacy249d.lib
11. opencv_flann249d.lib


Configuration: Active(Debug) Platform: Active(Win32)

Additional Dependencies: opencv_core249d.lib;opencv_imgproc249d.lib;opencv_hi

Additional Dependencies

```
opencv_core249d.lib
opencv_imgproc249d.lib
opencv_highgui249d.lib
opencv_ml249d.lib
opencv_video249d.lib
```

Inherited values:

```
opencv_core243d.lib
opencv_imgproc243d.lib
opencv_highgui243d.lib
opencv_ml243d.lib
opencv_video243d.lib
```

☑ Inherit from parent or project defaults

Figure 15 Additional Dependencies

After confirming the dialog boxes the property sheet for debug mode is ready, and can be found under project directory. The “OpenCV_Debug.props” file is ready to use. It can be copied to a new project directory and can be used without any editing.

As an example for a test program of OpenCV, these steps may be followed. Right click to project and follow this path “Add → New Item... → Visual C++ → C++ File”. Give a name to the new file and click “Add”. Insert the following code into the new file and then press F5 to run the code.:

```
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
   Mat im = imread("c:/full/path/to/an image");
   if (im.empty())
   {
      cout << "Cannot load image!" << endl;
      return -1;
   }
   imshow("Image", im);
   waitKey(0);
}
```

CHAPTER 5
EXPERIMENTS

5.1 INTRODUCTION

Experiments are classified in two groups. First class is detecting and extracting faces. Second class is recognition of faces.

In detecting experiments, only the performance of systems are compared since the same detecting and extracting code is executed on PC, Pandaboard, and BeagleBone Black. The faces used in the experiment are extracted form a TV Series Buffy season 5 episode 2. Then two different experiment is done.

In recognition experiments, three algorithms (Figenfaces, Fisherfaces, and LBPH) are compared by performance and accuracy. For recognition experiments, 3 different characters' all face images are picked up from image pool which are extracted in first experiment and face recognition is applied to these images. The movie characters, Buffy, Michelle, and Nicholas are selected for experiments. In experiment dataset total of 753 images belongs to character Buffy, Michelle's 624 images, and Nicholas's 205 images. Images total number is 1582. As male faces are better suited for face recognition algorithms, Nicholas is added to test bed so that at least one male character exist in test bed.

The experiments are named with illustrative words in the rest of the thesis for better understanding. The first experiment is named as “Detection”. The experiment 2 is named as “Recognition” and step 1 is named as “Single Track” and step 2 is named as “Multi Track”.

5.2 EXPERIMENT 1 - FACE DETECTION

Detecting and extracting faces from a video stream is a really challenging task. There is a great deal of data to process. Standard videos have 25 frames in a second. This means that there are 25 pictures for image processing in every second.

For example, the selected TV series is 42 minutes 6 seconds long. This makes 2526 seconds, and if it is multiplied with 23 frames (as this video’s frames per second is 23), in total the video consists of 58098 images. This amount of data need huge processing power. Since this thesis focuses on embedded systems, some performance improvement is necessary.

The first applied performance improvement is capturing 1 frame in every 12 frames. So this solution reduces the dataset approximately 12 times. The number is still very high. Because the time taken to process this number of images is long for real time applications. A second performance improvement is: processing all the images as in greyscale. Since RGB images has 3 dimensions and greyscale images has 2 dimensions. This reduces the dataset by three times. Applying the two performance improvement reduces the data set 36 times. But in some cases more performance solutions can be needed. In that case decreasing sample rate as in the first performance solution can be helpful, like capturing 1 frame in every 25 images or more. While decreasing sample rate, the problem’s characteristics should be considered. The characteristics of the problem can vary, for example if the scenes of the video are changing fast, decreasing sample rate can lead to track loss.

Haar cascade face detector has proven itself as a fast and accurate face detector and thus, is the most popular face detection method. That’s way this method has used in this thesis for detecting faces in video frames. More detail about haar cascade classifier is given in the following topic.

5.2.1 Haar Cascade Classifier

Haar feature-based cascade classifier is an effective and thus mostly preferred method for object detection in image processing systems. This method is based on machine learning. So it needs a training set for both true and false samples. After training, features extracted by using rectangle features shown in Figure 16 [3]:

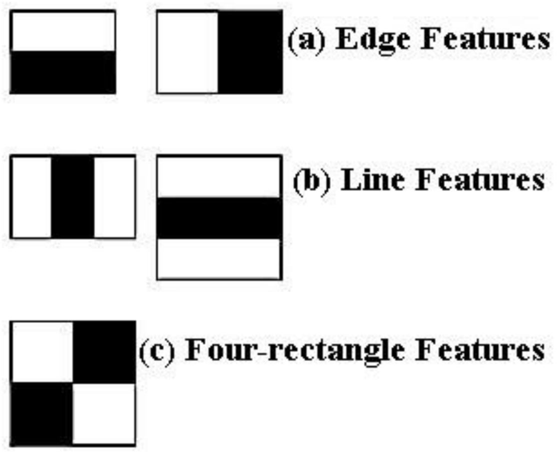


Figure 16 Haar Cascade Rectangle Features

Each feature is used to calculate a value by subtracting sum of pixels covered by white rectangle from sum of pixels covered by black rectangle. Calculating these features requires too much calculation, which makes the method unfeasible. This problem is solved by using integral image in [3]. Integral image value of a pixel is calculated by summing up all pixels' values which are above and left of the calculated pixel. The pixel p at location (x,y) is calculated with the Equation 5.1:

$$p(x,y) = \sum_{x' \leq x, y' \leq y} p'(x',y') \quad (5.1)$$

$p(x,y)$ is the integral value of the pixel located at (x,y) and p'(x,y) is original image. Using the formulas in Equation 5.2 and 5.3, the integral image calculation can be done faster:

$$s(x,y) = s(x,y-1) + p'(x,y) \quad 5.2$$

$$p(x,y) = p(x-1,y) + s(x,y) \quad 5.3$$

In the above formula $s(x, y)$ is the cumulative row sum, $s(x, -1) = 0$, and $p(-1, y) = 0$. The integral image is computed with one loop throughout the original image. Integral image enables calculating rectangular images with only four point values of integral image. Eight points are enough to calculate the rectangular feature since it is calculated by subtracting white rectangular area values from black rectangular area values. In Figure 17, how haar rectangular features are applied is shown.

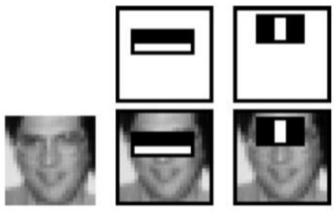


Figure 17 Haar Rectangular Features Application

5.2.2 Implementation of Detection Experiment

Below the face detection algorithm flow diagram is shown in Figure 18. As it can be seen in the figure, first the video frames are extracted from a video stream then the frames are processed with haar cascade classifier.

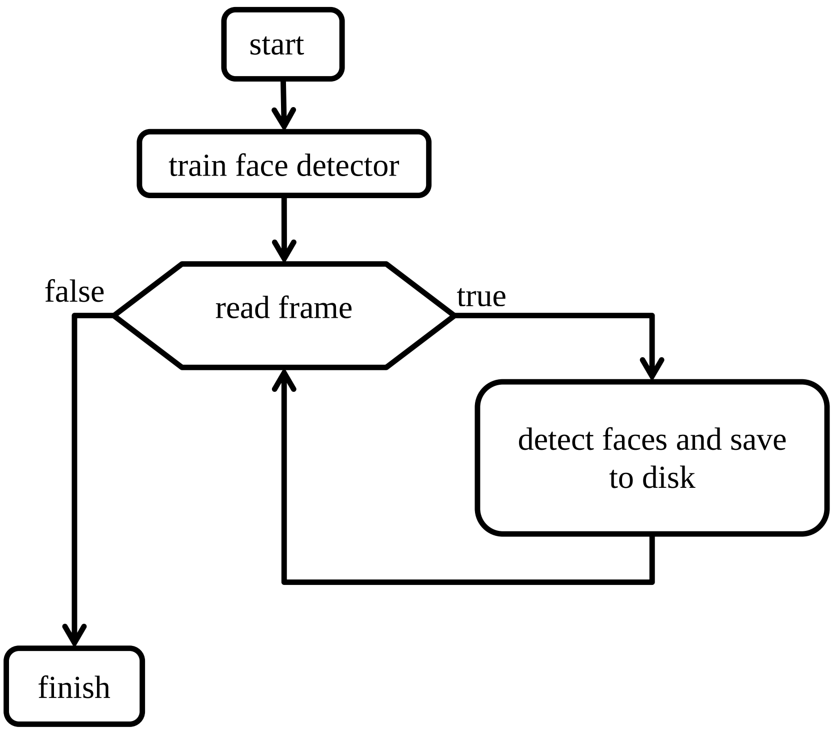


Figure 18 Face Detecting Flow Diagram

Necessary libraries for OpenCV functions are below:

```
#include <opencv2/objdetect/objdetect.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
```

Necessary libraries for C++ functions are below:

```
#include <iostream>
#include <stdio.h>
#include <direct.h>
#include <time.h>
#include <fstream>
```

“iostream” and “stdio.h” are used for standard input-output functions. “direct.h” is needed for directory operations. “time.h” contains the time functions which are used to calculate the time for processing. “fstream” is needed for writing the results to a file.

For naming the extracted images with numbers and strings, the below code is implemented:

```
string IntToStr(int n)
{
   stringstream result;
   result << n;
   return result.str();
}
```

The “IntToStr” function is needed while naming the extracted images. Because C++ does not allow strings and integers to concatenate directly.

Video stream is split into frames for face detection end extraction. OpenCV’s “CvCapture” class is used to get frames from stream:

```
CvCapture* capture = cvCaptureFromFile("buffy_s5_e2.mp4");
```

A function named “skipNFrames” is defined to realize the first performance solution:

```
IplImage* skipNFrames(CvCapture* capture, int n)
{
   for(int i = 0; i < n; ++i)
      if(cvQueryFrame(capture) == NULL)
         return NULL;
   return cvQueryFrame(capture);
}
```

In “main” function the results of “skipNFrames” function is processed in a while loop:

```
captureFrame = skipNFrames(capture, 12);
while(!captureFrame.empty())
{
  detectFaces(captureFrame,i);
  captureFrame = skipNFrames(capture, 12);
  cout << "frame :"+IntToStr(i) << endl;
  i++;
}
```

As it can be seen in the code, the face detecting and extracting process is done in main function. The function is described below and the code can be seen in APPENDIX A:

```
Define variables;
Get the current time as “start_time”;
Generate a “CascadeClassifier”;
Load "haarcascade_frontalface_alt.xml" and train the classifier;
Generate path for storing images;
Generate “CvCapture” for capturing frames from video file;
Use “skipNFrames” function to get a frame from video stream;
**While** captured frame is not empty
{
  Convert captured frame to greyscale;
  Equalize histogram of greyscale frame;
  Detect faces on equalized frame and store to “faces” vector with “detectMultiScale” function;
  Write original image to disk;
  **If** there is detected face(s)
  {
          **For** each element of “faces” vector
          {
                  Draw rectangle for face area on frame;
                  Write face area as separate file to disk;
          }
          Write original frame with rectangles to disk;
  }
  Capture next frame;
}
Get current time and subtract start_time to calculate processing time;
```

“Haar Cascade Frontalface” library is used to detect faces. All detected faces are saved as separate files with fixed to 112x92 size. On the processed frame all detected face regions are marked with a rectangle and saved as a separate file. The “detect_faces_struct” structure is used to get return information of “detectFaces” function. “detect_faces_struct” is declared as follows:

```
struct detect_faces_struct
{
  Mat frame;
```

```
    std::vector<Rect> faces;
    bool is_detected;
    detect_faces_struct(): is_detected(false) {}
  };
```

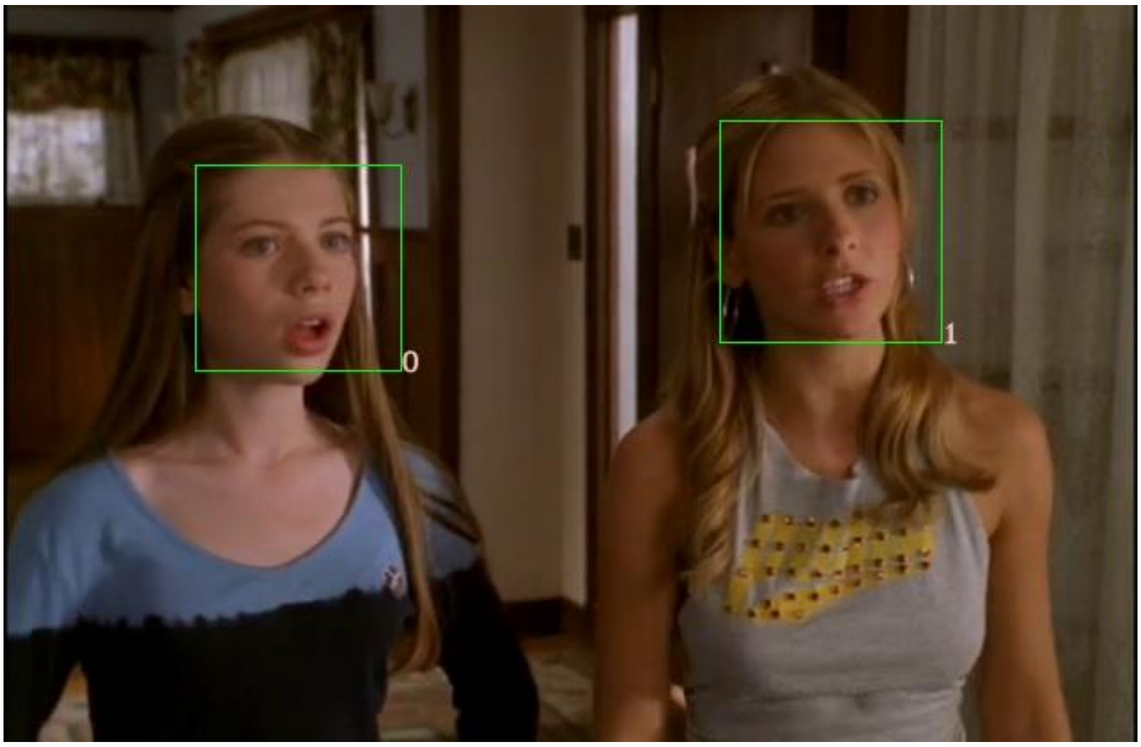


Figure 19 Detected Faces

In Figure 19 it is shown that how “detectFaces” function finds and marks the face areas on frames. Detected and extracted face regions are saved in grayscale format as seen Figure 20.

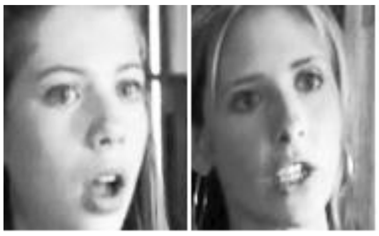


Figure 20 Extracted Faces

5.2.3 Comparison Of Detecting Performance Of Systems

The presented code in previous section is executed in three different systems: x86 system, arm based (Pandaboard), and arm based (BeagleBone Black).

Since detection and extraction process need huge amount of memory, the memory of BeagleBone Black was not large enough. A usb flash drive is attached to the BeagleBone Black to overcome the memory problem. This solution caused another problem. BeagleBone Black was able to compile the code of first experiment on usb flash drive, but it was not able to execute the program. Since the usb flash drive is formatted as non-linux format FAT32, the file permission for execute is not allowed. Another solution is added by copying the code to BeagleBone Black’s home directory and generating the image processing outputs to usb flash drive. Of course the TV Series is accessed from usb flash drive. This is because there is not free disk space on BeagleBone Black even for the video file. The code is updated accordingly.

The PC system’s configuration as follows:

1. Windows 7 Professional 64 bit operating system
2. Intel® Core™ i5-2520M CPU @ 2.50GHz
3. 4 GB of ram
4. 160 GB of disk
5. Intel® HD Graphics 3000

Since Pandaboard and BeagleBone Black configurations are described in relevant sections, they will not be mentioned again.

Metrics	***Intel***	***Arm-Pandaboard***	***Arm-BeagleBone Black***
Time For Detection Process(seconds)	1789	9960	15000
Frames Captured From Video	4658	4658	4658
Face-Found Frames	3020	3020	2990
Detected Faces From All Captured Frames	3989	3997	3970

Table 1 Face Detection Results

In Table 1, the performance evaluations of Detection experiment are shown. The intel machine has the most processing power so it surpasses the embedded systems as expected. Of course Pandaboard is at the second place, because it has more processing power than BeagleBone Black.

In Detection experiment BeagleBone Black had a disadvantage because of the insufficient memory space for the video that will be extracted and for the output files of the experiment. Therefore, these processes(reading the video file and saving the output files) are done on an external usb flash drive. Reading and writing speed of usb flash drive is lower than an internal memory. Without this drawback it is expected from BeagleBone Black to finish the experiment in %20-25 percent later than Pandaboard which means if Pandaboard finishes the experiment in 9960 seconds, BeagleBone Black should finish the experiment in about 12000 seconds, but it is ended up with 15000 which is about %34 more than Pandaboard, due to the low speed of usb flash drive.

Since the same video processed in all systems, the extracted frame counts are the same, 4658.

When it comes to the number of detected faces, the systems show some difference. This difference arises from false detections and misses, and OpenCV library versions differences.

5.3 EXPERIMENT 2 - FACE RECOGNITION

In Recognition experiment, performances of face recognition algorithms and systems are compared. So the three algorithms are executed on two different datasets one by one, and for each system this process is repeated. 18 measurements are gained.

Eigenfaces, Fisherfaces, and LBPH algorithms in OpenCV library need a training dataset for recognizing. 1582 images are split into two datasets, one for training set, other for test set. The dataset split into train and test sets in two different ways, this makes experiment with two steps.

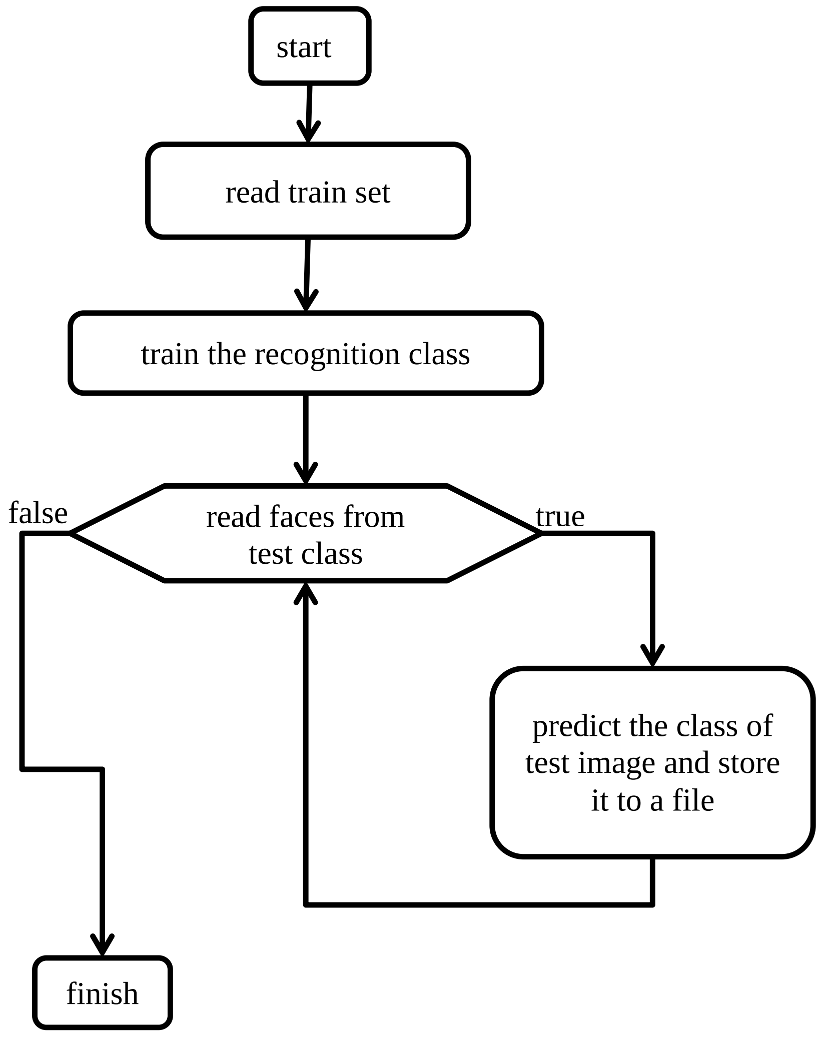

start
read train set
train the recognition class
false
read faces from test class
true
predict the class of test image and store it to a file
finish

Figure 21 Flow Diagram of Recognition Experiment

The necessary library files for OpenCV functions are included as follows:

```
#include "opencv2/core/core.hpp"

#include "opencv2/contrib/contrib.hpp"

#include "opencv2/highgui/highgui.hpp"
```

The main structure of face recognition program consists of the paths of training files and their class labels are presented in two different vectors and these two vectors are passed to training method. When training is finished, the faces in the test set are sent to recognition method.

The two vectors for paths of training images and labels of training labels are defined as follows:

```
vector<Mat> images;

vector<int> labels;
```

"Mat" is a special OpenCV structure for storing images and necessary information. Since the size of training set is not fixed for different runs, defining the image array as "vector" enables to extend the size. And of course, the true labels are needed for training the recognizer. Buffy is labeled as class "0", Michelle is labeled as class "1" and Nicholas is labeled as class "2".

As it can be seen in the code below, a "FaceRecognizer" class is generated with "createEigenFaceRecognizer" for recognition using Eigenfaces algorithm.

```
Ptr<FaceRecognizer> model = createEigenFaceRecognizer();
model->train(images, labels);
```

In face recognition experiment only this part of the code needs to be changed for three different algorithms. Next two algorithms' codes are below:

Fisherface:

```
Ptr<FaceRecognizer> model = createFisherFaceRecognizer();

 model->train(images, labels);
```

LBPH:

```
Ptr<FaceRecognizer> model = createLBPHFaceRecognizer();

model->train(images, labels);
```

After training part is done, face recognition process is started. The code explained below shows the steps of face recognition process for one character, in this case Buffy.

"predictedLabel" and "testLabel" is used to store if the recognition is correct or false, in a text file. "predictedLabel" is the answer of previously trained system and "testLabel" is the true class. "line" is used to hold the result string which will be stored in the result file. "myfile" is the file stream for the result file.

```
int predictedLabel;

Mat testSample;

int testLabel;

string line;

ofstream myfile;
```

The code below is an example of predicting(recognizing) Buffy. First the “buffy” filestream is defined to read the paths of face images of buffy from “dataset1\\buffy\\buffy.txt”.

```
//predict buffy

cout << "Predicting buffy." << endl;

std::ifstream buffy("dataset2\\buffy\\buffy.txt", ifstream::in);

if (!buffy)

  string error_message = "No valid input file was given, please check the given filename.";

  myfile.open ("buffy_results2_fisherfaces.txt");

  testLabel = 0;
```

The “while” loop below reads every test file of Buffy and sends the image to recognition system. Then the recognition result(predicted class), actual class, and the path of image is stored in result file.

```
while (getline(buffy, line))

{

  testSample = imread(line, 0);

  predictedLabel = model->predict(testSample);

  string result_message = format(";Predicted class = %d ; Actual class = %d", predictedLabel,
testLabel);

  myfile << line << result_message << endl;

}
```

The code explained above is executed for three algorithms on three different systems with two different data sets. All results are stored in text files, and then these text files are used to evaluate the results.

5.3.1 DATA SETS

In the Single Track Recognition experiment, one face track from each three characters is selected. In a scene of a video, a consecutive faces of one character are called a face track. The selected face tracks of the characters Buffy, Michelle, and Nicholas are shown in Figure 22, Figure 23, and Figure 24 below respectively:

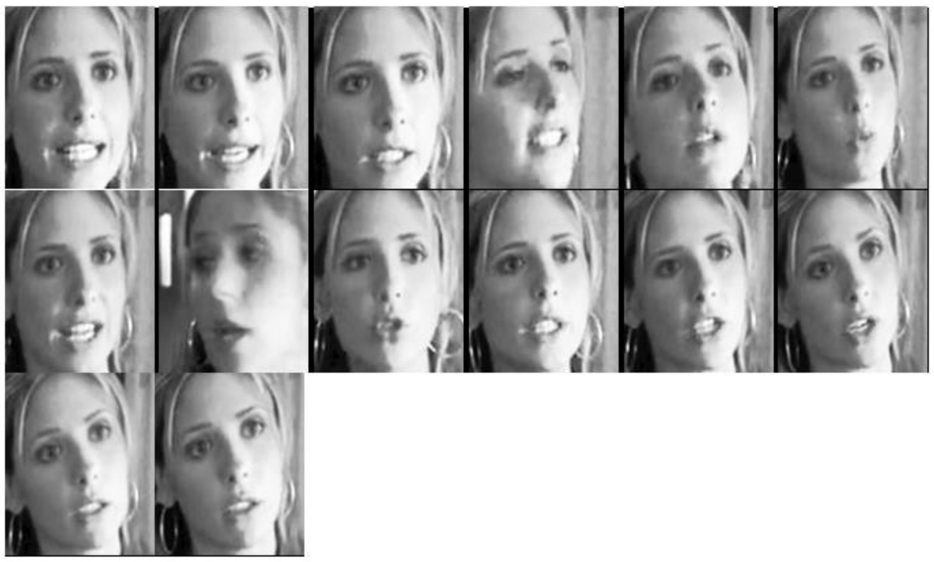


Figure 22 Buffy Face Track

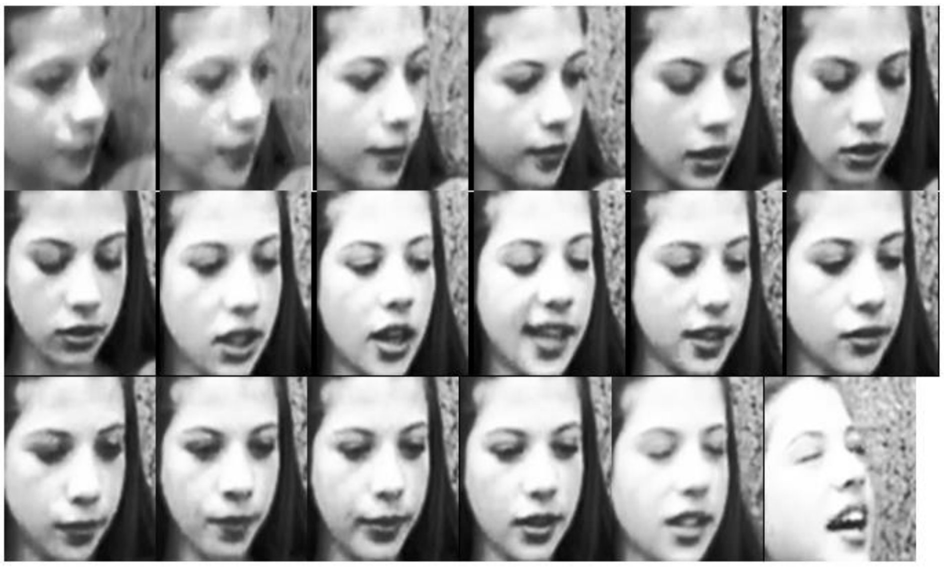


Figure 23 Michelle Face Track

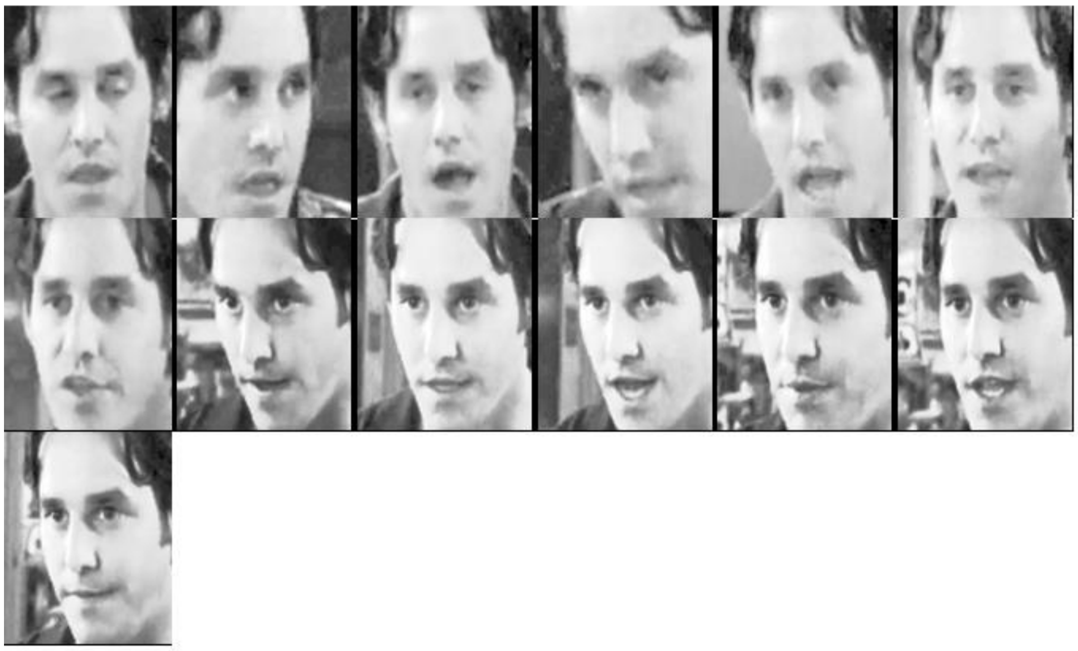


Figure 24 Nicholas Face Track

In Single Track Recognition step Buffy has 14 images, Michelle has 18 images and Nicholas has 13 images in train set. All 45 of these images are used for training step and the rest is used for testing.

The Multi Track Recognition step is different than the Single Track Recognition step. 60% of the whole images are selected uniformly for training and 40% is left for testing. Uniformly means it is aimed to pick train images from nearly every face track. For training set, 451 images of Buffy, 375 images of Michelle, and 123 images of Nicholas are selected. This makes total 949 images for training and the rest for testing.

5.3.2 COMPARISON OF FACE RECOGNITION PERFORMANCES

	Single Track		**Multi Track**	
Characters	***Train(# of face images)***	***Test(# of face images)***	***Train(# of face images)***	***Test(# of face images)***
Buffy	14	739	451	302
Michelle	18	606	375	249
Nicholas	13	192	123	82
TOTAL	45	1537	949	633

Table 2 Distribution of Datasets

As it can be seen from the Table 2, the difference of Single Track Recognition and Multi Track Recognition, enables us to learn the impact of number and characteristics of elements for training.

Using one face track for training shows the effect of lighting conditions. This is because in one face track, the illumination does not change too much, since all images are under the same illumination but when the scene is changed, the illumination differs drastically, even human eyes have trouble for recognition.

Methods	***Intel (seconds)***	***Arm-BeagleBone Black (seconds)***	***Arm-Pandaboard (seconds)***
Eigenfaces	1.91	30.65	22.11
Fisherfaces	5.96	8.3	3.88
LBPH	55.02	136.52	58.36

Table 3 Single Track Recognition Experiment Time Durations

First, the time durations of Recognition experiment is presented in Table 3 above.

In Table 3, it is clearly seen that Fisherfaces algorithm has a superior timing performance in Single Track Recognition experiment. As it is expected intel-based system has a global superiority about timing to arm-based systems. It is because the Intel processor has more capacity since arm processor are designed for mobile devices and for consuming minimum energy. If there is a gain from energy, this means there is a loss from performance. In order to improve performance of embedded systems, some solutions should be applied appropriate for embedded system specifications. For example, Pandaboard’s cpu has 2 cores, so this can be used to speed up the processing. However, this solution does not work for BeagleBone Black, since it does not have 2 cores inside its cpu.

Methods	***Intel (seconds)***	***Arm-BeagleBone Black (seconds)***	***Arm-Pandaboard (seconds)***
Eigenfaces	432.08	3600	2128.78
Fisherfaces	549.84	3360	1836
LBPH	143.61	907.02	370.87

Table 4 Multi Track Recognition Experiment Time Durations

In Table 4, the performance sequence changes, since the dataset is changed. This shows how the dataset affects the performance of algorithm. In general view the amount of the training set has negative effect on performance. All the algorithms are effected from training set size change according to algorithm characteristics. This situation gives some clues for selecting the proper algorithm for a certain problem or project.

Methods	***Intel***	***Arm-BeagleBone Black***	***Arm-Pandaboard***
Eigenfaces	226.22	117.46	95.28
Fisherfaces	92.25	404.82	473.2
LBPH	2.6	6.64	6.35

Table 5 Effect Ratio of Timing Performance(Multi Track / Single Track)

In Table 5, the time results of all three algorithms in Single Track and Multi Track experiments from Table 3 and Table 4 are compared. The value of Table 4 is divided with the value of Table 3. It can be seen from the Table 5, the most robust algorithm is LBPH algorithm. Since Eigenfaces and Fisherfaces algorithms' performances change by 92 to 473 times, LBPH algorithm's performance changes only 2.6 to 6.64 times.

Methods	***True Answers***	***False Answers***	***Hit Ratio (%)***
Eigenfaces	509	1028	33
Fisherfaces	519	1018	34
LBPH	679	858	44

Table 6 Accuracy Results of Single Track Recognition Experiment

In Table 6, accuracy measurements of the Single Track Recognition Experiment is shown. It can be indicated that LBPH algorithm is superior to other methods. LBPH algorithm has gained %10 more accuracy than other algorithms. Since there are three different classes (three different persons), the accuracy level about %33 is like random guessing. This level of accuracy is not acceptable. There should be some improvements more than random guessing if some special algorithms are implemented. The failure of the algorithms in Single Track Recognition experiment is not due to their inadequacy, but they should be implemented in an appropriate way in which the algorithm works better. This means that the experiment environment should be adjusted according to the algorithms' needs, where the algorithm can run more efficiently. It can be figured out that while implementing an algorithm, all the needs of the algorithm should be fulfilled. After that it can run better and give more healthy results for what it is intended to.

Another result that can be derived from Table 6 is that the illumination variance has a huge negative effect on recognition success of all selected three algorithms. This is because selecting one face track for training is not sufficient. In a face track the illumination of the scene does not change so much. So the algorithm is trained only for one illumination condition and when illumination changes, the algorithm fails to recognize the face correctly. Even human eyes has troubles while recognizing faces in different illumination conditions.

Methods	***True Answers***	***False Answers***	***Hit Ratio (%)***
Eigenfaces	586	47	93
Fisherfaces	540	93	85
LBPH	594	39	94

Table 7 Accuracy Results of Experiment II Step 2

In Table 7, accuracy results of Multi Track Recognition experiment are shown. While selecting training sets, it is considered that the set should have members from nearly all face tracks. With this in mind, the illumination variance effect could be reduced. Because every scene has different characteristics, having members from all scenes is helpful to face recognition algorithm for learning different light conditions, and how the face looks like in those illumination conditions. It is like human vision, if a person is seen once(one face track), recognizing that face is hard, but if that person is seen many times in many different places, it is easy to recognize even in very hard situations like darkness or one way light source.

In Multi Track Recognition experiment, LBPH surpasses other two algorithms, both in time performance and accuracy rate. Of course, all three algorithms gained very high accuracy. This is because the training set enables the algorithms to learn the classes(characters) very well in detail.

In problems that can provide a training set these algorithms are very handy. Especially LBPH has a superior performance than other two algorithms.

5.4 FUTURE WORK

Eigenfaces, Fisherfaces and LBPH methods require a training class for face recognition. In real world applications many times using a training set is not feasible, if the system should be used on many different image sources. Training set makes system slower and needs more storage place. Because of this, using unsupervised classification could help improve this limitations. Image clustering is an alternative solution for face recognition systems which are faced with faces that are not in the training set. Clustering also needs some methods to be able to use the features of images. Different feature extraction methods can be implemented and different clustering methods can be used on the image feature database.

Timing results of algorithms should be increased. This can be done using multithreading and parallel processing techniques. BeagleBone Black has single core so implementing parallel processing on it, is not feasible, but multithreading can be applied to all three different systems. Pandaborad has A9 cortex, which means it has two cores. These cores enable both multithreading and parallel processing. This feature enables Pandaboard to be more appropriate for solutions with multithreading and parallel processing.

CHAPTER 6
CONCLUSION

In this thesis, face recognition is studied both how different algorithms demonstrate performance and how different system architectures affect performance. The main aim of this thesis is to investigate the face recognition algorithms on embedded systems. Therefore, one intel based PC system and two arm based embedded systems are used. The intel based PC system is selected as a control class, if the performance metrics of embedded systems are low or high when compared to intel based PC system's performance metrics.

Three different algorithms are selected to expose the effects of algorithms on face recognition systems since the face recognition has been searched on embedded systems in this thesis.

When all the results of Detection and Recognition experiments are considered, some inferences can be exposed.

In the Detection experiment, it is proved how the system's architecture and capacity affects the performance. There is a negative effect of embedded architecture on timing performance, in both Detection and Recognition experiments. As it can be seen in "Table 1 Face Detection Results" even the Pandaboard which is among high performance embedded systems, has shown performance about 4 times slower than the intel based PC system. Thus, there are urgent challenges in face detection on embedded systems to be improved.

In the Recognition experiment, there are two steps, which are also different experiments actually. Since their scopes are the same, they are both put in one experiment as two steps. The difference of the two steps is the training and test datasets. In Single Track step, one face track is selected as train set for each character. This makes the train set 2.8% of the dataset when three characters' train sets are calculated. This makes the system not to be trained very well for the whole dataset. In fact, there are three different faces' examples in the train set, which brings to mind that it should be enough to recognize. This is not true at all. Because using one track does not take the illumination effect into consideration. In one track the illumination does not change at all, besides the face does not move too much. So the algorithm cannot train itself enough. That’s why the algorithms could not show good performance, even close to random selection.

In Multi Track Recognition step, the success rate of the three algorithms can be seen as required(85%, 93%, and 94%). This is because there is enough member in training set for both challenges, illumination and pose. In Multi Track step, the train set is chosen as it will be 60% of the whole dataset. The images are chosen homogenous. This allows algorithms to be able to learn the faces for different illumination conditions and different poses.

In conclusion, face recognition is at an acceptable level for some algorithms such as LBPH. However, face detection process requires some improvements which is subject for further research.

CHAPTER 7
REFERENCES

[1] Wikipedia, Online, Available: http://en.wikipedia.org/wiki/Facial_recognition_system, 12.12.2013

[2] Topcoder, Online, Available: http://www.topcoder.com/blog/about-face-how-will-innovators-use-facial-recognition-technology/, 15.12.2013

[3] V. Paul and M. J. Jones, «Robust real-time face detection,» *International journal of computer vision,* pp. 137-154, 2004.

[4] Pham-Ngoc, Phuong-Trinh, and K.-H. Jo, «Color-based face detection using combination of modified local binary patterns and embedded hidden markov models,» *SICE-ICASE, 2006. International Joint Conference. IEEE,* 2006.

[5] A. F. Abate, M. Nappi, D. Riccio, and G. Sabatino, «2D and 3D face recognition: A survey,» *Pattern Recognition Letters,* volume 28, no. 14, pp. 1885-1906, 2007.

[6] M. A. Turk and A. P. Pentland, «Face recognition using eigenfaces,» *Computer Vision and Pattern Recognition*, 1991.

[7] P. Belhumeur, J. Hespanha, and D. Kriegman, «Eigenfaces vs. Fisherfaces: recognition using class specific linear projection,» *Pattern Analysis and Machine Intelligence, IEEE Transactions,* volume 19, no. 7, pp. 711 - 720, 1997.

[8] O. Yamaguchi and K. Fukui, «Smartface-A Robust Face Recognition System under Varying Facial Pose and Expression,» *The trans. of the Institute of Electronics, Information and Communication Engineers D-II,* volume E86-D, no. 1, pp. 1045-1052, 2001.

[9] Aby P.K, Jose A.., Jose B., Dinu L.D, J. John, and Sabarinath G, «Implementation and Optimization of Embedded Face Detection System,» *Proceedings of 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies*, 2011.

[10] Bigdeli A., Sim C., Biglari-Ahbari M,. and Lovell B.C., «Face Detection on Embedded Systems,» *Embedded Software and Systems Lecture Notes in*

Computer Science, volume 4523, pp. 295-308, 2007.

[11] N. Aaraj, P. U. N. Dept. of Electr. Eng., S. Ravi, S. Raghunathan ve N. Jha, «Architectures for Efficient Face Authentication in Embedded Systems,» %1 içinde *Design, Automation and Test in Europe*, 2006.

[12] M. Sharkas, M. Abou Elenien, “Eigenfaces vs. Fisherfaces vs. ICA for Face Recognition; A Comparative Study,” 9th International Conference on Signal Processing, 2008, ICSP 2008., 2008, pp. 914–919

[13] P. N. Belhumeour, J. P. Hespanha, and D. J. Kriegman, “Eigenface vs. Fisherfaces: Recognition Using Class Specific Linear Projection”, IEEE Trans. on PAMI, 19(7), pp. 71 I-720, 1997

[14] T. Ojala, M. Pietikäinen, and D. Harwood (1994), "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions", Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994), vol. 1, pp. 582 – 585

[15] D. Huang , C. Shan , M. Ardabilian , Y. Wang and L. Chen "Local binary patterns and its application to facial image analysis: A survey", IEEE Trans. Syst., Man, Cybern., C, vol. 41, pp.765 -781 2011

[16] OpenCV, Online, Available: http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html#eigenfaces, 20.12.2014

[17] Eyalarubas, Online, Available: http://eyalarubas.com/face-detection-and-recognition.html, 20.12.2014

[18] T. Ojala, M. Pietikäinen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 7, pp. 971–987, Jul. 2002.

[19] [6] H. Yang and Y. Wang, “A LBP-based face recognition method with Hamming distance constraint,” in Proc. Int. Conf. Image Graph., Aug., 2007, pp. 645–649.

[20] BeagleBoard, Online, Available: http://beagleboard.org/BLACK, 21.12.2014

[21] Elinux, Online, Available: http://elinux.org/Beagleboard:BeagleBoneBlack,

21.12.2014

[22] Pandaboard, Online, Available: http://pandaboard.org/, 21.12.2014

[23] Omappedia, Online, Available: http://omappedia.org/wiki/Main_Page, 21.12.2014

[24] Do Van Quyen Online, Available: https://dovanquyen.wordpress.com/2012/12/11/installing-opencv-on-arm-board/, 21.12.2014

[25] OpenCV, Online, Available: http://opencv.org/downloads.html, 22.08.2014

[26] F. Samaria and A.C. Harter, "Parameterisation of a stochastic model for human face identification," Proc. Second IEEE Workshop Applications of Computer Vision, 1994.

[27] S. Tamura, H. Kawa, and H. Mitsumoto, "Male/Female identification from 8_6 very low resolution face images by neural network," Pattern Recognition, vol. 29, pp. 331-335, 1996.

APPENDIX A

CODE OF DETECING FACES

```
int main(int argc, const char** argv)
{
        int i = 0, count = 0;
        detect_faces_struct detected_faces;
        Mat captureFrame;
        Mat* point_mat;
        string path;

////////////////////////////////////////////////////
        //get date for result file names
        time_t rawtime;
        struct tm * timeinfo;
        char buffer[80];

        time (&rawtime);
        timeinfo = localtime (&rawtime);
        strftime(buffer,80,"%Y-%m-%d",timeinfo);
        std::string date_str(buffer);

////////////////////////////////////////////////////
        //open file for results
        ofstream myfile;
        myfile.open ("results_face_detect.txt");
        clock_t lbph_Start = clock();

        //setup image files used in the capture process
        Mat grayscaleFrame;
        Mat face;

        //create the cascade classifier object used for the face detection
        CascadeClassifier face_cascade;
        //use the haarcascade_frontalface_alt.xml library
        face_cascade.load("haarcascade_frontalface_alt.xml");
```

```
    //create a vector array to store the face found
    std::vector<Rect> faces;

    path = "imagesmain";
    mkdir(path.c_str());

    //create a loop to capture and find faces
    CvCapture* capture = cvCaptureFromFile("buffy_s5_e2.mp4");
    captureFrame = skipNFrames(capture, 12);
    while(!captureFrame.empty())
    {
        //convert captured image to gray scale and equalize
        cvtColor(captureFrame,grayscaleFrame,CV_BGR2GRAY);
        equalizeHist(grayscaleFrame,grayscaleFrame);

        //find faces and store them in the vector array
        face_cascade.detectMultiScale(grayscaleFrame, faces, 1.1, 3,
CV_HAAR_SCALE_IMAGE, Size(40,40));

        imwrite(path+"/frame_"+IntToStr(i)+"_orj.jpg", captureFrame);
        if(0 < faces.size())
        {
            //draw rectangle for all found faces in the vector array on the original image
            for(int i_face = 0; i_face < faces.size(); i_face++)
            {
                Point pt1(faces[i_face].x + faces[i_face].width, faces[i_face].y +
faces[i_face].height);
                Point pt2(faces[i_face].x, faces[i_face].y);

                rectangle(captureFrame,pt1,pt2, cvScalar(0,255,0,0),1,8,0);
                putText(captureFrame, IntToStr(i_face), pt1,
FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1, CV_AA);

                //extract face
                Rect ROI (faces[i_face].x, faces[i_face].y, faces[i_face].width,
faces[i_face].height);
                                        Mat imageROI = grayscaleFrame(ROI);
                                        //resize image to 92*112
                                        resize(imageROI, imageROI, Size(92, 112), 0, 0, INTER_CUBIC);
                                        //write image to folder
                                        imwrite(path+"/frame_" + IntToStr(i) + "_face_" + IntToStr(i_face) +
".bmp", imageROI);
                              }
                              imwrite(path+"/frame_"+IntToStr(i)+"_rect.jpg", captureFrame);
                    }

                    //detectFaces(captureFrame,i);
                    if(i % 100 == 0) cout << "main() frame :"+IntToStr(i) <<  endl;

                    captureFrame = skipNFrames(capture, 12);
                    i++;
          }
          myfile << "Time taken for face detecting : " << (double)(clock() -
lbph_Start)/CLOCKS_PER_SEC;
          cout << "Time taken for face detecting : " << (double)(clock() -
lbph_Start)/CLOCKS_PER_SEC;
          myfile.close();
          system("PAUSE");
   return 0;
}
```