DEVELOPMENT AND ANALYSIS OF EMBEDDED FACE RECOGNITION SYSTEM USING RASPBERRY PI

by

FALAH HASSAN ALWAN
1432321157

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science (Embedded System Design Engineering)

School of Computer and Communication Engineering
UNIVERSITI MALAYSIA PERLIS

2015

ACKNOWLEDGMENT

First and foremost, I would like to praise and thank Allah the almighty (SWT), who has granted me countless blessing, knowledge and opportunity to write this project. I offer thanks and profound gratitude to my supervisor Professor Dr. R.badlishah Ahmad who support me with his experience and knowledge which has opened new vistas to me in Embedded system Engineering. I would like to thank my wife Dr. Rasha Thabit, who her been very helpful to me. I would like to thank the UniMAP and School of Computer and Communication Engineering lecturer for their support during this intake. A great deal of thanks goes to my father and mother for their endless support to me.

FALAH HASSAN ALWAN

UNIVERSITY MALAYSIA PERLIS (UniMAP)

forstudy13@gmail.com

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D	Two-dimensional space
3D	Three -dimensional space
AC3	Audio codec
ARM	Advanced RISC Machine
ASF	formerly "Advanced Streaming Format"
AVI	Audio Video Interleaved
BSP	Board support package
CMOS	Complementary Metal–Oxide–Semiconductor
CPU	Central Processing Unit
CSI	Camera Serial Interface
DivX	Brand name of products created by "DivX, LLC."
DV	Digital Video format
EFM	Enhanced Fisher Model
EHMM	embedded hidden Markov model
ext3	Third Extended Filesystem
fat32	File Allocation Table 32
FLDA	Fisher's Linear Discriminant analysis
FRRP	Face Recognition using Raspberry Pi
HCI	Human–computer interaction
HD	High-Definition
HDMI	High-Definition Multimedia Interface
HMM	hidden Markov model
HW	Hardware
I/O	input/output
ID	identification

IEEE	Institute of Electrical and Electronics Engineers
JPEG	Joint Photographic Experts Group
MIPI	Mobile Industry Processor Interface
MP3	MPEG-1 or 2 Part 3
MPEG	Moving Picture Experts Group
OGG	Ogging
OpenCV	open source computer vision
OS	Operating System
PCA	principal component analysis
PDAs	personal digital assistant
PIL	Python Imaging Library
PKG	Packaging
RAM	Random-access memory
RGB	RGB color space(RGB)
RPi	RASPBERRY PI
RTOS	real-time operating systems
S5	camera connector socket in Raspberry Pi
SBC	Single Board Computer
SD	Secure Digital
SDHC	SD High-Capacity
SDIO	Secure Digital Input Output,
SDSC	Secure Digital Standard Capacity
SDXC	Secure Digital eXtended Capacity
SW	Software
TIFF	Tooth Interior Fatigue Fracture
TS	Technologic system
TV	Television

USB	Universal Serial Bus
VGA	Video Graphics Array
wpa_gui	WPA Graphical User Interface
XML	Extensible Markup Language

Pembangunan dan Analisis Sistem Terbenam Wajah menggunakan Raspberry Pi

ABSTRAK

Wajah manusia adalah bahagian yang paling ketara yang boleh digunakan untuk mengenali seseorang. Terdapat banyak sistem yang tersedia untuk mengenali wajah di pasaran, tetapi ianya besar, mahal, dan monopoli. Pelaksanaan teknik pengecaman wajah dalam sistem embedded adalah aspek yang sangat penting. Dalam projek ini akan membahaskan tentang reka bentuk yang tepat pada masanya, mudah alih, kos rendah sistem pengenalan wajah. Tujuan-tujuan yang boleh diperolehi dengan menggunakan system embedded, yang merupakan sistem komputer tujuan khas yang berupaya untuk melaksanakan set yang sangat kecil dari aktiviti-aktiviti yang ditetapkan. Dalam kajian ini, tahap pembangunan yang terdiri daripada Satu Papan Komputer (SBC, Raspberry Pi (Model A) sebagai proses menyatukan, GNU / Linux berasaskan system operasi Embedded Raspbian sebagai platform pembangunan aplikasi. Projek ini memberi tumpuan untuk menerapkan algoritma pengecaman wajah yang sesuai dengan Raspberry Pi (Model A) sistem yang dicadangkan diimplementasikan menggunakan pemproses ARM11 dan memori yang tidak cekap pada Raspberry Pi (Model A) lembaga, untuk mendapatkan prestasi yang boleh diterima dari sistem, gambar yang ditangkap pada resolusi (320 × 240), sistem perlu ≈2.1 saat untuk memproses imej yang ditangkap, system embedded dapat ditingkatkan apabila teknik pengesanan gerakan diterapkan.

Development and Analysis of Embedded Face Recognition System using Raspberry Pi

ABSTRACT

Human Face is the most visible part which can be used to recognize persons. There are many available systems for face recognition in the market, but they are bulky and expensive. The implementation of face recognition techniques in an embedded system is a very important aspect. This project involves design of a real-time, portable, low embedded cost face recognition system. Implementation and analysis of face recognition techniques on an embedded system, the development phase consists of Single Board Computer (SBC, Raspberry Pi (Model A) as process unite, and GNU/Linux based Embedded Raspbian Operating system is used as application development platform. This project focuses to apply the face recognition algorithm that is suitable with Raspberry Pi (Model A) The proposed system is implemented using ARM11 processor and inefficient memory on Raspberry Pi (Model A) board, to get an acceptable performance of the system, the images are captured at resolution (320×240), the system needs ≈ 2.1 sec to process the captured images, The performance of the embedded system is done by evaluating detection time and recognition time (is 1.75 sec, between 0.29 sec to 0.74 sec) respectively, together with CPU utilization and RAM utilization (33%, 17.75%) for detection and (36.5%, 22%) for recognition. Results obtained shows that the overall performance on the embedded system can be increased when motion detection techniques is applied.

CHAPTER 1

INTRODUCTION

1.1 Overview

Over the years, face recognition systems have been developed in order to be used in different applications such as security systems (e.g., controlling the access of the people to specific buildings, identifying the criminals in public places, etc.). The system should be able to detect face image, extract its features, and recognize it (Parmar and Mehta, 2013). Face detection and recognition algorithms have been developed rapidly in terms of performance and speed, but most of the researches and improvements have been directed towards software algorithms and their implementation. Many real-time face recognition algorithms have been successfully implemented; however, they require expensive hardware to operate (Theocharides, 2006). The application domains extend to portable environments, therefore, an accurate real-time face recognition system that can facilitate commercial needs is desirable.

1.2 Problem Statement

There are many available systems for face recognition in the market, but they are bulky, expensive, and monopoly (Mehrab et. al., 2012). Making face recognition systems ubiquitous require substantially reduction in the system complexity, size, and price (Sun, 2007). These aims can be obtained by using an embedded system, which is a special purpose computer system that is capable of performing very small sets of designated activities. Embedded systems have many characteristics such as small size, real-time operation, lightweight, portable, and low cost. Some embedded systems have been designed to perform the face recognition process using a Single Board Computer (SBC) TS-5500 and iris detection algorithm (Ahmad, 2010; Shuhaizar, 2010). There are various SBC provider and each board is different from the other in the design, hardware, and its operating system. Therefore, aiming for an improved performance, other SBC boards can be suggested to be used for implementing and performing the face recognition process in the embedded system. In addition, other face recognition algorithm can be chosen to be implemented on the suggested board.

1.3 Research Objectives and Aim

1) To design and implement a specific system capable of performing real time face recognition process using a new SBC board.
2) To select and modify face detection and recognition algorithms that can be used with the selected board.
3) To evaluate the suggested and implemented face recognition system in real time.

1.4 Research Scope

This research focuses on developing an embedded system for face recognition process using a Single Board Computer (SBC) called Raspberry Pi (Raspberry Pi Foundation, 2014). The work in the hardware part includes preparing the hardware components, setting connected devices, selecting the packages and libraries that will be used, and installing them onto the GNU/Linux Operating System (OS). For the software implementation, the Haar Feature-based Cascade Classifier (Lienhart and Maydt, 2002) and the Fisherface algorithm (Belhumeur et al., 1997) are chosen to perform the face detection, tracking, and identifying tasks.

1.5 Thesis Outline

The remaining part of the thesis is organized as follows:

- Chapter 2: presents the literature review for the project consisting of embedded systems, Single Board Computers (SBC), and face recognition systems.
- Chapter 3: explains the methodology process used to implement the embedded system prototype and discusses the configuration of its two main parts (i.e., the hardware part and the software part).
- Chapter 4: covers the software development.
- Chapter 5: shows the results obtained from the experimental prototype and the efficiency of the developed system.
- Chapter 6: presents the conclusions of the research and some recommendations for the future research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The high growth in the technology and the augmentation of the integrated circuit technology opened new challenges and development of advanced embedded system for different applications. One of the applications that have attracted interest is the embedded system for the face recognition process. This chapter presents background information and reviewing techniques that are related to the topic of this research. Next section provides the background information about embedded systems and their specifications. Then embedded operating system and GNU/Linux are explained. Thereafter, face recognition systems and techniques are reviewed.

2.2 Embedded Systems

Embedded system is a combination of hardware and software, which is designed to perform specific tasks. Early use of the embedded systems returns to the 1960s where it has been used for controlling the electromechanical telephone switches (Ahmad, 2010). Nowadays, the embedded systems are used in different fields such as military fields, medical fields, and many others. Examples on uses of the embedded systems are as follows (Ahmad, 2010):

- Network equipment such as firewall, router, and switches.
- Consumer equipment such as MP3 players, cell phones, PDAs, digital cameras, and camcorders.
- Household appliances such as microwaves and washing machines.

The operating systems required for Embedded system is unlike that for the fully featured personal computers. Embedded systems are restricted with small memory footprint and a fraction of the processing power that is usually available in desktop personal computers; therefore, it requires a different kind of operating system specifically built for embedded system environment. Design of the embedded system consists of designing the hardware and the software in parallel. The life cycle of designing the embedded system passes through some specific phases as illustrated in Figure 2.1 (Berger, 2002).

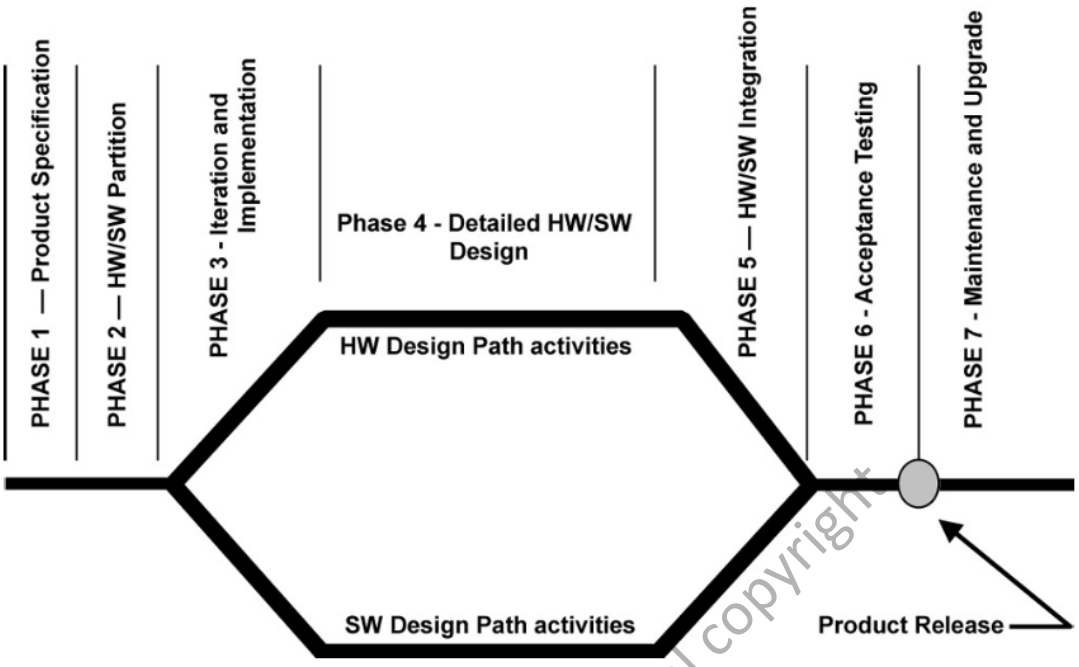

Figure 2.1: A schematic Representation Of The Embedded Design Fife Cycle (Berger, 2002).

The seven phases shown in Figure 2.1 are as follows (Berger, 2002):

1. Product specification and requirements.
2. Partitioning of the design into its software (SW) and hardware (HW) components.
3. Iteration and improvement of the partitioning.
4. Independent hardware and software design tasks.
5. Combination of the hardware and software components.
6. Product testing, evaluation, and release.
7. On-going maintenance and upgrading.

The economics and the reality of a design requirement usually force the decisions to be made before the designers of the embedded system can consider the best design trade-offs. Therefore, the designer looks for the prospect of obtaining a design in which the requirement constraints are minimal and can be strictly specified in terms of performance and cost goals.

2.2.1 Single Board Computer (SBC)

Single board computer (SBC) is an example of the embedded system that has recently attracted a lot of interest. A single board computer is any electronic system with at least a microprocessor, memory, and I/O that fits on a single circuit board (SBC Information and Resources, 2014). There exist different SBC provider companies and each board is different in the design, the hardware, and the operating system. Examples of the SBC boards are Raspberry Pi (RPi) (RASPBERRY PI, 2014), BeagleBone Black (BeagleBone Black, 2014), Parallela (Parallela, 2014), ODROID-XU3 (ODROID Platforms, 2014), Hackberry (Miniand Products, 2014), UDOO (UDOO Features, 2014), APC Rock, Cubieboard2, Marsboard, A13-OLinuXino (Open Source Hardware Boards, 2014), and Technologic Systems (TS7800) (Technologic Systems, 2014). The choice of the SBC board depends on the targeted application, the cost limitations, and the preferred performance. Table 2.1 lists SBC boards that have been mentioned in this section with some of their specifications (i.e., CPU, RAM, OS, I/O ports, and the storage unit).

Table 2.1: Specifications Of Some SBC Types

SCBs	Specifications	
Raspberry Pi	**Components**	**description**
	Chip	Broadcom BCM2835 SoC full HD multimedia applications processor
	CPU	700 MHz Low Power ARM1176JZ-F Applications Processor
	GPU	Dual Core VideoCore IV®Multimedia Co-Processor
	Memory	256MB SDRAM (model A) or 512SDRAM (model B)
	I/O ports	USB2.0, Ethernet (model B), HDMI, RCA, Audio jack and GPIO
	Onboard Storage	SD, MMC, SDIO card slot
	Operating System	ARM Linux distributions supporting ARMv6.Two main OS supported: Debian and Arch Linux ARM
BeagleBone Black	**Components**	**description**
	Chip	Based on Sitara XAM3359AZCZ100 processor
	CPU	AM335x 1GHz ARM® Cortex-A8
	GPU	PowerVR SGX530
	Memory	512MB DDR3 RAM
	I/O ports	USB2.0, Ethernet, HDMI, and GPIO
	Onboard Storage	2GB eMMC & micro SD card slot
	Operating System	ARM Linux distribution.2 main OS supported: Ubuntu and Angstrom
Cubieboard2	**Components**	**description**
	Chip	Based on the AllWinnerTech SOC A20
	CPU	ARM @Cortex™-A7 Dual-Core ARM 1GHz
	GPU	ARM® Mali400MP2, Complies with OpenGL ES 2.0/1.1
	Memory	1GB DDR3 @480M
	I/O ports	USB2.0, Ethernet, HDMI, GPIO and IR
	Onboard Storage	NAND+MicroSD or TSD+ MicroSD or 2*MicroSD
	Operating System	Mac, Linux, windows

Table 2.1: Continued

SCBs	Specifications	
Parallella	**Components**	**description**
	Chip	Based on the Epiphany 16-core CPU E16G301
	CPU	Xilinx Zynq Dual-core ARM A9 XC7Z020 1GHz
	GPU	16 or 64-core Epiphany Multicore Accelerator
	Memory	1 GB DDR3
	I/O ports	USB2.0, Ethernet, HDMI and GPIO
	Onboard Storage	Micro-SD
	Operating System	ARM Linux distribution.2 main OS supported: Ubuntu
UDOO	**Components**	**description**
	Chip	Based on the Freescale i.MX 6
	CPU	ARM Cortex-A9 CPU Dual/Quad core 1GHz
	GPU	GPU Vivante GC 2000 for 3D + Vivante GC 355 for 2D (vector graphics) + Vivante GC 320 for 2D
	Memory	RAM DDR3 1GB
	I/O ports	USB2.0, Ethernet, HDMI,LDVS, and GPIO
	Onboard Storage	Micro-SD (boot device) and SATA (Only Quad-Core version)
	Operating System	Android 4.3 Jelly Bean and ARM Linux distribution Ubuntu
TS 7800	**Components**	**description**
	Chip	Marvell MV88F5182
	CPU	ARM9 CPU 500MHz
	GPU	Non
	Memory	128MB DDR-RAM
	I/O ports	USB2.0, Ethernet, RS-485 , ADC, DIO, LCD, and GPIO
	Onboard Storage	SD, MMC, SDIO card slot
	Operating System	Kernel 2.6 and Debian Linux

2.2.2 Embedded Operating System (OS)

Embedded OS is an operating system specified for embedded computer systems. These operating systems are designed to be fast, lightweight, reliable, efficient at resource usage, and uses relatively few resources. In addition, discarding from many functions and service that non-embedded OS provide, which may not be used in the

applications runs on embedded OS. Some embedded systems are real-time operating systems (RTOS) such as ARM Linux distribution, Windows CE, and Minix (Shibu, 2009).

Usually, the embedded OS is running on very limited hardware resources such as RAM and CPU, therefore, the functions and services that are made for the embedded OS must be very specific. These functions and services are created to cover specific tasks or scopes in order to get advantage of the computer's hardware resources.

2.2.3 Embedded Linux

Linux kernel (GNU/Linux) operating systems are used in embedded systems such as smart TVs, in-vehicle infotainment, networking equipment, machine control, industrial automation, navigation equipment, spacecraft flight software, and medical instruments. In general, the Linux kernel embedded system is widely used because of two reasons the first is the ease of customization a kernel for devices and the second is the low cost. Linux has been shipped in a lot of embedded systems (Hongjun, 2010).

Because of the availability of the source code and the communities surrounding the devices, the enhancement of the Linux distribution on the device has been made possible. The advantages of embedded Linux include (a) multiple suppliers for software, development, and support, (b) no royalties, or licensing fees, (c) a stable kernel, and (d) the ability to read, modify and redistribute the source code. The technical disadvantages include (a) a comparatively large memory footprint (kernel and root filesystem), (b) complexities of user mode, and (c) kernel mode memory access, and a complex device drivers framework.

2.3 Face Recognition Systems and Techniques

Face recognition systems and techniques have been presented from many years ago (Chellappa et al., 1995; Belhumeur, 1997); within the years, numerous algorithms have been implemented. At first, the face recognition was treated as a 2D pattern recognition problem. To identify a person's face, the distance between some important points has been calculated, e.g., the distance between the eyes or measuring different angles of facial components. A fully automatic face recognition system is required. Face recognition is such a challenging yet interesting problem that it has attracted researchers who have different backgrounds: computer vision, psychology, pattern recognition, neural networks, and computer graphics (Parmar and Mehta, 2013). Different methods have been used for face recognition process such as:

- Holistic Matching Methods
- Feature-based methods
- Hybrid methods

In the holistic approaches, the whole face region is taken into account for generating the input data into face detection system. One of the best examples of holistic methods is Eigenfaces (Turk and Pentland, 1991). In the feature-based methods, local features such as eyes, nose and mouth are first extracted and their locations and local statistics are transferred to a structural classifier. A big challenge for feature extraction methods is feature restoration; this is when the system tries to regain features that are invisible due to large variations, for instance, the head position, or the distance of the captured face. The hybrid face recognition systems are a combination of both holistic and feature extraction methods. Usually 3D Images are used in hybrid methods. The image of a person's face is captured in 3D, thus the system can obtain the curves of the eye

sockets, for example, or the shapes of the chin or forehead. The 3D face recognition systems are usually passed through the detection, position, measurement, representation, and Matching (Parmar and Mehta, 2013).

2.3.1 Face Recognition Applications

Face recognition is useful in different applications and it is requested in many fields. Some of the face recognition applications and examples on them are highlighted in the following points:

A. Face Identification: Face recognition systems establish the existence of an authorized person not only just checking the identification (ID). Example on this application, the face identification is used to prevent a person how takes more than one ID from voting multiple times in a nationwide voter registration system. If the face recognition system detects the face of the person and there was a match with a previously voted person's face then the person will not be allowed to vote again (Parmar and Mehta, 2013).

B. Access Control: In the access control applications, such as office access or computer logon, the number of people that need to be recognized is relatively small. The face pictures are also captures under natural conditions thus the face recognition system of this application can achieve high accuracy. Example on this application, a face recognition system is continuously monitor who is in front of a computer. The user is allowed to leave the terminal without closing files and logging out. When the user leaves for some time, a screen saver covers up the work and disables the mouse & keyboard. When the user comes back, the system recognizes the user then screen saver clears. Any other unauthorized user cannot view the computer.

C. Security: For security purposes, the airport protection systems used face recognition technology at many airports around the world. The system is designed to alert airport public safety officers whenever an individual matching the appearance of a known terrorist suspect enters the airport's security checkpoint. Anyone recognized by the system would have further investigative processes by public safety officers.

D. Image database investigations: Searching image databases of licensed drivers, benefit recipients, missing children, immigrants, and police bookings.

E. General identity verification: Electoral registration, banking, electronic commerce, identifying newborns, national IDs, passports, employee IDs.

2.3.2 Face Detection and Tracking

Human face is a dynamic object that comes in many forms and colors (Muller et al., 2004). Facial recognition is not possible if the face is not separated from the background. The process of finding the face and identifying its position in the image called face detection and tracking. Although many different algorithms exist to perform face detection, each has its own pros and cons. Some use flesh tones, some use contours, and other are even more complex involving templates, neural networks, or filters.

These algorithms suffer from the high computational cost (Wilson and Fernandez, 2006). An image is only a collection of color and/or light intensity values. Analyzing these pixels for face detection is time consuming and difficult to accomplish because of the wide variations of shape and pigmentation within a human face. Viola and Jones (2000) presented an algorithm to detect the human faces, which is based on Haar-like features and instead of pixels. Many face recognition algorithms have adopted the Haar

Cascade algorithm for face detection step such as (Wilson and Fernandez, 2006; Shuzhi et al., 2008; Nishimura and Kuroda, 2010; Zakaria and Suandi, 2011).

In this research, Haar Cascade method is adopted for the face detection step, therefore, some basic background information about this method are presented in the following subsections:

A. Haar Cascade Classifier

Haar classifier features are found using the change in contrast values between adjacent rectangular groups of pixels. The light and dark areas are decided according to the contrast variances between the pixel groups. Two or three adjacent groups with a relative contrast variance form a Haar-like feature. In order to scale the Haar features one can easily increase or decrease the size of the pixel group that are tested. This allows features to be used to detect objects of various sizes (Viola and Jones, 2004).

B. Integral Image

The simple rectangular features of an image are calculated using an intermediate representation of an image, called the integral image (Viola and Jones, 2004). The common Haar features are as shown in Figure 2.2.

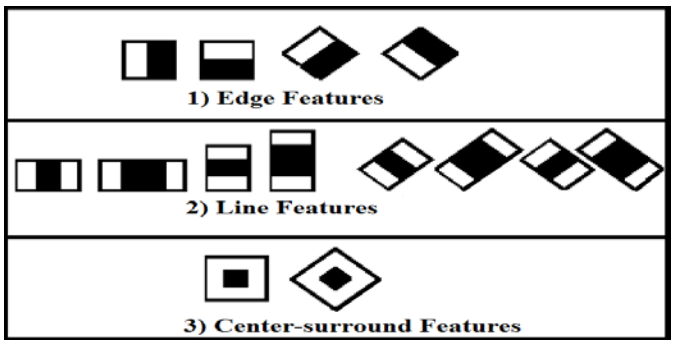

Figure 2.2: Common Haar Features (Wilson and Fernandez, 2006)

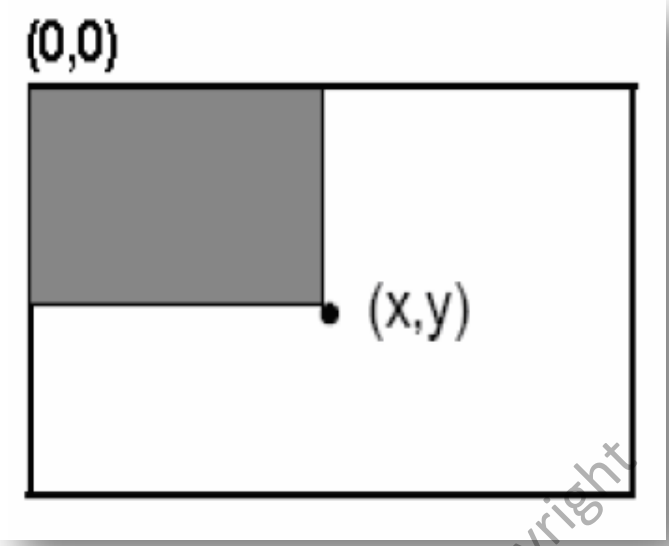

Figure 2.3: Summed Area of Integral Image (Wilson and Fernandez, 2006)

The integral image is an array containing the sums of the pixels' intensity values located directly to the left of a pixel and directly above the pixel at location (x, y) inclusive as illustrated in Figure 2.3 (Wilson and Fernandez, 2006). A feature can be calculated using the appropriate integral image and taking the difference between six to eight array elements, which form two or three connected rectangles. Thus calculating a feature is extremely fast and efficient.

C. Cascaded Classifiers

Although calculating a feature is efficient and fast, calculating all 180,000 features contained within a 24 × 24 sub-image is impractical (Viola and Jones, 2004; Wilson and Fernandez, 2006). Only some of the calculated features are necessary to determine if a sub-image contains the desired object. In order to eliminate as many sub-images as possible, only a few of the features that define an object are used when analyzing sub-images. The cascading of the classifiers allows only the sub-images with the highest

probability to be analyzed for all Haar-features that distinguish an object; in addition, it provides the ability to vary the accuracy of a classifier.

D. Training Classifier for Facial Features

Detecting human facial features, such as the mouth, eyes, and nose require that Haar classifier cascades first be trained. The AdaBoost algorithm and Haar feature algorithms are implemented to train the classifier. Intel has developed an open source library called Open Computer Vision Library (OpenCV) to facilitate the implementation of computer vision related programs. The OpenCV library is designed to be used in conjunction with applications that pertain to the field of HCI, robotics, biometrics, image processing, and other areas where visualization is important and includes an implementation of Haar classifier detection and training (OpenCV Reference Manual, 2001).

2.3.3 Face Identification

One of the face identification methods is the Eigenface method, which is based on linearly projecting the image space to a low dimensional feature space (Turk and Pentland, 1991a, b; Moses and Ullman, 1994). This method uses principal components analysis (PCA) for decreasing the dimensionality; however, PCA retains unwanted variations due to lighting and facial expression (Belhumeur et al., 1997). As stated by Moses et al., “the variations between the images of the same face due to illumination and viewing direction are almost always larger than image variations due to change in face identity” (Moses et al., 1994). Another approach for face recognition that is insensitive to large variations in lighting and facial expressions has been presented in (Belhumeur et al., 1997). This method considers that lighting variability includes not only intensity, but also direction and number of light sources where the same person, with the same facial

expression. and seen from the same viewpoint, can appear dramatically different when light sources illuminate the face from different directions. The face recognition method in (Belhumeur et al., 1997) called fisherface method. It has been proven that the fisherface method has less error rate than the Eigenface methods that have been presented in (Turk and Pentland, 1991a, b; Moses and Ullman, 1994).

The fisherface method has been adopted by many researches in the field of face recognition such as (Lee et al., 2001; Shan, 2002; Heseltine, 2004a; b; Yin, 2005; Zhangi and Ruan, 2010; Li et al., 2012). Research in this thesis will also adopt the fisherface method for face identification process; therefore, some background information about the fisherface method is explained.

In the fisherface algorithm (Belhumeur et al., 1997), the information from the images learning set has been used to implement a reliable method in order to reduce the dimensionality of the feature space. Aiming to better performance than the Linear Subspace method or the Eigenface method, the authors suggested the use of specific linear methods for dimensionality reduction and simple classifiers in the reduced feature space. The Fisher's Linear Discriminant (FLD) method has been applied, which has the ability to "shape" the scatter in order to make it more reliable for classification. This method makes the ratio of the between-class scatter and the within-class scatter is maximized.

If the between-class scatter matrix and the within-class scatter matrix are defined as shown in Equation (1) and Equation (2) (Belhumeur et al., 1997):

$$S_{B=\sum_{i=1}^{c} N_i(\mu_i-\mu)(\mu_i-\mu)^T} \quad (1)$$

$$S_{W=} \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \quad (2)$$

Where μ_i is the mean image of class X_i, and N_i is the number of samples in class X_i. If S_W is nonsingular, the optimal projection W_{opt} shown in Equation (3) is chosen as the matrix with orthonormal columns, which maximizes the ratio of the determinant of the between-class scatter matrix of the projected samples to the determinant of the within-class scatter matrix of the projected samples.

$$W_{opt=} \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|} = [w_1, w_2, \dots, w_m] \quad (3)$$

Where $\{w_i | i = 1, 2, \dots, m\}$ is the set of generalized eigenvectors of S_B and S_W corresponding to the m largest generalized eigenvalues $\{\lambda_i | i = 1, 2, \dots, m\}$, i.e.,

$$S_B w_i = \lambda_i S_W w_i, \quad i=1, 2,\dots, m$$

There are at most c - 1 nonzero generalized eigenvalues, therefore, the upper bound on m is c - 1, where c is the number of classes.

In order to overcome the complication of a singular S_W, (Belhumeur et al., 1997) proposed a method, which they called Fisherfaces. The method projected the image set to a lower dimensional space so that the resulting within-class scatter matrix S_W is nonsingular. The W_{opt} is calculated as follows:

$$W_{opt}^T = W_{fld}^T = W_{pca}^T \quad (4)$$

Where

$$W_{pca=} \arg\max_W |W^T S_T W| \quad (5)$$

$$W_{fld=} \arg\max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|} \quad (6)$$

The optimization for W_{pca} is performed over n × (N - c) matrices with orthonormal columns, while the optimization for W_{fld} is performed over (N - c) × m matrices with orthonormal columns. In computing W_{pca}, the smallest c - 1 principal components have been neglected.

2.3.4 Embedded Systems for Face Recognition

As mentioned before, the embedded system for face recognition has many advantages and it attracted attention. In (Lee et al., 2005), an embedded system for recognizing the face and the facial expressions has been proposed to be used for the Human-Robot interaction. The AdaBoost algorithm has been applied for the face detection process and the Principal Component Analysis (PCA) has been applied for the face recognition. For recognizing the facial expressions, the Gabor wavelets have been combined with the Enhanced Fisher Model (EFM). In (Sun et al., 2007), an embedded system using ARM9 microprocessor and the hidden Markov model (HMM) has been presented. The system uses a CMOS digital imaging sensor OV7640, a S3C2410A processor and the Linux Operation System for the image processing. In (Ahmad, 2010), an embedded system for face identification based on iris detection has been presented. In this system, the x86 processor TS-5500 SBC and the iris detection algorithm have been used. The recognition system depends on the template matching.

In (Chen et al., 2010), a real-time face detection and recognition system has been presented. In the visual system, a multi-information method consisting of an Adaboost algorithm, and color information has been used for the face detection part. An embedded hidden Markov model (EHMM) is presented, using a 2-DCT feature vector as the observation vector, to recognize the detected faces. In (Shuhaizar, 2010), an embedded operating system optimization for face recognition systems has been presented. This system concentrates on the compiler optimizations to develop the embedded operating system and to study the effect of such implementation on the performance of the system. Focusing on face recognition systems implemented on embedded Single Board Computers, optimization effects to an embedded operating system are studied through software and hardware perspective.

2.4 Summary

This chapter reviews embedded systems and their specifications. The overview of embedded systems includes the background information about the SBC, embedded OS, and embedded Linux. Then, the face recognition systems, techniques, and applications are reviewed. Finally, a brief review of the face recognition systems using embedded systems has been presented.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter explains in details the design of embedded system for face recognition application using Raspberry Pi (model A). The design consists of two main parts that are the hardware and software parts. For short, the proposed system will be named as (FRRP), which refers to Face Recognition using Raspberry Pi. The design and implementation of FRRP using ARM1172 and GNU/Linux is presented. The settings and configurations of the hardware and the OS are described. This chapter provides explanation on how to install and manage all the packages and the libraries that are required to develop and execute the codes.

3.2 FRRP Overview

An overall architecture of the proposed FRRP is shown in Figure 3.1. Next sections explain the details about the hardware and the software of the face recognition system.

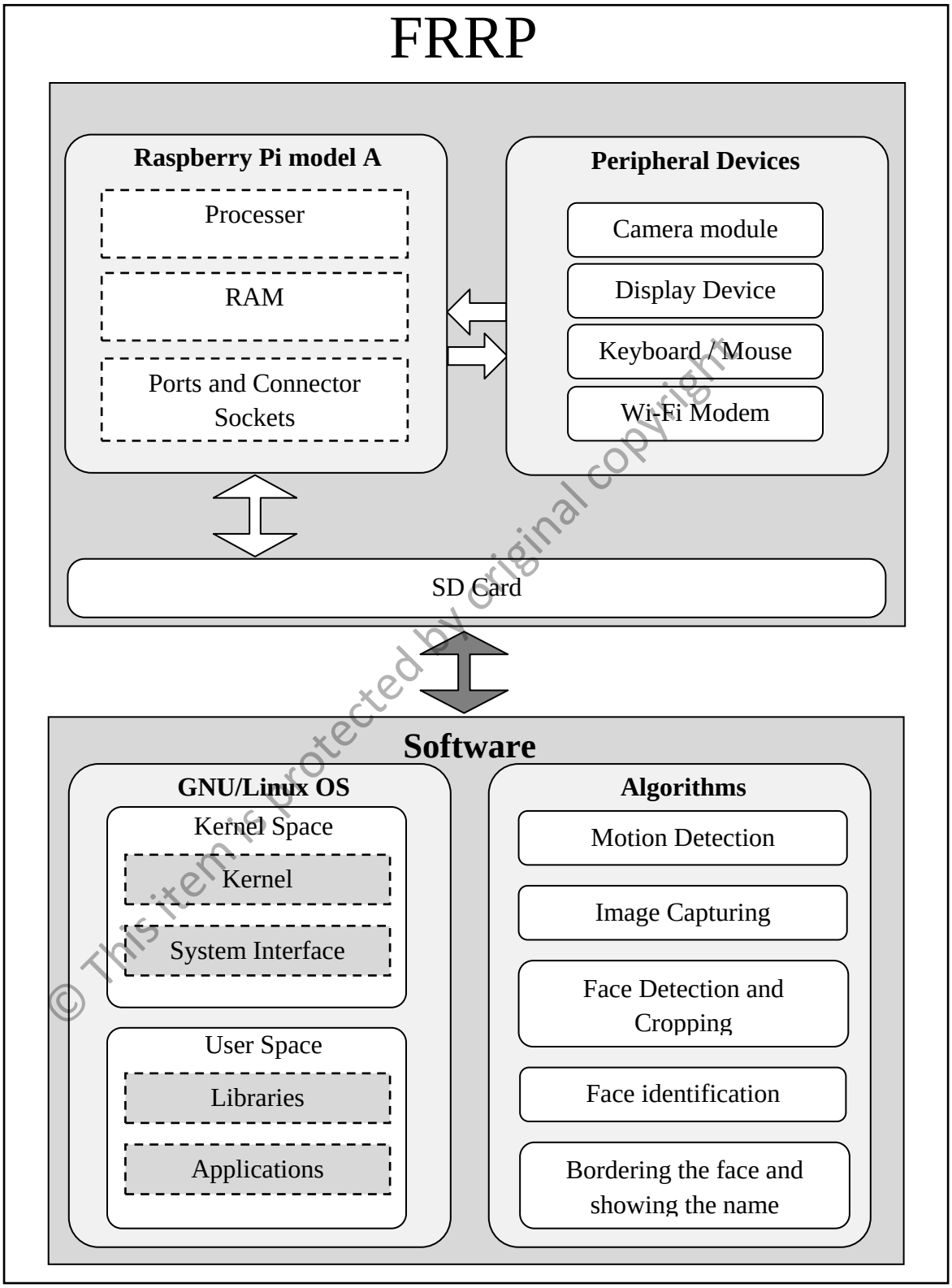

Figure 3.1**:** Overall FRRP Architecture

3.3 FRRP Hardware Description

The hardware platform can be described by highlighting the main components and the connection between them. The core of the design is the Raspberry Pi model (A) which contains computer system and input / output interface. Figure 3.2 shows the hardware block diagram of FRRP. The Raspberry Pi (A) supports the Raspberry Pi Camera module by Camera Serial Interface (CSI). The camera connector socket (S5) is a special socket for ribbon cables that allows the camera module to be connected. In addition, the USB port allows the EDiMax wireless adapter, the keyboard, and the mouse to be connected via USB2.0 power hub. The following subsections describe the hardware components.

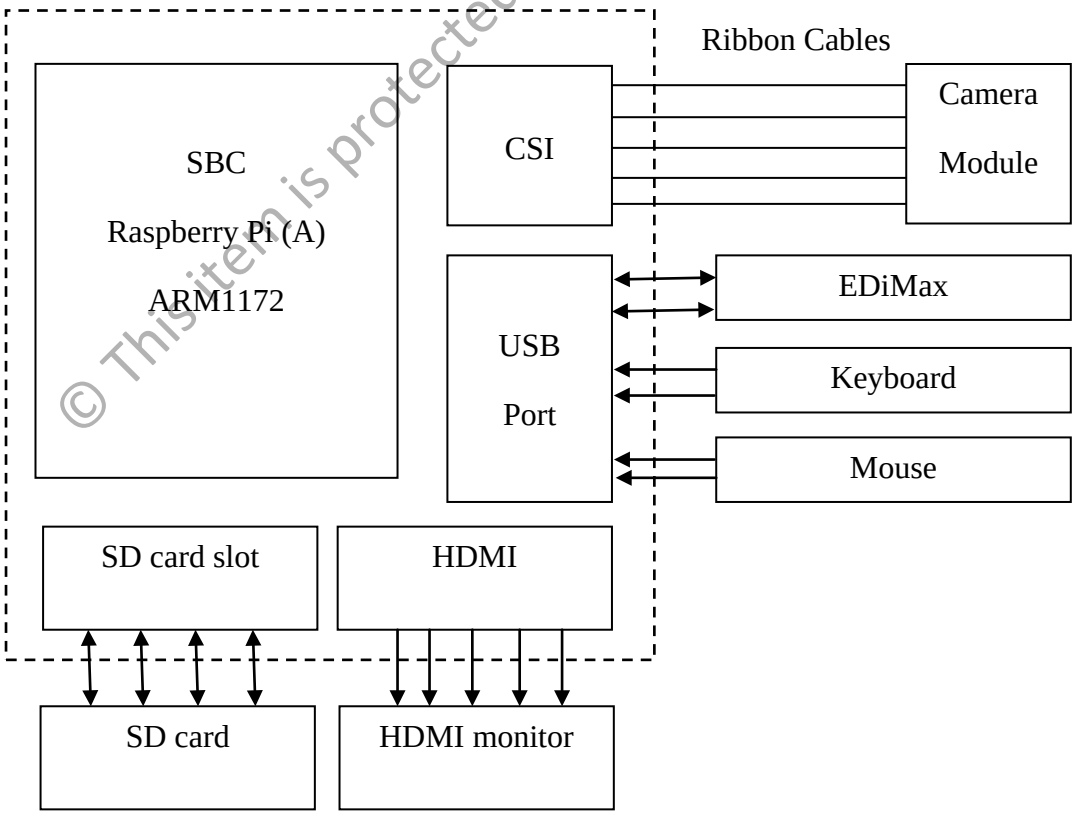

Figure 3.2**:** Block Diagram of the FRRP Hardware

3.3.1 Raspberry Pi model (A) Description

Raspberry Pi model (A) is a compatible SBC based on ARM processor that contains a standard set of the build-on peripheral as shown in Figure 3.3 and its technical features are illustrated in Table 3.1 (element14, 2014).

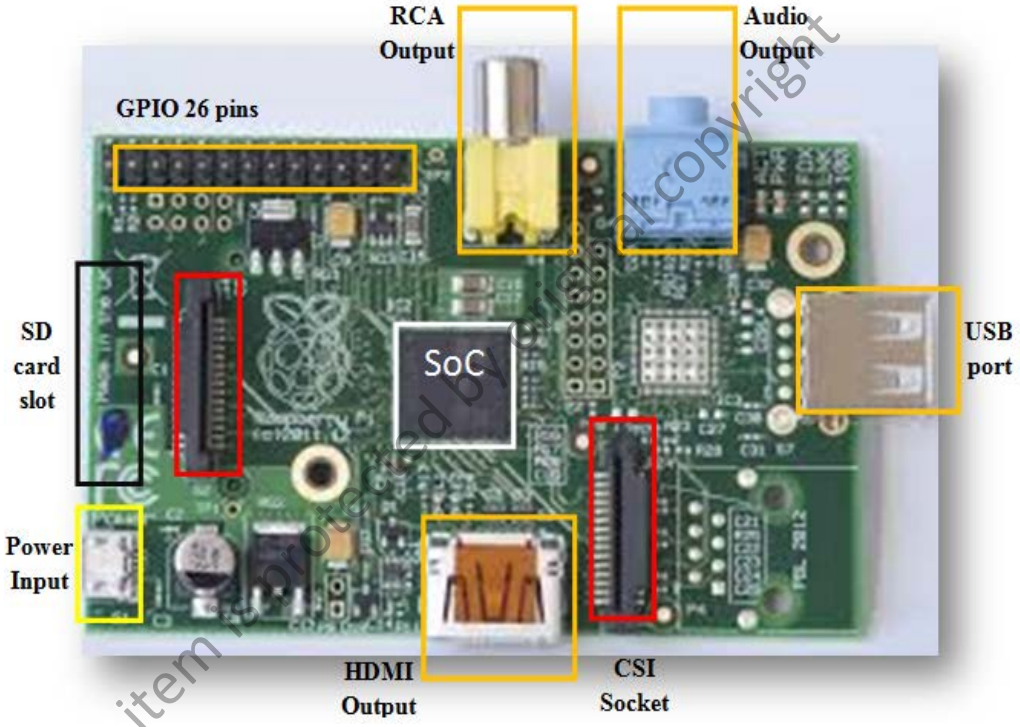

Figure 3.3: Raspberry Pi Model (A)

Table 3.1: Technical Features of the Raspberry Pi model (A)

No.	Components	Description
1	Chip	Broadcom BCM2835 SoC full HD multimedia applications processor
2	CPU	700 MHz Low Power ARM1176JZ-F Applications Processor
3	GPU	Dual Core VideoCore IV®Multimedia Co-Processor
4	Memory	256MB SDRAM
5	USB 2.0	Single USB Connector
6	Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
7	Audio Output	3.5mm jack, HDMI
8	Onboard Storage	SD, MMC, SDIO card slot
9	Serial Interface (S5,S2)	CSI (Camera Serial Interface) DSI (Display Serial Interface)
10	GPIO	26 pins
11	Operating System	Linux
12	Dimensions	8.6cm x 5.4cm x 1.5cm

3.3.2 Peripheral Devices

One of the most important peripherals that need to be inserted in the Raspberry Pi is the storage device (SD) card that stores the OS and related files. The SD card works as a key to the make the Raspberry Pi operates. Switching on the Raspberry Pi will execute a special code called “bootloader”, which reads another code from SD card to start the Raspberry Pi (RPi SD cards, 2014). SD cards come in a range of storage sizes and the producers of the Raspberry Pi recommended using a 4GB device; however, the Raspberry Pi can able to with SD card of size 2GB. In addition to the difference in the size, there are different types of SD cards. Among the different SD cards that are shown in Figure 3.4 (i.e., SDHC (SD High-Capacity), SDSC, SDXC, and SDIO), the best choice is the SDHC (Raspberry Pi: The Essential Manual, 2014), therefore it has been chosen for the proposed FRRP.

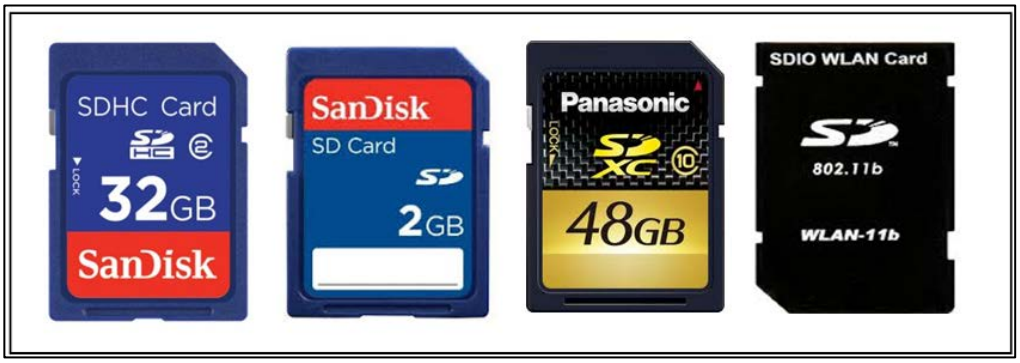

Figure 3.4: SD Card Types

Another peripheral that is connected to the Raspberry Pi is the camera module, which can be used to take HD video, as well as stills photographs. The module has a 5 megapixel with fixed-focus, and supports 1080p30, 720p60 and VGA90 video modes, as well as stills capture. It can be connected via a 15cm ribbon cable to the CSI port on the Raspberry Pi. Figure 3.5 shows the camera module and the ribbon cable. With ever-increasing camera resolutions (measured in megapixels), the amount of data that are. Transferred from the camera module to the processor is also increased. The CSI specification allows the designers to integrate camera to any processor such as that on the Raspberry Pi, which supports the MIPI interface.

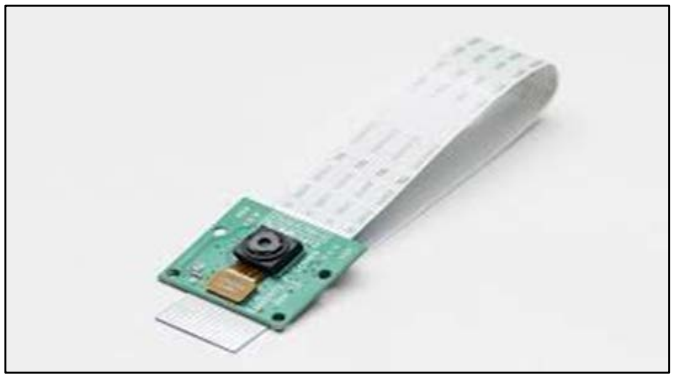

Figure 3.5: Camera Module and Ribbon Cable

For the wireless connection on the Raspberry Pi, a USB wireless adapter should be used. In the proposed FRRP system, the EDiMAX (model: EW-7811Un) is used, which is shown in Figure 3.6. The EDiMAX (model: EW-7811Un) provides higher data

rate of up to 150Mbps, supports maximum range, provides high speed, and it complies with the wireless IEEE802.11b/g/n standards. The EDiMAX (model: EW-7811Un) is fully compatible with Raspberry Pi Operating System (Raspbian).

Figure 3.6: EDiMAX USB Wireless Adapter

3.4 FRRP Software Module

The operating system is a software collection of applications, libraries, and developer tools, plus a program to allocate resources and talk to the hardware, known as a kernel. GNU is typically used with a kernel called Linux (GNU Operating System, 2014). This combination is the GNU/Linux operating system, which has been used by millions. The proposed FRRP software is implemented using GNU/Linux OS. There are some tasks required to achieve employing embedded Linux in a product that are:

- Board support package (BSP): A BSP contains a boot-loader and Linux kernel that are suitable with Raspberry Pi (model A) board. This task is done by using the Raspbian Operating System which based on Debian optimized for the Raspberry Pi hardware

- System integration: Integrating all the components such as boot-loader, Linux kernel libraries, packages, and applications into the embedded system in order to achieve their purposes.

The next subsections discuss Raspbian OS, the installation of the packages and libraries. Then the configurations and settings for SWAP for Raspbian OS, the camera module, and EDiMAX wireless adapter are explained.

3.4.1 Raspbian Operating System

The Raspbian is a free operating system based on Debian optimized for the Raspberry Pi board. This operating system is the set of the basic applications and utilities that make the Raspberry Pi run. However, it is more than a pure OS; it comes with over than 35,000 packages, pre-compiled software bundled. Debian 7.6 was released on July 12th, 2014, which is focused on better support for the Raspberry Pi and it became the official recommended Linux distribution for the Raspberry Pi. The Raspbian is a complete rebuild of Debian Wheezy AMRHF for the ARM11 CPU on the Raspberry Pi. The installation of the Raspbian OS for Raspberry Pi into the SD card using the Graphical Interface is given in Appendix B.

Insert the SD card into a slot then chose the downloaded file (after extracting file) and flash device, click “Write to Device”. When the operation is completed, a message box will appear. Figure 3.7 shows the Image Writer and the message.

Figure 3.7: Image Writer software and the success message.

To install Raspbian OS for Raspberry Pi into the SD card by using the Command Line Interface is given in Appendix B.

After completing the installation process, there are three Partition Disks:

- Partition primary 78.6 MB B f fat32 /rpiboot
- Partition primary 255.9 MB f swap swap
- Partition primary 3.6 GB f ext3

3.4.2 Installing Packages and Libraries

Once login to Raspbian board EDiMAX modem is used to connect the board internet and required packages and libraries are installed. The library is a file containing compiled code from various objects and/or functions. The installation process of the packages and the libraries can be summarized in the following steps:

- **Step 1: Installation of the required Image I/O and video libraries can be found in Appendix C.**
- **Step 3: Installation of Python ver 3.4.1**

Python is a dynamic, interpreted language, and it does not require declaration of the types of variables or parameters. That makes the code flexible and short. The compiling time for the code file is fast in comparison with other languages (Google Developers Python Introduction, 2014). Python 3.4.1 was released on May 18th, 2014. Installing Python ver(3.4.1) on the Raspbian OS by using the Command Line Interface is as follows:

Download the desired Python-3.4.1.tgz file

```
pi@raspberrypi:~$ wget http://www.python.org/ftp
/python/3.4.1/Python-3.4.1.tgz
```

Extract Python-3.4.1.tgz file in the current directory

```
pi@raspberrypi:~$ tar -xzf Python-3.4.1.tgz
```

Configure and Install

```
pi@raspberrypi:~$ cd Python-2.7.6

pi@raspberrypi:~$ ./configure

pi@raspberrypi:~$ make  && sudo make install
```


- **Step 4: Installation OpenCV ver. 2.4.9**

OpenCV is open source computer vision and machine learning software library. It has been built to provide a common infrastructure for applications based on computer vision. The OpenCV has a BSD-license that makes it easy to use and modify the code. Installing the OpenCV ver.2.4.9 by using the Command Line Interface is as follows:

First, install the Essential, Cmake and PKG tools for building OpenCV installation files as shown:

```
pi@raspberrypi:~$ sudo apt-get install pkg-config
```

```
pi@raspberrypi:~$ sudo apt-get install cmake
```

```
pi@raspberrypi:~$ sudo apt-get install build-
essential
```

Download the desired opencv-2.4.9.tar.gz file:

```
pi@raspberrypi:~$  wget https://codeload.github
.com/Itseez/opencv/tar.gz/2.4.9/opencv-
2.4.9.tar.gz
```

Extract tar file:

```
pi@raspberrypi:~$ tar -xzf opencv-2.4.9.tar.gz
```

Create a FRG directory, Run configure file and make install:

```
pi@raspberrypi:~$ mkdir FRG

pi@raspberrypi:~$ cd FRG

pi@raspberrypi~/FRG$ make && sudo make install
```


3.5 Configuration and Setting

This section contains the three parts for the configuration FRRP, which are the SWAP for Raspbian OS, the camera module, and EDiMAX wireless adapter.

3.5.1 Size of SWAP Partition

Swap space under Raspbian or any GNU/Linux operating systems is a virtual memory. This means that if the system runs out of physical memory, then it will transfer some of the lesser used data in memory to this specific space. In the design of the embedded systems, memory allocation is among the main challenges for the designers.

As shown in Table 3.1, the Pi's memory is 256MB but the actual memory size is 186MB. This limitation together with the face recognition algorithm will result in a very slow performance of the system. To reduce this lack of memory size problem in the proposed FRRP, the SWAP file size has been expanded using the following commands. Figure 3.8 illustrates changing the size of the SWAP file from 100MB to 512MB.

```
pi@raspberrypi:~$ sudo su
```

```
root@raspberrypi:/home/pi# free -htl
```

```
root@raspberrypi:/home/pi# nano /etc/dphys-
swapfile
```

Chang CONF_SWAPSIZE=100 **to** CONF_SWAPSIZE=512

```
root@raspberrypi: dphys-swapfile setup
```

```
root@raspberrypi: dphys-swapfile swapon
```

```
root@raspberrypi:/home/pi# free -htl
```


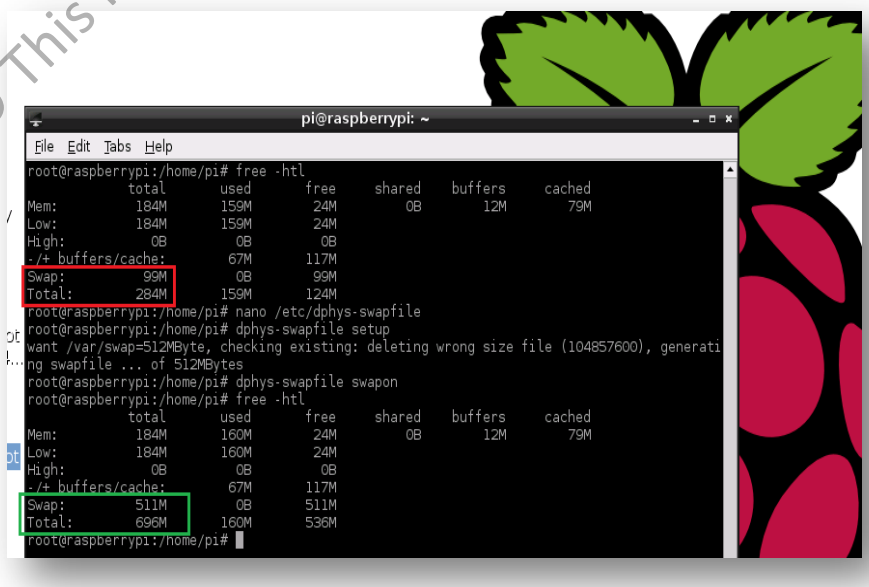

Figure 3.8: Changing The Size Of The SWAP File

3.5.2 Setting Camera Module

The camera module attaches to the Raspberry via a ribbon cable. The ribbon cable needs to be attached to the camera module PCB and the CSI socket. To connect the ribbon cable correctly, the blue label on the ribbon cable should be facing away from the camera PCB and it should be facing towards the place of the Ethernet connection. To make the camera module ready to work, the following instructions must be executed in the command line:

- Update and Upgrade packages

These two processes are necessary, because the camera enable option does not appear until the upgrade process complete. The upgrade process take long time depend on the internet speed. It is download around 224 MB.

```
root@raspberrypi:/home/pi# apt-get update
root@raspberrypi:/home/pi# apt-get upgrade
```

- Reboot the Raspberry Pi.
- After login to the system, call the Raspberry Pi configuration. Figure 3.9 illustrates the selection process of the options.

```
pi@raspberrypi:~$ sudo raspi -config
```

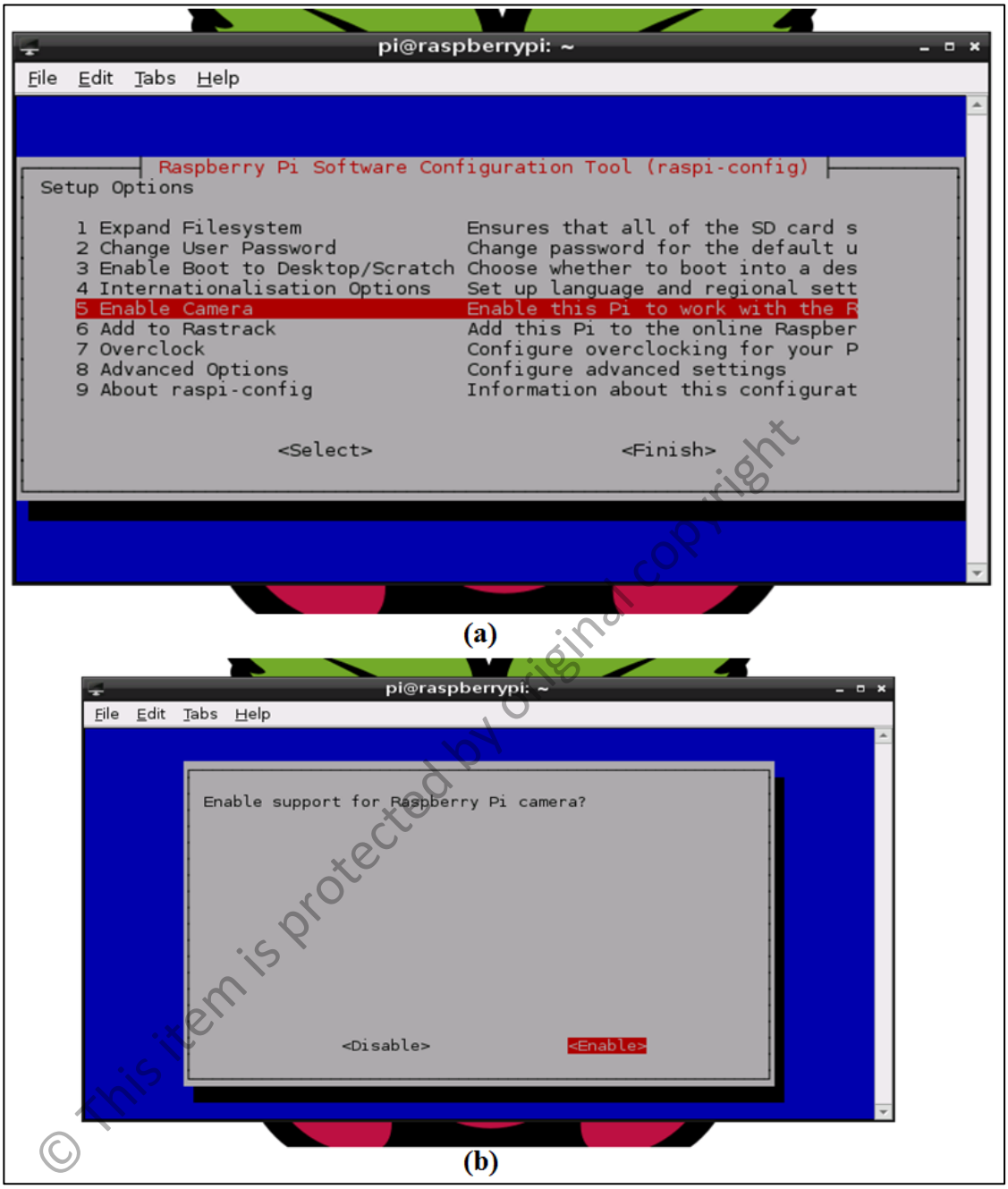

Figure 3.9: The Selected Options To Activate The Camera

3.5.3 EDiMax Configuration

Step1: List all devices connected with the Raspberry Pi via USB port. Figure 3.10 shows this process.

```
pi@raspberrypi:~$ lsusb
```

```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 005: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 006: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 007: ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un 802.11n Wi
reless Adapter [Realtek RTL8188CUS]
pi@raspberrypi ~ $
```

Figure 3.10: Listing All USB Devices

Step: Open the wpa_gui application. Then scan to the specific network and setup all parameters requirement. Figure 3.11 shows wpa_gui application.

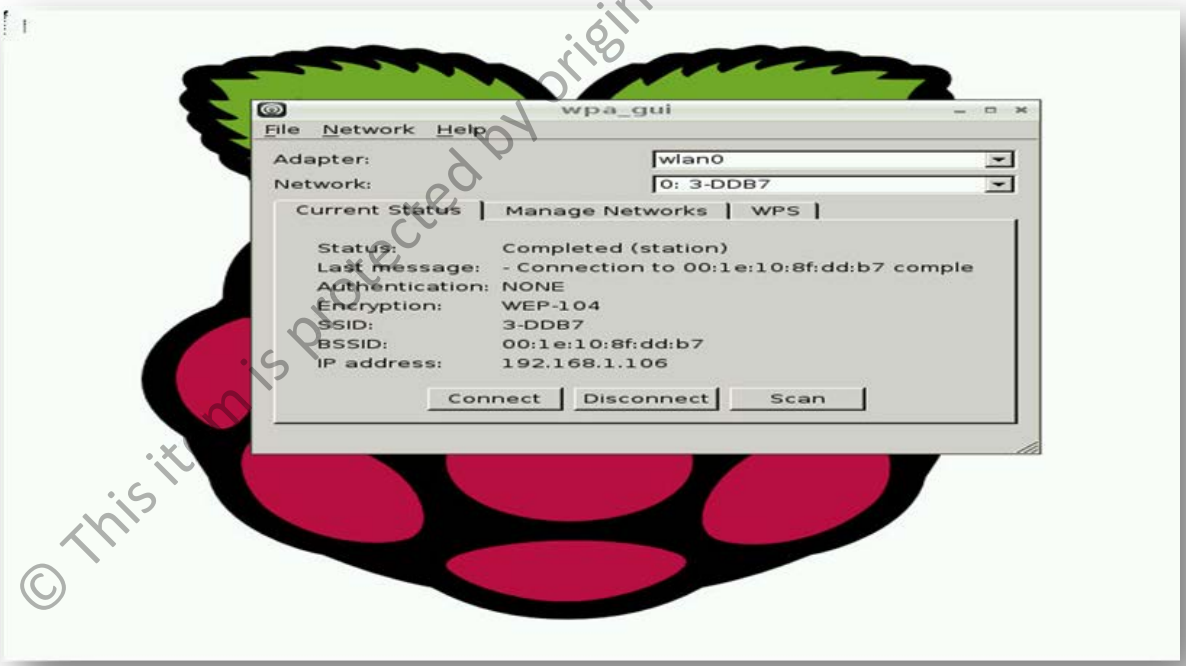

Figure 3.11: WPA Graphical User Interfac

3.6 Summary

In this chapter, the proposed Face Recognition System using Raspberry Pi (FRRP) is explained. The hardware description of FRRP includes the detailed information about Raspberry Pi model (A) and the peripheral devices. The software description of FRRP discusses Raspbian OS, the installation of the packages and libraries. Then the configurations and settings for SWAP for Raspbian OS, the camera module, finally EDiMAX wireless adapter have been explained.

CHAPTER 4

SOFTWARE MODULES DEVELOPMENT

4.1 Introduction

The proposed system is developed on embedded Raspbian based Raspberry Pi (Model A). This chapter describes the software development for face recognition application. The face recognition composes of three main modules: (a) Capture Module, (b) Learning Module, and (c) Recognition Module.

4.2 Capture Module

To prepare the database of the faces' images in order to be used later for the learning process, this module captures and saves the images of the face only. Figure 4.1 shows the flowchart of the Capture Module. In this module, the pre-processing includes the following five main steps:

1. Capture the image.
2. Convert the image to grayscale.
3. Find the location of the face.
4. Crop the face and resize.
5. Save the images of the face and display the captured image.

The details of the Capture Module are as follows:

- Input person's name to create sub-folder inside DataSet directory.

```
directory = sys.argv[1]
#Create Own Directory With DataSet Folder and Subject Folder
if not os.path.exists("DataSet/"+ directory):
      os.mkdir("DataSet/"+ directory)
```

• Load all the needed libraries such as (sys, os, cv2), load Haarcascade XML file, and define all the parameters like (FC).

```
# Load and create object FaceFeature for Face Detection
FaceFeatuer = cv2.CascadeClassifier('face2.xml')
```

• Initialize the Raspberry Pi Camera Module.

• Capture the Frame with resolution 320×240 pixels.

• Convert the captured image from RGB to Grayscale.

```
# Convert to GRAY
Framegray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

• Find a face in the frame by using Cascade Classifier object.

```
faces = FaceFeatuer.detectMultiScale(frame)
```

• If any face is detected in the frame, read the coordinates (x, y) and the dimensions (h, w). Then, crop the face from the frame and resize the image of the cropped face. The image after resizing has a resolution 90×112 pixels.

```
Img = graysave[y:y+h,x:x+w]
img = cv2.resize(img, (90, 112), interpolation =
cv2.INTER_CUBIC)
```

• If a face detected in the frame, then draw a border on the face depending on the new coordinate and dimensions without cropping it. Otherwise, draw a full circle that has the same center of the frame.

• The FC counts the number of the saved images and terminates the module when reaching 20 images.

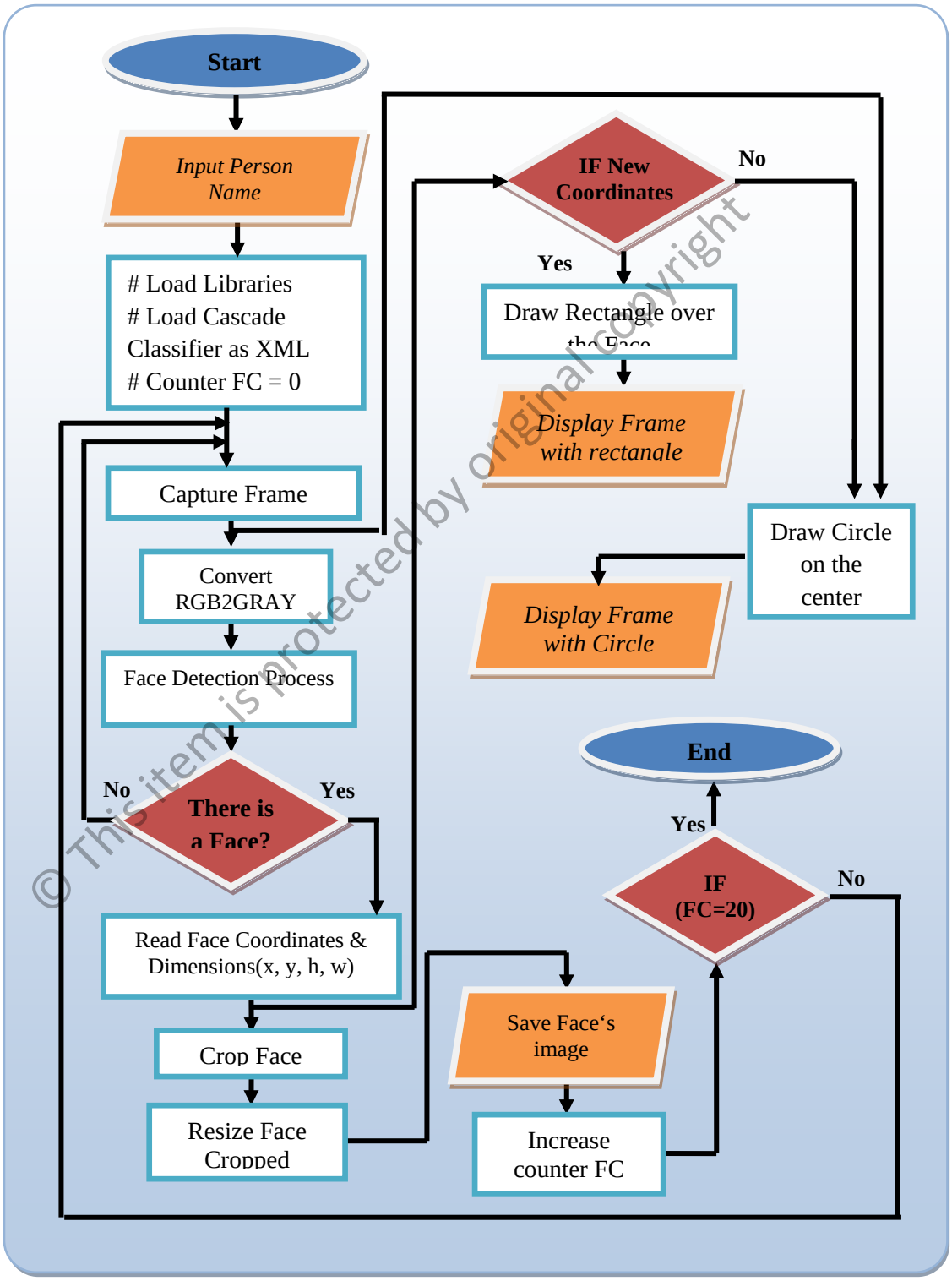

Figure 4.1: Flowchart For Capture Module

4.3 Learning Module

This module is responsible of computing Fisherface training class for groups of facial images with identity by adopting the algorithm from (Belhumeur et al., 1997), which is explained in Detection. This module includes the following: (a) Construct image Matrix *Function A* and (b) Create Predictable Model. Figure 4.2 shows the flowchart of this Module.

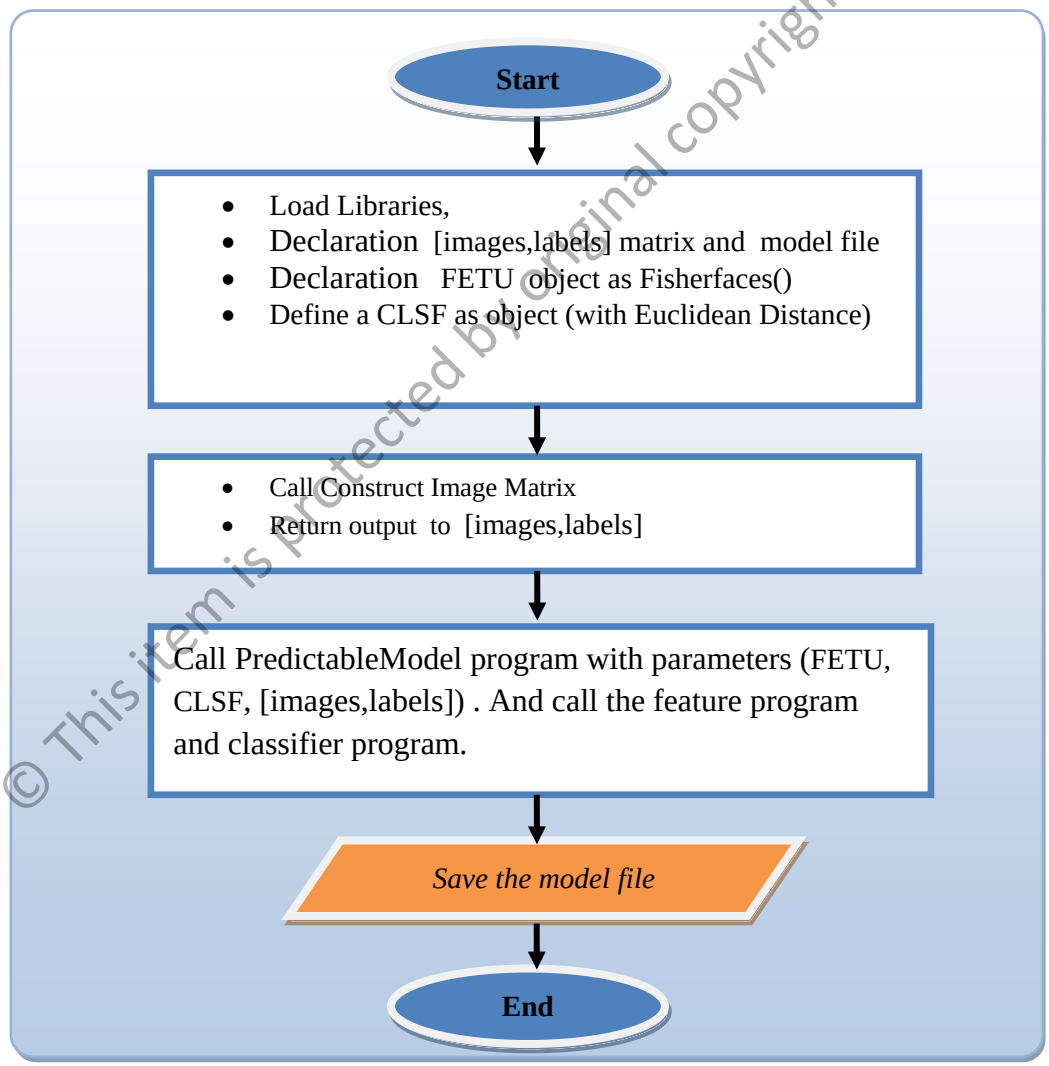

Figure 4.2: Flowchart For Learning Module

4.3.1 Images Matrix and Folder Names Matrix

This part is on reading images, reading subject (label), and creating image matrix. The input for the module is directory of the DataSet folder (Path) which contains at least two subfolder (i.e., at least two groups of images) for different persons. However, this subfolder has 20 face's images (depends on FC) for one subject (person). The output returns to Learning Module. Figure 4.3 shows the flowchart of the Construction Module.

The details of the Construct the Images Matrix (Function (A)) are as follows:

- Load all the needed libraries such as (sys, os, PIL.Image), and define all parameters like (C, X, Y).
- Read all sub-directory names in the input directory.

```
for dirname, dirnames, filenames in os.walk(path):
  for subdirname in dirnames:
    subject_path = os.path.join(dirname,  subdirname)
```

- Read the file name in each sub-directory and open it. Then check the size of the image. If *Yes* go to step A if *No* go to step B.

```
for filename in os.listdir(subject_path):
   im = Image.open(os.path.join(subject_path, filename))
      if (sz is not None):
```

  - Step A. Append matrix X by image has been read as array (8-bits) and append matrix Y by C value after that increase the C by add one.
  - Step B. Show the Error message and continue the loop.

- Function (B) in the Construct module (shown in Figure 4.3) is used for reading the label or subject, which reads the *folder_names* matrix, reads all sub-directory,

and appends *folder_names* matrix by the name of each sub-folder (the name of folder indicates to person's name).

```
    folder_names = []
    for dirname, dirnames, filenames in os.walk(path):
        for subdirname in dirnames:
            folder_names.append(subdirname)
    return folder_names
```

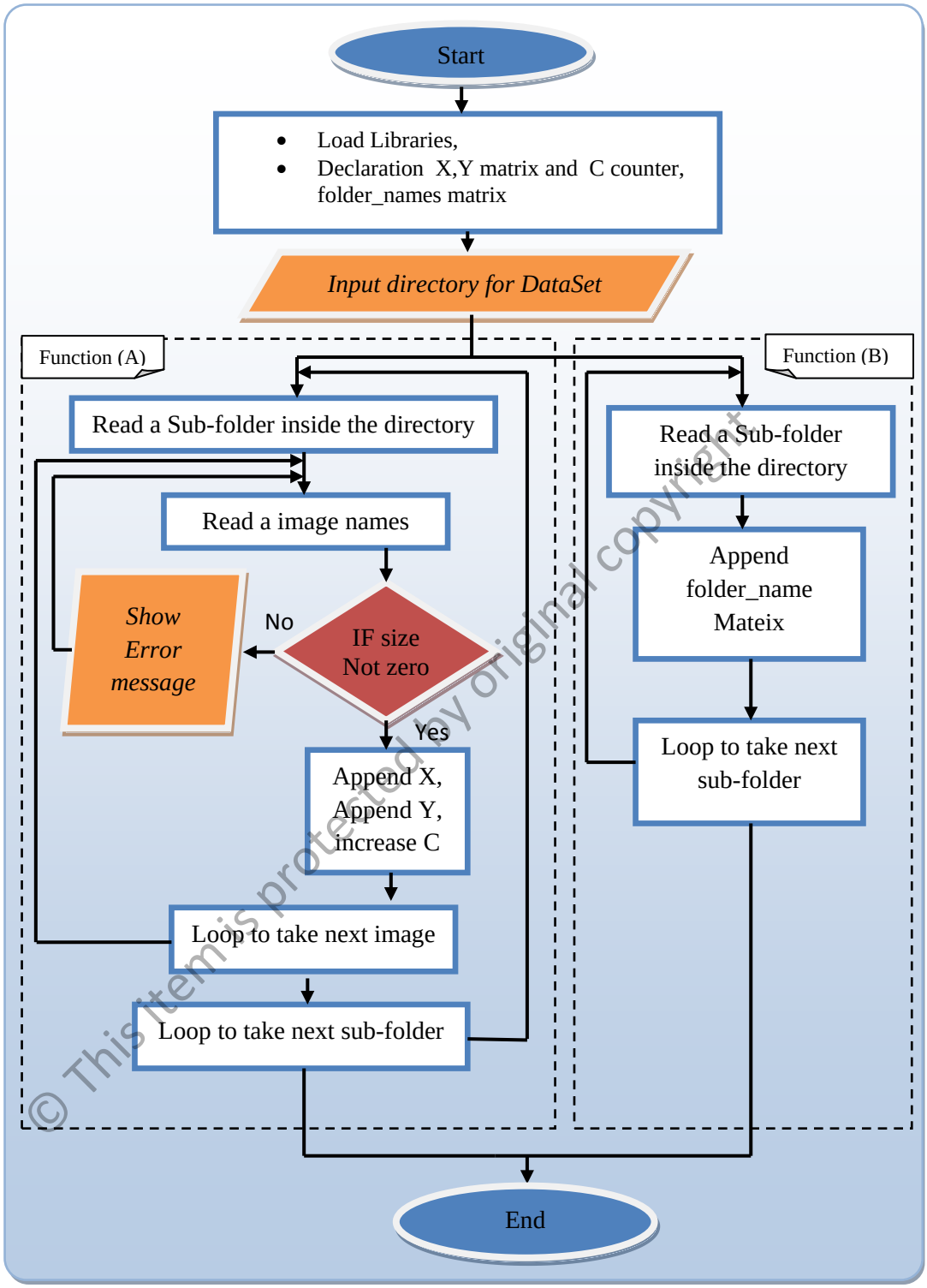

Figure 4.3: Flowchart For Construct Module

4.3.2 Create Predictable Model

To create Predictable Model, it is required to implement the creator and call the PCA and LDA classes by following these steps:

- Load all the needed libraries such as (numpy as np)
- Turn into numpy representation for X and Y matrixes.

```
Xc = asColumnMatrix(X)
yc = np.asarray(Y)
```

- Gather some statistics about the dataset like unique elements of an array and length of dataset.

```
n = len(yc)
c = len(np.unique(yc))
```

- Define features to be extracted by using (Class PCA and Class LDA)

```
pca = PCA(num_components = (n-c))
lda = LDA(num_components = self._num_components)
```

- Define the Fisherfaces as chained feature of PCA followed by LDA

```
model = ChainOperator(pca,lda)
```

- Compute the chained model then calculate both decompositions

```
model = ChainOperator(pca,lda)
```

4.3.2.1 Class PCA

- Set a valid number of components

```
      if self._num_components <= 0 or (self._num_components
      > XC.shape[1]-1):

            self._num_components = XC.shape[1]-1
```

- Center dataset

```
        self._mean = XC.mean(axis=1).reshape(-1,1)

        XC = XC - self._mean
```

- Sort eigenvectors by eigenvalues in descending order.

```
self._eigenvectors, self._eigenvalues, variances =
np.linalg.svd(XC, full_matrices=False)

idx = np.argsort(-self._eigenvalues)

self._eigenvalues, self._eigenvectors =
self._eigenvalues[idx], self._eigenvectors[:,idx]
```

- Use only num_components.

```
self._eigenvectors =
self._eigenvectors[0:,0:self._num_components].copy()

self._eigenvalues =
self._eigenvalues[0:self._num_components].copy()
```

- Turn singular values into eigenvalues

```
self._eigenvalues = np.power(self._eigenvalues,2) /
XC.shape[1]
```

- Get the features from the given data

```
features = []
        for x in X:
            xp = self.project(x.reshape(-1,1))
            features.append(xp)
        return features
```

- Finally, some Functions and properties are used

```
    def project(self, X):
        X = X - self._mean
        return np.dot(self._eigenvectors.T, X)
    @property
    def num_components(self):
        return self._num_components
    @property
    def eigenvalues(self):
        return self._eigenvalues
    @property
    def eigenvectors(self):
        return self._eigenvectors
```


4.3.2.2 Class LDA

- Calculate dimensions and set a valid number of components

```
 d = XC.shape[0]
 c = len(np.unique(y))
 if self._num_components <= 0:
            self._num_components = c-1
elif self._num_components > (c-1):
            self._num_components = c-1
```

- Calculate total mean for the images matrix

```
meanTotal = XC.mean(axis=1).reshape(-1,1)
```

- Calculate the within and between scatter matrices

```
  Sw = np.zeros((d, d), dtype=np.float32)

  Sb = np.zeros((d, d), dtype=np.float32)

  for i in range(0,c):

      Xi = XC[:,np.where(y==i)[0]]

      meanClass = np.mean(Xi, axis = 1).reshape(-1,1)

      Sw = Sw + np.dot((Xi-meanClass), (Xi-meanClass).T)

      Sb = Sb + Xi.shape[1] * np.dot((meanClass -
meanTotal), (meanClass - meanTotal).T)
```

- Sort eigenvectors by their eigenvalue in descending order

```
self._eigenvalues, self._eigenvectors =
np.linalg.eig(np.linalg.inv(Sw)*Sb)

idx = np.argsort(-self._eigenvalues.real)

self._eigenvalues, self._eigenvectors =
self._eigenvalues[idx], self._eigenvectors[:,idx]
```

- Only store (c-1) non-zero eigenvalues

```
self._eigenvalues =
np.array(self._eigenvalues[0:self._num_components].real,
dtype=np.float32, copy=True)

self._eigenvectors =
np.matrix(self._eigenvectors[0:,0:self._num_components].rea
l, dtype=np.float32, copy=True)
```

- Get the features from the given data.

```
features = []
        for x in X:
            xp = self.project(x.reshape(-1,1))
            features.append(xp)
        return features
```

- Finally, some Functions and properties are used

```
def project(self, X):
        X = X - self._mean
        return np.dot(self._eigenvectors.T, X)
    @property
    def num_components(self):
        return self._num_components

    @property

    def eigenvalues(self):

        return self._eigenvalues

    @property

    def eigenvectors(self):

        return self._eigenvectors
```


4.3.3 Serializing Class

Serializing model is used to implement an algorithm for turning a Python object into a series of bytes. This process is also called "*serializing*" the object. The bytes stream representing the object can then be stored or transmitted, and later reconstructed to create a new object with the same characteristics. In Learning Module, the *savemodel* uses to store the *model* file in the system.

- Load the cPickle as library
- Input a file name and Model to *savemodel* function.

```
def savemodel(file_name, model):
    outputfile = open(filename, 'wb')
    cPickle.dump(model, outputfile)
```

4.4 Recognition Module

After formulating the representation of each face, the last step is to recognize the identities of these faces. Face identification means given a face image, to the system then the system tell who he / she is or the most probable identification (nearest class). To start Face identification Module the system must detect motion. This module includes the following approaches: (a) Motion Detection, (b) Face Detection, (c) Call "*load_model"* function from the Serializing Model, (d) Classifier module and (e) Construct the folder names Matrix "*Function B"*. Figure 4.5 shows the flowchart this Module.

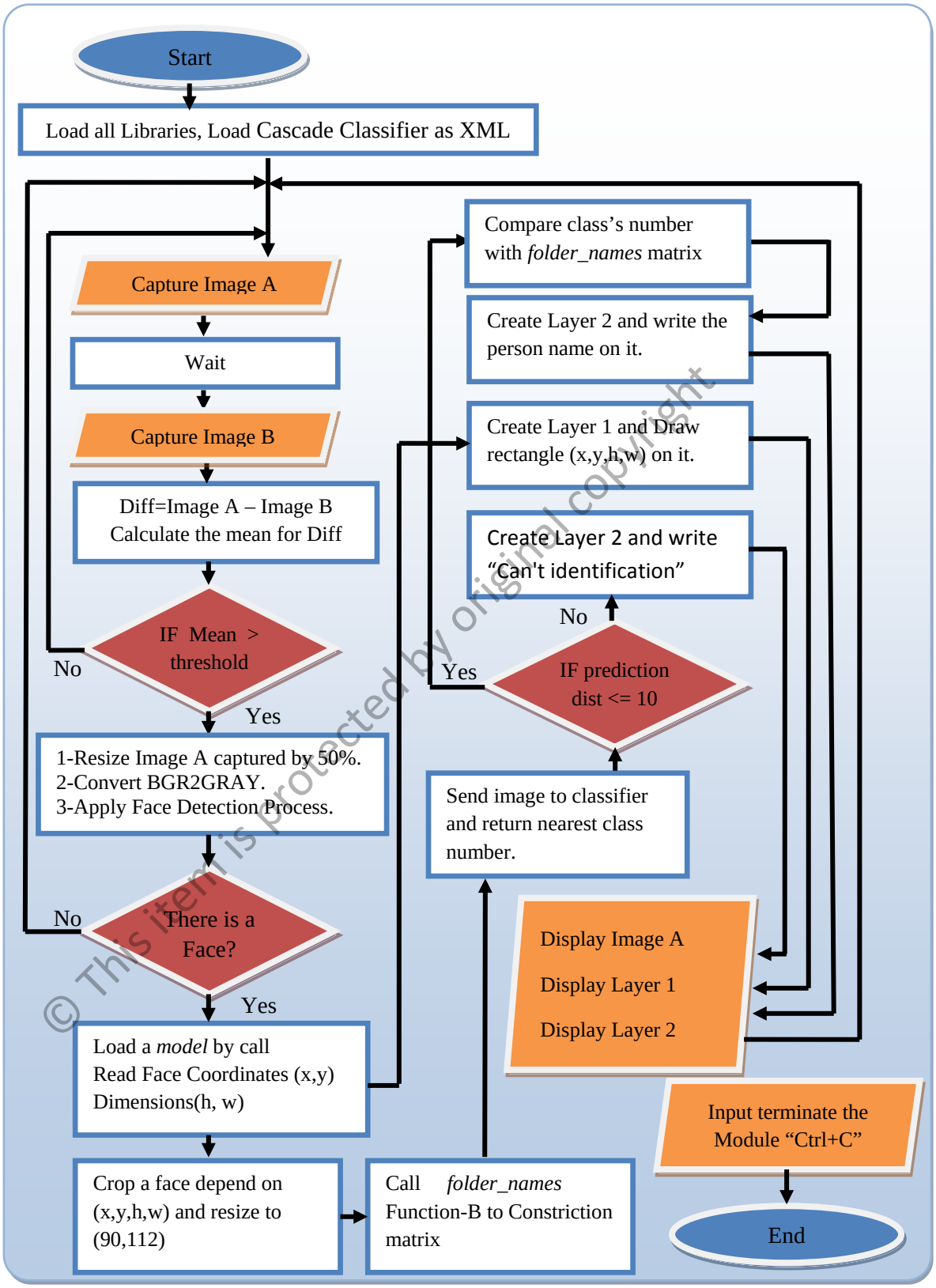

Figure 4.4: Flowchart Of The Recognition Module

4.4.1 Motion Detection

Motion detection is mechanism of detecting a change in the position, the rotation, and/or size of an object or objects relative to its environment, or the change in the surroundings relative to an object. The process of motion detection method consists of the following steps:

- Load all the needed libraries and initialize the Camera Module.
- Capture first image (A) and Wait.

```
ImageA=Cam.read()
Cv2.Wiatkey(100)
```

- Capture second image (B)
- Calculate the difference between two images.
- Calculate the total Mean.

```
mean = diff.mean(axis=1).reshape(-1,1)
```

- If the total mean greater than the threshold continue to Face Detection process else return to the first step.

4.4.2 Face Detection

Face detection process is responsible of detecting the face and cropping it, then sending it to the classification process. In the previous sections (the Capture Module), there are details about face detection that are similar to the face detection process in this module but the only difference is that there is no saving images and no FC counter.

4.4.3 Serializing Model

Call "*load_model"* function from the Serializing Model this time to reconstruct the Model to use it in the Face Identification process.

```
def load_model(filename):
    pkl_file = open(filename, 'rb')
    res = cPickle.load(pkl_file)
```


4.4.4 Classifier module

There are several steps to apply the Classification method. First, the features are extracted from the cropped image, then these features are sent to the Fisheface Class (Call PDA and LDA classes also) to project the image features. Thereafter, call predicted class which it return a first element (this class returns the distances for the first k-Nearest Neighbors). If the distance is accepted, the subject's name (person's name) will appear if not the phrase "Can't identify" will appear.

- Feature extraction for cropped image (faceToRec)

```
ImageArray = np.asarray(faceToRec)
Feature = self.feature.extract(ImageArray)
```

- Call the FasherFace class which in turn calls PDA and LDA classes.

```
pca = PCA(num_components = (n-c))

lda = LDA(num_components = self._num_components)
```

- Call predicted class which calculate the distance between image feature after protection and all subject classes and return the nearest subject.

```
predicted_label = max(hist.iteritems(),
key=op.itemgetter(1))[0]
```

- The Layer 1 and Layer 2 show over Image A.

```
# Draw Rectangle On Layer 1
cv2.rectangle(Layer_1,(x,y),(x+w,y+h),(0,255,255),3)
```

```
# write Person's name On Layer 2
cv2.putText(Layer_2, subject_dictionary[prediction[0]], (x,
y), cv2.FONT_HERSHEY_PLAIN, 2.0, (255, 255, 255), thickness
= 6, lineType=cv2.CV AA)
```

```
# write "Can't identification" On Layer 2
cv2.putText(Layer_2, "Can't identify", (x  , y +
int(h*0.09)), cv2.FONT_HERSHEY_PLAIN, 2.0, (0, 0, 0),
thickness = 2, lineType=cv2.CV AA)
```

4.5 Summary

This chapter focuses on the description of the software development for face recognition application. The algorithms of the three main modules of the face recognition system (i.e., Capture Module, Learning Module, and Recognition Module) are explained. The recognition module description includes the details of different approaches such as motion detection, face detection, serializing model, and the classifier module.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Introduction

This chapter presents overall results and performance evaluation of the system CPU utilization and system RAM utilization for each module. The outputs of three main modules (i.e., Capture Module, Learning Module, and Recognition Module) are presented and analyzed.

5.2 Capture Module

In this section, the output of the Capture Module is presented the performance evaluation are based in terms of RAM usage, CPU usage, and execution time.

5.2.1 Capture Module Output

Capture Module operations starts by scanning the captured image to detect any human face in front view position. The output produced by Capture Module consists of one folder contains 20 images for each subject, includes the coordinates of the (top-left) corner of the face's image, and the dimensions of the face's image (h, w). Once the output of the module is saved, a rectangle is drawn over the detected face. Figure 5.1 shows sample images of faces that are detected by the capture module consist of six subjects. Figure 5.2 shows the execution of the capture module.

Figure 5.1: Samples Of Face's Images Captured And Stored In The System.

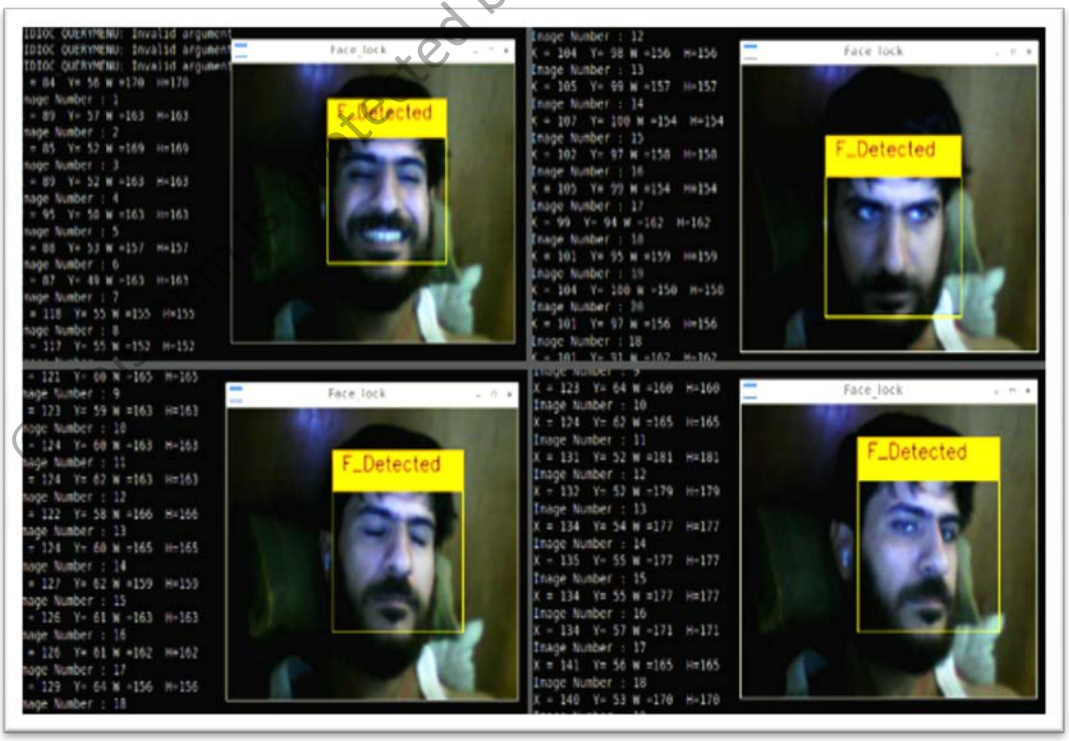

Figure 5.2: Operation Of Capture Module For Different Angle Of Face Detection.

5.2.2 Capture Module Performance

System performance for the SBC refers to the CPU and RAM usage in a period of time when the system is executed. The data are collected by using *ps* command via Linux shell prompt. Figure 5.3 shows the CPU usage percentage when the capture module is executed for four different image resolutions (i.e., 160×120, 320×240, 640×480 and 800×600 pixels). Figure 5.4 shows the RAM usage percentage when the capture module is executed for four image resolutions. The average CPU percentage, average RAM percentage, and the time required to finish different processes for each resolution are shown in Table 5.1.

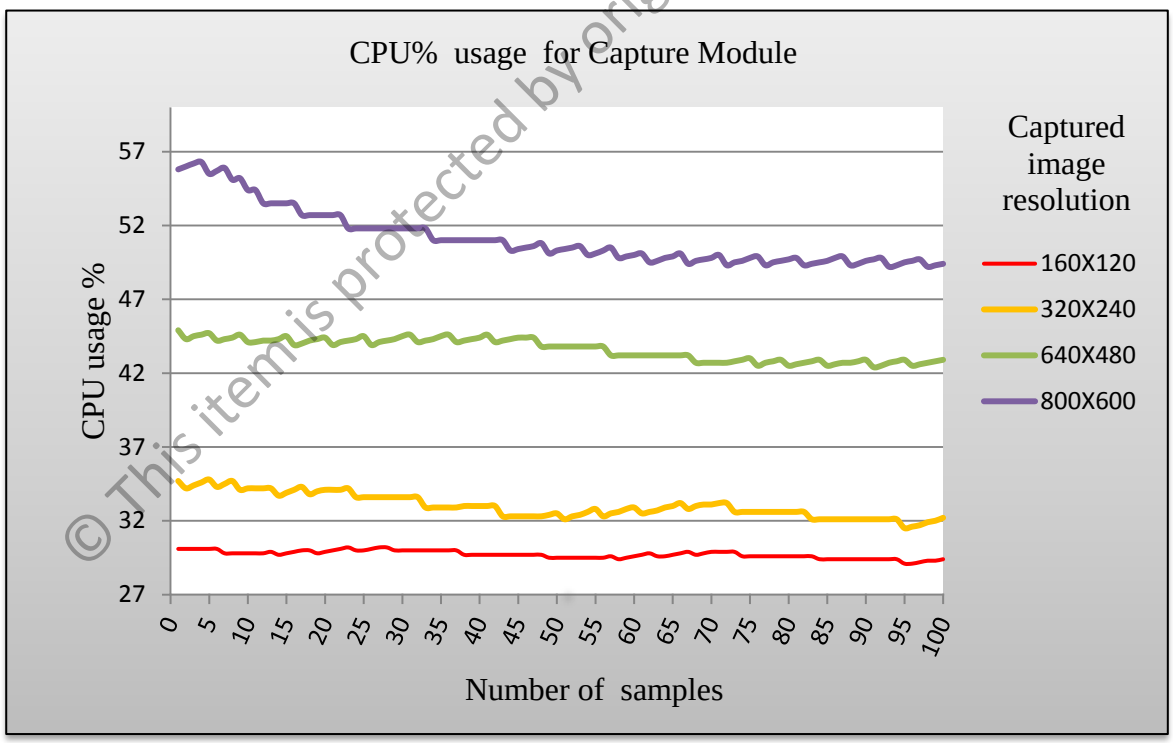

Figure 5.3: CPU Percentage Usage When The Capture Module Executed

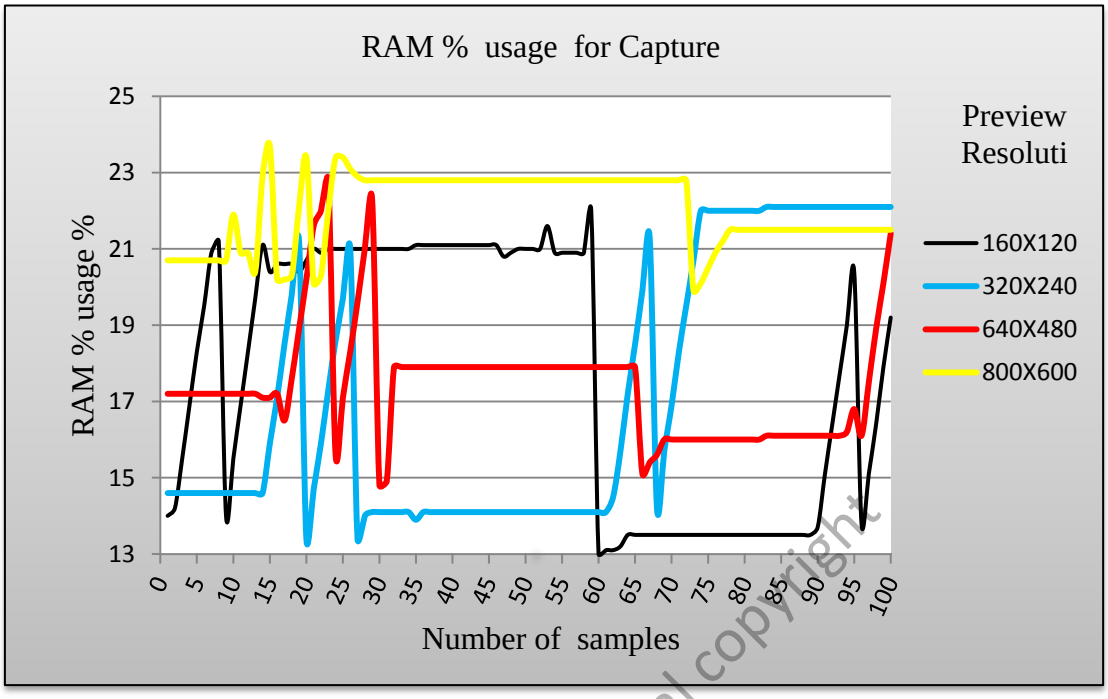

Figure 5.4: RAM Percentage Usage When The Capture Module Executed

Table 5.1: Capture Module Performance And Timing.

Images Resolution	160X120	320X240	640X480	800X600
CPU average percentage	29.72%	33.01%	43.61%	51.13%
RAM average percentage	17.48%	17.75%	18.89%	22.02%
Time to capture a frame in sec.	2.4071538	2.4240889	2.40506720	2.419525862
Time to detect a face in sec.	**1.3328051**	**1.7555449**	**4.21649980**	**6.208329201**
Time to save image in sec.	0.0231230	0.0507159	0.05048799	0.058834076
Total time for each frame to process in sec.	9.0523579	10.076761	12.7662150	17.49780297

5.3 Learning Module

In this section, the output results of the learning module are shown. Then the performance evaluation in terms of RAM usage, CPU usage, and execution time are illustrated.

5.3.1 Learning Module Output

As explained in Chapter 4 (Section 4.3) the Fisherface algorithm is applied to generate the feature space, than these features are sent to serializing class, which generates a file in the system that has been named as "*model*". To present the content of the model file, the classes are reshaped as images. Figure 5.5 shows the reshape images by 16 having eigenvectors for each class.

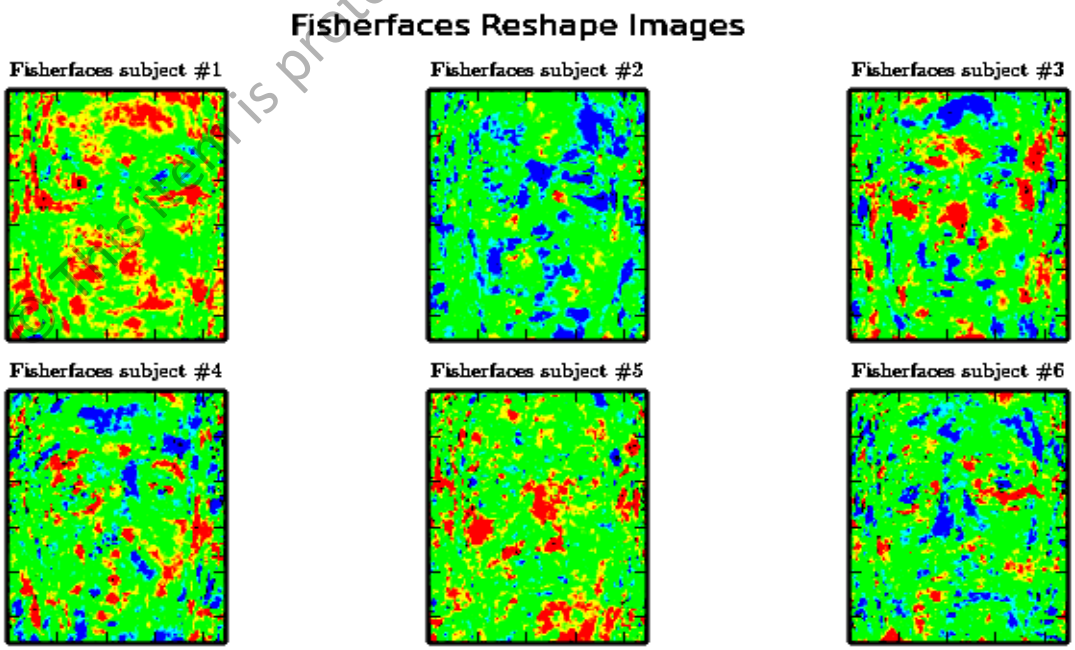

Figure 5.5: Fisherface Reshape Images For Six Classes.

5.3.2 Learning Module Performance

Table 5.2 shows the CPU average percentage and RAM average percentage usage to create the "*model*" file. It also contains the number of classes and images used to create this "*model*" file and the time for reading all images in each subject folder and the total time required to create "*model*" file. Figure 5.6 shows screenshot for system after

Table 5.2: Learning Module performance

Information for	Number
Number of classes	6
Number of images	120
Information for	Size
Total size for subjects images	2,617,344 bytes
Size of model file	1,384,448 bytes
Information for	Time
Time to Read all images in DataSet Directory	2.1231200695 sec
Total time to compute model file	97.3548810482 sec
Time to save model file	1.21821713448 sec
Information for	Average
CPU average percentage usage to create model file	96 %
RAM average percentage usage to create model file	41 %

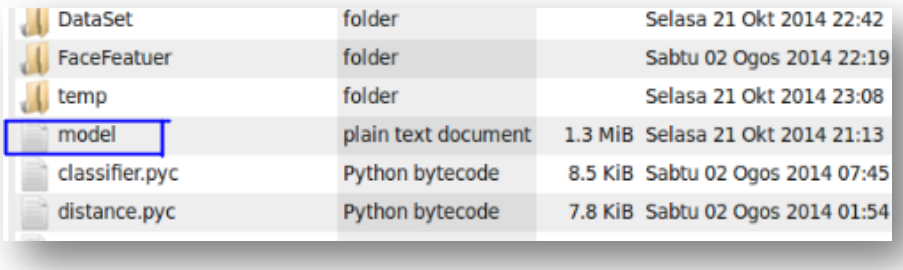

Figure 5.6: Screenshot For System Directory

5.4 Recognition Module

In this section, the results of the recognition module are explained. Then the performance evaluation in terms of RAM usage, CPU usage, and execution time are illustrated.

5.4.1 Recognition Module Output

Output of the recognition module is a preview window for the captured image with two layers. The first layer is for showing a rectangle over the detected face. The second layer is for showing the name of the subject that is matching the detected face or showing “Can't identify” if the subject did not match the saved objects. At the same time a message printed on the command prompt shows the status of the recognition and some information that needs to be shown. Figure 5.7 shows a Recognition module output.

Figure 5.7: Recognition Module Output.

5.4.2 Recognition Module Performance

Performance for recognition module depends on the resolution of the captured image via Raspberry Pi Camera module. When the resolution of the image increased, better view of the image can be obtained but that is at the cost of increasing the time required for face detection in each captured image as tabulated in Table 5.1. To obtain an acceptable balance between the view of the image and the time required for face detection, the resolution of 320×240 is chosen. As shown in Table 5.1, at 320×240 resolution the time to detect the face in the captured image was 1.756 second shorter than that for 640×480 resolution image. On the other hand, at 320×240 resolution, better view can be obtained in comparison with 160×120 image resolution. Figure 5.8 shows the CPU usage percentage and RAM usage percentage for the Recognition module. The time to recognition for one captured image contains face between 0.29s to 0.74s.

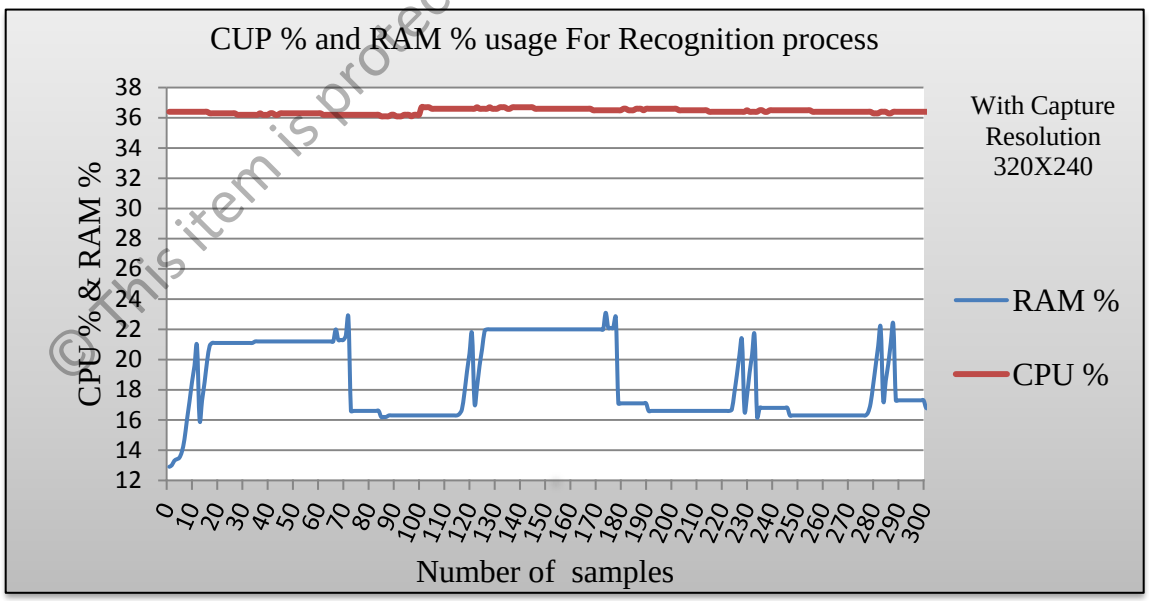

Figure 5.8: CPU And RAM Usage For The Recognition Module.

5.4.3 Motion Detection on the Performance of Recognition Module

The motion detection process is performed before entering the face recognition process. The aim of using motion detection is to allow the system to perform face recognition process only if the difference between two means for sequence images is above a specific threshold value, which depends on some factors and the most effective factor is the light condition. Figure 5.9 shows the recognition module performance when the motion detection process is applied.

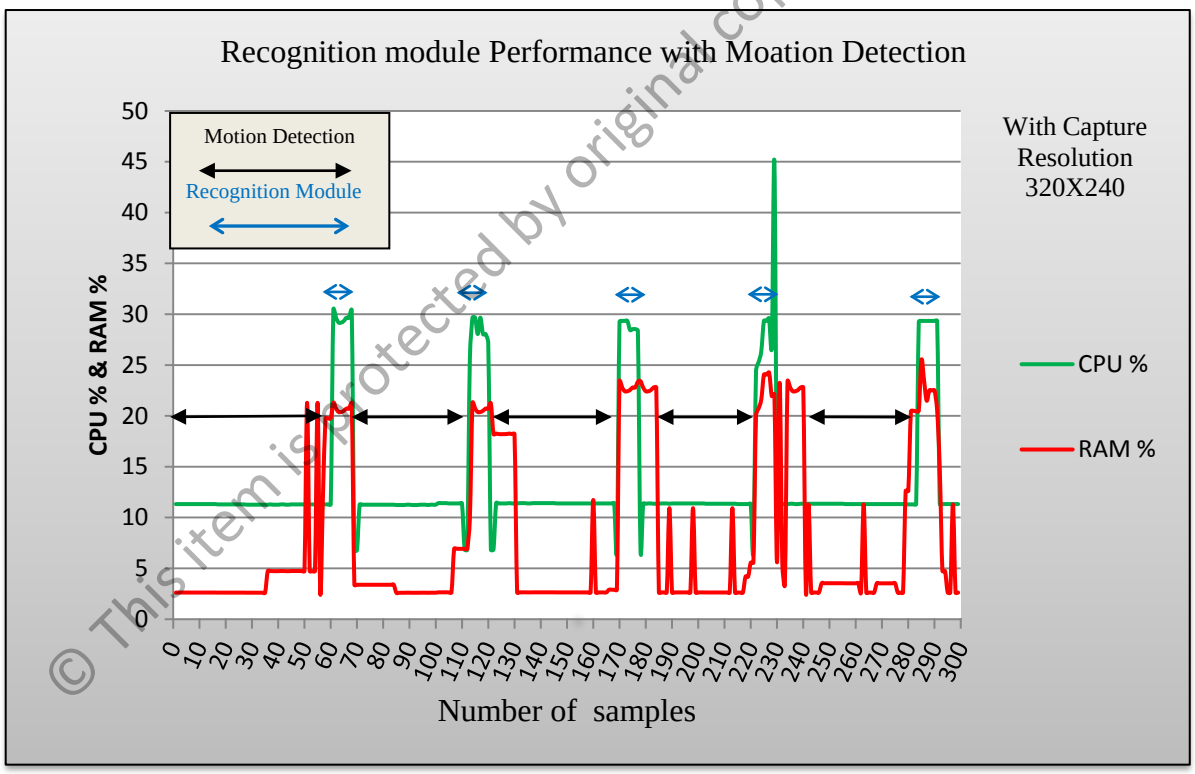

Figure 5.9: CPU And RAM Usage Recognition Module With Motion Detection.

5.5 Summary

This chapter presents the performance evaluation and results for all modules (i.e., Capture Module, Learning Module, and Recognition Module). The output of Capture Module contains face images for the subjects, which are stored in the system. The performance of this module is related to the resolution of the captured images. The output of the Learning Module is the model file that contains the features of the subjects. The performance is evaluated in terms of CPU and RAM usage to create the model file. The mentioned two modules (i.e., Capture Module and Learning Module) are run to prepare the database for the system for the face recognition process, therefore their performance is not directly affect the performance of the system. The Recognition Module can be considered as the core module of the system and its performance is related to the change of the resolution of the captured image. The results prove that the higher the resolution is, the more the CPU and RAM usage. At the end of this chapter, the effect of the motion detection process is presented, which illustrates that the usage of the CPU and RAM are reduced when no motion detected.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This dissertation presents a real time face recognition system using Raspberry Pi (model A). The low power consumption is one of the important requirements for embedded systems design and this is obtained through the reduction of the CPU and RAM usage. In the proposed system, the use of the motion detection process before running the recognition technique reduces the CPU and RAM usage.

The proposed system is implemented using ARM processor and inefficient memory on Raspberry Pi (Model A) board, therefore, the system needs more time to process the captured images compared to the processing time in PC or other expensive types of SBCs. There is a tradeoff between the resolution of the captured images and the performance of the systems in terms of CPU and RAM usage. To get an acceptable performance of the system, the images are captured at resolution (320×240). The selected resolution is not high and thus the HW resource usage is reduced; at the same time, the resolution is not very low and thus it can meet the requirements of the face recognition system.

The software modules of the system are handled by the Python programs, which can perform the required functions. The performance of the system is better when Motion Detection algorithm attach with the Recognition module. Finally, the face recognition system using Raspberry PI (FRRP) consists of:

- Capture Module to captured face images for subjects and save them in the system.

- Learning Module to create model file from the subjects' images.
- Recognition Module to perform the face recognition function.

6.2 Future work

The research has opened up a number of possibilities for the future work. Some suggestions are as follows:

1) The hardware of the system is implemented using Raspberry PI (Model A). For the future work, another type of SBC board can be used to implement the system and test the performance.
2) The software of the system depends on using Fisherfaces and Haar cascade algorithms for face detection and recognition processes. For the future work, other algorithms can be suggested, which may obtain better performance in terms of CPU and RAM usage.
3) There has been a significant research in the field of multi face recognition techniques. These techniques work well for off-line data. It will be desirable to test the technique in on-line data where the data is continuously updated for recognition.

REFERENCES

Ahmad Nasir Che Rosli (2010). *Embedded System for Biometric Identification, Robot Vision*, Ales Ude (3Ed.), ISBN: 978-953-307-077-3, InTech, pp. 557-58, Retrieved from:http://www.intechopen.com/books/robot-vision/embeddedsystem-for-biometric-identification

Beagle Board. (2014). Black. Retrieved September 10, 2014, from http://www.beagleboard.org/black.

Berger, Arnold S. (2002). *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. Lawrence, Kansas, USA. CMP Media LLC.

Belhumeur, P. N., Hespanha J. P., & David J. Kriegman. (1997). Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7.

Chellappa, R., Wilson, C.L.& Sirohey, S. (1995). Human and machine recognition of faces: a survey. Proceedings of the IEEE , vol.83, no.5, pp.705,741.

Chen, S., Zhang T., Zhang C. & Cheng Y.. (2010). A real-time face detection and recognition system for a mobile robot in a complex background. Artif Life Robotics, vol. 15, pp. 439-443.

GNU /Linux (2014). GNU Operating System. Retrieved September 9, 2014, from http://www.gnu.org/

Google Inc. (2014). Google Developers Python Introduction. Retrieved September 9, 2014, from https://developers.google.com/edu/python/introduction.

Hansen, C. A.. (1970) Face Recognition. Second mandatory assignment in the course INF-3701 - Advanced Database Systems, Institute for Computer Science University of Tromso, Norway.

Heseltine, T., Pears, N. & Austin, J. (2004a). Three-Dimensional Face Recognition A Fishersurface Approach. In Proc. of the International Conference on Image Analysis and Recognition, pp. 684-691.

Heseltine, T., Pears, N. & Austin, J. (2004b). Combining multiple face recognition systems using Fisher's linear discriminant. In Proc. of the SPIE Defense and Security Symposium, Biometric Technology for Human Identification, vol. 5404, pp. 470-481.

Hongjun, G., Zhi'an W., Xuhui W.(2010), Transplant of Linux and Embedded System of Boot Loader and LED Driver. Machine Vision and Human-Machine Interface (MVHI), 2010 International Conference, pp. 733–736.

Lee, Hyung-Ji, Lee,Wan-Su & Chung, Jae-Ho. (*2001*). Face recognition using Fisherface algorithm and elastic graph matching. Proceedings. 2001 International Conference on Image Processing, vol. 1, pp.998-1001.

Lee, Y., Moon, Y. & Kim, Y.. (2005). Face and Facial Expression Recognition with an Embedded System for Human-Robot Interaction. Lecture Notes in Computer Science, vol. 3784, pp. 271-278.

Li, Dong-Lin, Prasad, M., Hsu, S., Hong, C., & Lin, C. (2012). Face recognition using nonparametric-weighted Fisherfaces. Li et al. EURASIP Journal on Advances in Signal Processing, 2012:92

Lienhart Rainer & Maydt Jochen. (2002). An Extended Set of Haar-like Features for Rapid Object Detection. IEEE International Conference on Image Processing (ICIP), vol. 1, pp. I-900-I-903.

Mehrab, A.K.M Fazla, Debnath P. & G.M. Mashrur-E-Elahi. (2012). An Approach to Real-Time Portable Device for Face Recognition System. 15th International Conference on Computer and Information Technology (ICCIT).

Miniand Products. (2014). Miniand Products. Retrieved September 10, 2014, from https://www.miniand.com/products/Hackberry%20A10%20Developer%20Board.Vision, 286-296.

Moses, Y. & Ullman, S. (1994). Face Recognition: The Problem of Compensating for Changes in Illumination Direction. European Conf. Computer Vision, pp. 286-296.

Muller, N., Magaia, L. & Herbst B.M. (2004). Singular value decomposition, eigenfaces, and 3D reconstructions. *SIAM Review*, vol. 46, no. 3, pp. 518–545.

Nishimura, J., & Kuroda, T. (2010). Versatile Recognition Using Haar-Like Feature and Cascaded Classifier. Sensors Journal, IEEE , vol.10, no.5, pp.942,951.

ODROID Platforms. (2014). ODROID Platforms. Retrieved September 10, 2014, from http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127.

Open Computer Vision Library Reference Manual. *Intel Corporation*, USA, 2001.

Open Source Hardware Boards. (2014). Retrieved September 10, 2014, from https://www.olimex.com/Products/OLinuXino/.

Parallella Company. (2014). Parallella. Retrieved September 10, 2014, from http://www.parallella.org/board/.

Parmar, Divyarajsinh N & Mehta, Brijesh B. (2013). Face Recognition Methods & Applications. International Journal of Computer Technology & Applications, vol 4, no. 1, pp. 84-86.

RASPBERRY PI. (2014). RASPBERRY PI. Retrieved September 10, 2014, from http://www.raspberrypi.org/

Raspberry Pi. (2014). Element14 & Pi, Raspberry. Retrieved September 8, 2014, from http://downloads.element14.com/raspberryPi1.html

Raspberry Pi. (2014). RPi SD cards. Retrieved September 9, 2014, from http://elinux.org/RPi_SD_cards

Raspberry Pi: The Essential Manual. (2013). The TECHPRO SERIES, Future Publishing Ltd 2013.

Shan, S., Cao, B., Gao, W. & Zhao, D. (2002). Extended Fisherface for face recognition from a single example image per person. ISCAS 2002. IEEE International Symposium on Circuits and Systems, vol. 2, pp.II-81,II-84.

Shibu, K. V. (2009). Intro to Embedded Systems. Tata McGraw Hill Education Private Limited.http://books.google.com.au/books/about/INTRO_TO_EMBEDDED_SYSTEMS_1E.html?id=mp_neOX_uEEC

Shuhaizar, B. D. (2010). Embedded Operating System Optimization for Face Recognition System. Msc. Thesis, School of Computer & Communication Engineering, Universiti Malaysia Perlis.

Shuzhi Sam Ge, Samani, H.A, Ong, Y.H.J. & Chang Chieh Hang. (2008). Active affective facial analysis for human-robot interaction. The 17th IEEE International Symposium on Robot and Human Interactive Communication, *RO-MAN 2008.* pp. 83-88.

Sun, Y., Xie, L., Wang, Z., & An, Y. (2007). An Embedded System of Face Recognition Based on ARM and HMM. L. Ma, R. Nakatsu, and M. Rauterberg (Eds.): ICEC 2007, LNCS 4740, pp. 389–394.

Technologic Systems. (2014). Retrieved September 10, 2014, from http://www.embeddedarm.com/products/board-detail.php?product=TS-7800

Theocharides, Theocharis. (2006). Embedded Hardware Face Detection For Digital Surveillance Systems. PhD Thesis, the Pennsylvania State University, the Graduate School Department of Computer Science and Engineering.

Turk, M. & Pentland, A. (1991a). Eigenfaces for Recognition. J. Cognitive Neuroscience, vol. 3, no. 1.

Turk, M. & Pentland, A. (1991b). Face Recognition Using Eigenfaces. Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 586-591.

UDOO. (2014). UDOO Features. Retrieved September 10, 2014, from http://www.udoo.org/features/.

Viola, P., Jones M. J.,(2004), Robust Real-Time Face Detection. International Journal of Computer Vision, vol. 57, Issue 2, pp. 137-154.

Wilson, P. I., and Fernandez, J. (2006). Facial Feature Detection Using Haar Classifiers. Journal of Computing Sciences in Colleges, vol. 21, no. 4. pp. 127-133

Yin, B., Bai, X., Shi Q. & Yanfeng Sun. (2005). Enhanced Fisherface for Face Recognition. Journal of Information & Computational Science, vol. 2, no. 3, pp. 591-595.

Zakaria, Z., & Suandi, S.A. (2011). Face detection using combination of Neural Network and Adaboost. TENCON 2011 - 2011 IEEE Region 10 Conference , vol., no., pp.335,338.

Zhangi, C-Y & Ruan, Q-Q. (2010). Face Recognition Using L-Fisherfaces. Journal Of Information Science And Engineering, vol. 26, pp. 1525-1537.

APPENDICES

Appendix A: Linux Commands

Following are some commonly used in Raspbian for this project

Command	Description
`apt-get`	The command-line tool for handling packages, and may be considered the user's.
`cd`	Change the current working directory to a specific folder.
`dd`	Dump Data –convert and copy a file (use for RAW storage)
`dphys-swapfile`	dphys-swapfile computes the size for an optimal swap file (and resizes an existing swap file if necessary), mounts an swap file, unmounts it, and delete it if not wanted any more.
`ls`	List information about files by current directory.
`lsusb`	Utility for displaying information about USB buses in the system and the devices connected.
`make`	GUN make utility to maintain groups of programs.
`mkdir`	Create new folder
`nano`	Small, free and friendly text editor.
`ps`	Process status, information about process run in the memory.
`raspi-config`	The script helps user to configure your Raspberry Pi
`su`	Substitute user identity. Run a command with substitute user and group id; allow one user to temporarily become another user.
`tar`	Tape archive, store list or extract files in an archive
`top`	Provide information about the most CPU-intensive processes currently running.
`umount`	The umount command is used to manually unmount file systems on Linux and other Unix-like operating systems.

`wget`	A free utility for non-interactive download of files from the Web

Appendix B: The installation of the Raspbian OS

❖ Using the Graphical Interface.

```
Falah@UNimap:~$ sudo wget http://202.166.85.154/bt/4627bda
58f99162374ecb442a24cf6d4db2f1e09/data/2014-06-20-wheezy-
raspbian.zip

e09/data/2014-06-20-wheezy-raspbian.zip

Connecting to 202.166.85.154:80... connected.

HTTP request sent, awaiting response... 200 OK

Length: 826192020 (788M) [application/zip]

Saving to: ‘2014-06-20-wheezy-raspbian.zip’
```

Installation of the usb-imagewriter package.

```
Falah@UNimap:~$ sudo apt-get install python-glade2
```

```
Falah@UNimap:~$ sudo apt-get install usb-imagewriter
```

❖ Using Command Line Interface.

Download the desired raspbian.img file

```
Falah@UNimap:~$ sudo umount /dev/ sdbX
```

Run the shell below to write the image to SD card.

```
Falah@UNimap:~$ sudo dd if=/path/to/ raspbian.img
of=/dev/sdbX bs=1M
```

Appendix C: The installation of the Image and video libraries

- JPEG library (the Independent JPEG library is a library for handling JPEG files).

```
pi@raspberrypi:~$ sudo apt-get install libjpeg62-
dev
```

- libtiff4-dev (Tag Image File Format library(TIFF))

```
pi@raspberrypi:~$ sudo apt-get install libtiff4-
dev
```

- libjasper-dev (Development files for the JasPer JPEG-2000 library)

```
pi@raspberrypi:~$ sudo apt-get install libjasper-
dev
```

- libavcodec-dev (This is the codec library from the ffmpeg project. It supports most existing encoding formats (MPEG, DivX, MPEG4, AC3, DV,...)

```
pi@raspberrypi:~$ sudo apt-get install libavcodec-
dev
```

- libavformat-dev (Libav is a complete, cross-platform solution to decode, encode, record, convert and stream audio and video. It supports most existing file formats (AVI, MPEG, OGG, Matroska, ASF,...)

```
pi@raspberrypi:~$ sudo apt-get install
libavformat-dev
```

- libswscale-dev (This is the video scaling library from the ffmpeg project).

```
pi@raspberrypi:~$ sudo apt-get install
libswscale-dev
```

- libv4l-dev (libv4l is a collection of libraries which adds a thin abstraction layer on top of video4linux2 devices).

```
pi@raspberrypi:~$ sudo apt-get install libv4l-dev
```