

**VISION-BASED HAND GESTURE RECOGNITION FOR  
REAL-TIME EMBEDDED SYSTEM PLATFORMS**

**BY**

**JAKKRIT DULAYATRAKUL**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF ENGINEERING (INFORMATION AND  
COMMUNICATION TECHNOLOGY FOR EMBEDDED SYSTEMS)  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY  
THAMMASAT UNIVERSITY  
ACADEMIC YEAR 2016**

**VISION-BASED HAND GESTURE RECOGNITION FOR  
REAL-TIME EMBEDDED SYSTEM PLATFORMS**

**BY**

**JAKKRIT DULAYATRAKUL**



**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF ENGINEERING (INFORMATION AND COMMUNICATION  
TECHNOLOGY FOR EMBEDDED SYSTEMS)  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY  
THAMMASAT UNIVERSITY  
ACADEMIC YEAR 2016**

**VISION-BASED HAND GESTURE RECOGNITION FOR REAL-TIME  
EMBEDDED SYSTEM PLATFORMS**

A Thesis Presented

By

JAKKRIT DULAYATRAKUL

Submitted to

Sirindhorn International Institute of Technology

Thammasat University

In partial fulfillment of the requirement for the degree of  
MASTER OF ENGINEERING (INFORMATION AND COMMUNICATION  
TECHNOLOGY FOR EMBEDDED SYSTEMS)

Approved as to style and content by

Advisor and  
Chairperson of Thesis Committee



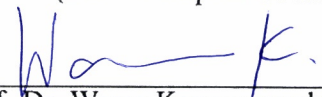
(Asst. Prof. Dr. Itthisek Nilkhamhang)

Committee Member



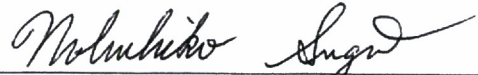
(Dr. Wutthiphat Covanich)

Committee Member and  
Chairperson of Examination Committee



(Assoc. Prof. Dr. Waree Kongprawechnon)

Committee Member



(Assoc. Prof. Dr. Nobuhiko Sugino)

AUGUST 2017

## **Acknowledgments**

The authors would like to thank Thailand Advance Institute of Science and Technology (TAIST), Sirindhorn International Institute of Technology (SIIT), Thammasat University, National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology, and National Research University Project (NRU), Thailand Office of Higher Education Commission for financial support.



## **Abstract**

### **VISION-BASED HAND GESTURE RECOGNITION FOR REAL-TIME EMBEDDED SYSTEM PLATFORMS**

by

Jakkrit Dulayatrakul

Bachelor of Engineering in Aviation Maintenance Engineering, Rangsit University,  
2011

Master of Engineering (Information and Communication Technology for Embedded  
Systems), Sirindhorn International Institute of Technology, 2017

The purpose of this thesis is to develop a real-time, vision-based hand gesture recognition system on an embedded platform that can be used for accurate and efficient human-machine interaction under dynamic lighting conditions. A monocular camera will be employed to capture a human user situated 50-80 cm from the system to control media playback of an MP3 player using six different hand gestures. The application design is divided into three main parts, including image processing, control interface, and MP3 playback. The image processing part includes camera interfacing, hand segmentation, and finger detection. The control interface consists of a finite machine to connect the output of hand gesture recognition with MP3 playback commands. The real-world problem of changes in light intensity and illumination is directly considered by using HSV color segmentation and UGV background subtraction to provide accurate performance, sensitivity, and robustness in both dim and bright conditions. This is then combined with top-hat transform to produce a simple and precise algorithm for recognizing hand gestures, which includes open palm (OP), forefinger (FF), forefinger and thumb (FT), fist (FS), leftward thumb (TL), and rightward thumb (TR). In addition, the execution time of the algorithm is analyzed on

an embedded system. By using parallel processing and multi-thread programming to handle the recognition of both the left and right hand regions simultaneously, the average processing time is found to be 545 ms or approximately 1.83 fps, which is sufficient to ensure smooth operations.

**Keywords:** HMI, Hand gesture, Image processing, Embedded system.



## Table of Contents

Chapter	Title	Page
	Signature Page	i
	Acknowledgments	ii
	Abstract	iii
	Table of Contents	v
	List of Figures	viii
	List of Tables	x
	List of Acronyms	xi
1	Introduction	1
	1.1 Hand Gesture Recognition	1
	1.2 Real-Time System and Hand Gesture Recognition	3
	1.3 Motivation	4
	1.4 Problem Statement	5
	1.5 Objectives	5
	1.6 Scope and Limitation	5
	1.7 Thesis Organization	6
	1.8 Publication	6
2	Literature Review	8
	2.1 Embedded Systems for Image Processing	8
	2.1.1 Embedded System	8
	2.1.1.1 Software-based Architecture	8
	2.1.1.2 Hardware-based Architecture	9
	2.1.2 Criteria of Image Processing	11
	2.1.3 Real-time Image Processing on Embedded system	11
	2.2 Image Processing for Hand Gesture Recognition	13
	2.2.1 Hand Segmentation	14

2.2.1.1	Color Segmentation	14
2.2.1.2	Background Subtraction	16
2.2.2	Finger Detection	18
2.2.2.1	Convex Hull	19
2.2.2.2	K-Curvature	21
2.2.2.3	Center of Gravity Distance Measurement	22
2.2.2.4	Other Methods	23
3	System Design	26
3.1	Application Design	26
3.2	Hardware Design	27
3.2.1	Raspberry Pi2 Module B	28
3.2.2	Raspberry Pi camera	28
3.2.3	Expand memory of Raspberry Pi2 module B	29
3.3	Software Design	29
3.4	Acquiring an Image from Camera	30
3.5	Environment Setting	33
3.6	Multithreading Implementation	33
3.7	Controller Interface	34
3.8	MP3playback Application	35
4	Hand Segmentation	39
4.1	Overview of Hand Segmentation	39
4.1.1	Pre-Processing	39
4.1.2	Color Segmentation	41
4.1.3	Background Subtraction	43
4.1.4	AND Operation	47
4.1.5	Post-Processing	48
4.2	Discussion of the Hand Segmentation Output	49
5	Finger Detection and Gesture Recognition	50



5.1	Finger Detection by Using Top-Hat Transform.	50
5.1.1	Top-Hat Transform	50
5.1.2	Distance Transformation	52
5.1.3	Center of Gravity	53
5.1.4	Distance Measurement	54
5.2	Hand Gesture Recognition	55
5.3	Discussion of Finger Detection and Hand Gesture Recognition	57
6	Experimental Results	58
6.1	Dynamic Lighting Condition	58
6.1.1	Light Intensity	58
6.1.2	Performance Analysis	59
6.1.2.1	Background Subtraction	61
6.1.2.2	Color Segmentation	62
6.2	Effect of Skin Color	65
6.3	Real-Time Implementation	66
6.3.1	Execution Time	66
6.3.2	Serial and Parallel Processing	67
7	Conclusion	69
	References	70

## List of Figures

<b>Figures</b>	<b>Page</b>
1.1 Human-machine interaction [35].	1
1.2 Hand anatomy and gestures.	2
1.3 Glove-based and vision-based hand gesture recognition.	2
1.4 Comparison of PC vs embedded systems.	4
2.1 Architecture of software-based embedded system [40].	9
2.2 Common architecture of hardware-based embedded systems [9].	10
2.3 CLPD architecture [9].	10
2.4 FPGA architecture [9].	11
2.5 The image processing pyramid [9].	12
2.6 The algorithm flow chart of [46] and [17].	14
2.7 The RGB cube [23].	15
2.8 The YCrCb cube[22] and HSV cone [23].	16
2.9 Example of clockwise and counter-clockwise direction with three reference points [41].	19
2.10 Determining a simple convex polygon [41].	20
2.11 The convex hull method for finger detection [20].	21
2.12 Example error of convex hull method for finger detection [20].	21
2.13 The K-curvature method for finger detection [20].	22
2.14 The perimeter curvature method for finger detection [20].	24
2.15 Circle detection with pixel grouping [16].	24
2.16 Middle axis method [18].	25
3.1 Hand gesture pattern for MP3 player.	27
3.2 Raspberry Pi2 Module B and Pi camera.	28
3.3 32 GB Micro SD card for Raspberry Pi2 module B	29
3.4 Software design flowchart.	31
3.5 Control finite state machine.	31
3.6 The image acquisition process by using Userland and OpenCV	32

3.7	Region of interest and the distance between the user and the camera.	34
3.8	Controller interface software flow chart	36
3.9	The MP3player flow chart.	37
4.1	Hand segmentation flow chart	40
4.2	Pre-processing image.	41
4.3	Human skin color segmentation	43
4.4	5×5 custom Sobel operator mask	44
4.5	Background subtraction output image.	47
4.6	Output image from combination of color space segmentation and background subtraction methods.	47
4.7	Post-processed output image from combination of color space segmentation and background subtraction methods.	48
5.1	Finger detection flow chart	51
5.2	Finger detection using top-hat transform.	52
5.3	Euclidean distance transform 5×5 mask.	52
5.4	Comparison between palm region and distance transform of palm region.	53
5.5	The example image of finger detection.	55
6.1	Three different light intensity levels.	59
6.2	Six different hand gestures under bright lighting conditions.	60
6.3	Six different hand gestures under dim lighting conditions.	60
6.4	RGB color segmentation under bright (a)-(c) and dim (d)-(f) conditions.	62
6.5	Comparison of sensitivity for each color space using UGV-based background subtraction method.	63
6.6	YCrCb color segmentation under bright (a)-(c) and dim (d)-(f) conditions.	64
6.7	HSV color segmentation under bright (a)-(c) and dim (d)-(f) conditions.	64
6.8	FF hand-gesture recognition of 5 users with different skin color.	65
6.9	Hue circle and threshold values for 5 users with different skin color.	65
6.10	The proposed method implemented using serial and parallel processing.	67

## List of Tables

<b>Tables</b>	<b>Page</b>
2.1 Comparison of execution time between ARM and PC in [46].	12
2.2 The comparison of execution time between FPGA and PC in [17].	12
3.1 The setting parameter for capturing the image.	32
4.1 Human skin threshold color in three different color space	42
5.1 The relationship between number of finger and hand gesture	56
6.1 Light intensity	59
6.2 Sensitivity of RGB Color Space	61
6.3 Sensitivity of YCrCb Color Space	61
6.4 Sensitivity of HSV Color Space	61
6.5 Execution time of each process.	66

## List of Acronyms

HMI	Human-Machine Interaction (HMI)
ARM	Advanced RISC Machines
OS	Operating Systems
FPGA	Field Programmable Gate Arrays
SAD	Sum of Absolute Difference
API	Application Programming Interface
MMAL	Multi-Media Abstraction Layer
OP	Open Palm
FF	Forefinger
FS	Fist
FT	Forefinger and Thumb
TL	Leftward Thumb
TR	Rightward Thumb
MCU	Main Controller Unit
CMOS	Complementary Metal-Oxide-Semiconductor
UGV	Unit Gradient Vector
SE	Structure Element
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
GPIO	General Port Input-Output
ms	Millisecond
$\mu s$	Microsecond
fps	Frame Per Second

# Chapter 1

## Introduction

Communication methods between humans and machines are often referred to as human-machine interaction (HMI), which evolved into human-computer interaction (HCI) with the rise of modern computer systems [2]. HMI is concerned with ergonomics as it relates to human perception and action. Human perception includes vision, hearing, touch, and movement, while human action refers to force, speech, and gestures. Conventional methods for HMI usually take the form of plug-in devices, such as a keyboard, mouse, joystick, and so forth. Fig. 1.1 illustrates the typical HMI process in which the human user perceives information on a display and responds using an actuator to send commands to the machine. The machine executes those commands and outputs the results on to the display as feedback to the user. At present, HMI has become an important part of our digital life. Currently, one of the most popular and useful form of HMI is hand gesture recognition because of its capabilities and usability that can be applied to various applications and platforms.

### 1.1 Hand Gesture Recognition

Hand gesture is a physical, non-verbal form of communication using visible hand postures. The anatomical description of the human hand is shown in Fig. 1.2(a), consisting of the palm in the center that supports five extension digits, which are the thumb, index/fore, middle/long, ring, and small/little fingers. The movements and

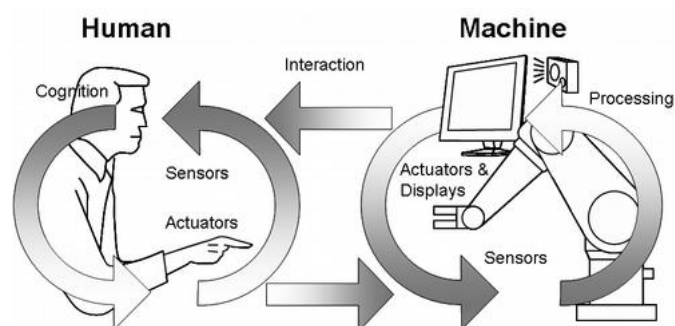
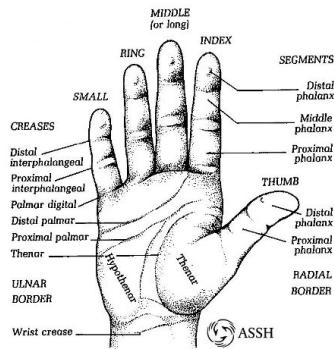
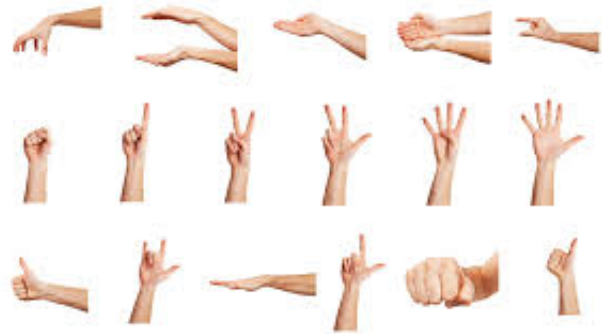


Figure 1.1 Human-machine interaction [35].



(a) Anatomy of human hand [5]

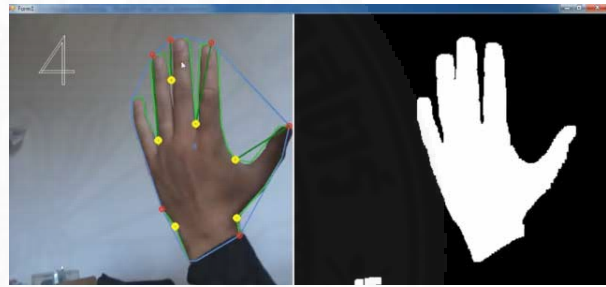


(b) Basic hand gestures [15]

Figure 1.2 Hand anatomy and gestures.



(a) Glove-based [21]



(b) Vision-based [3]

Figure 1.3 Glove-based and vision-based hand gesture recognition.

relative positions of the fingers and palm are defined as hand gestures can be used as sign language and communication tools, as shown in 1.2(b).

In computer science, hand gesture recognition is an important form of HMI. It can be implemented in two different ways, which are 1) glove-based interface, such as the “data glove”, and 2) vision-based using a camera, as shown in Fig. 1.3 [33]. The data glove uses accelerometers and fiber optic bend-sensors to determine hand position, movement, and finger bending. Vision-based systems employ cameras to capture images of the hand and then recognize each hand gesture using image processing and computer vision algorithms. Since there is no physical contact or limitations on the user, vision-based hand gesture recognition is more natural and comfortable but offers lower performance than glove-based interfaces. Furthermore, the accuracy is dependent on environmental factors, such as illumination changes and background clutter.

## 1.2 Real-Time System and Hand Gesture Recognition

An important goal in the development of HMI is the processing speed, which should be executed in real-time while considering the average time for human response. According to [30], a real-time system must satisfy bounded response-time constraints or risk severe consequences, including failure. In addition, the logical correctness of the system is based on both the correctness of the outputs and their timeliness. A real-time system can be classified into three categories based on consideration of the response time or deadline:

- **Hard real-time:** the system must respond within the deadline, otherwise the system will fail.
- **Soft real-time:** the system should respond within the deadline, otherwise the system performance will degrade but will not fail.
- **Firm real-time:** the system is tolerant of a small, fixed number of missed deadlines, beyond which the system will fail.

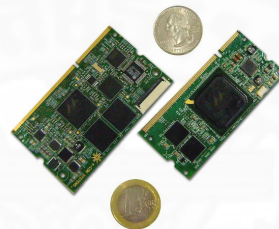
Hand gesture recognition is a soft real-time system that is subjected to both hardware and software performance constraints. The system speed depends greatly on hardware specifications of camera sensors, CPU, system memory, and architecture. Furthermore, hardware platforms can be classified as general-purpose computers (PCs) or embedded systems. The main differences between PCs and embedded systems include physical appearance, performance of the processor, memory resources, and power consumption. In general, the PC will have a large physical size with higher performing processor and memory units, as well as more power consumption. Fig. 1.4 shows the comparative size of a PC and embedded system. The challenge is therefore how to implement real-time gesture recognition on an embedded system with limited physical resources and performance.

Software performance depends on the programming language, algorithm, and execution time. Many programming languages support real-time system design, such as Ada, C, C++, C#, and Java [30]. The choice will therefore depend on the hardware platform, application, and open-source libraries. For HMI and similar systems, the





(a) General-purpose personal computer [19]



(b) Embedded system [44]

Figure 1.4 Comparison of PC vs embedded systems.

performance of the algorithm includes accuracy, robustness, and processing speed. In order to ensure that real-time deadlines are met, software optimization is necessary to improve the execution time.

### 1.3 Motivation

HMI applications using hand gesture recognition have been an active field of research since 1980 to the present [33]. In particular, vision-based systems using only one camera offers a low-cost alternative to glove-based interfaces that is more natural and comfortable for the user. Due to the recent growth and widespread adoption of low-cost embedded systems, such as Arduino and Raspberry Pi, that are compatible with different camera sensors, this research is motivated to develop a real-time vision-based hand gesture recognition system on an embedded platform for HMI applications. This system should offer the portability and power consumption advantages of embedded systems, while performing accurately in real-time and maintain robustness to illumination changes.

## **1.4 Problem Statement**

The main problem statement of this research is to develop an accurate vision-based hand gesture recognition system that is robust to dynamic lighting conditions while satisfying real-time requirements. Dynamic lighting refers to sudden changes in image intensity that often occur in real-world image processing applications for various reasons. This greatly effects the performance of conventional color segmentation and background subtraction algorithms. Therefore, robustness to illumination changes is an important part to ensure performance and accuracy of the hand gesture recognition system. Moreover, the algorithm is to be implemented on an embedded system with limited computing and memory resources. In order to ensure soft real-time performance for smooth interfacing, software optimization will be necessary.

## **1.5 Objectives**

The objectives of this research are as follows:

- Design a vision-based hand gesture recognition system to control an MP3 player on an embedded platform.
- Implement a hand gesture recognition algorithm that is robust to dynamic lighting conditions.
- Optimize the embedded software to satisfy soft real-time system requirements and ensure smooth operations.

## **1.6 Scope and Limitation**

This study will focus on developing a vision-based hand gesture recognition system to be implemented on a stand-alone embedded platform to control media playback of an MP3 player. The image will be captured by a monocular camera and processed on Raspberry Pi2 Module B with a quad-core ARM processor. Implementation of the image processing algorithms will utilize the OpenCV open-source library and multi-threading programming using the OpenMP open-source library. The distance between the human user and the camera set-up is specified as 50

to 80 cm, corresponding to a region-of-interest (ROI) for each hand of  $120 \times 120$  when the image resolution is  $320 \times 240$ . The pattern of hand gestures to be recognized in this study consists of six hand postures, which are open palm (OP), forefinger (FF), forefinger and thumb (FT), fist (FS), leftward thumb (TL), and rightward thumb (TR). This will translate into six commands for the MP3 player as follows: initialization, play, pause, stop, next track, and previous track. The user skin color is assumed to be typical for people with Asian ethnicity. The environment is a fluorescent-lit room with complex, cluttered background that can be set to dark, dim, and bright conditions. The soft real-time constraint is set to ensure smooth performance based on an average user interaction response time of one command per second.

## 1.7 Thesis Organization

The structure of this thesis starts with Chapter 2 that contains the literature review describing related methods used for this work. Chapter 3 describes the system design, showing the concept and organization of both hardware and software components. Chapter 4 discusses the hand segmentation used to robustly extract the hand region under dynamic lighting condition. Chapter 5 presents the algorithm for finger detection and gesture recognition, including classification of hand postures. Chapter 6 includes the experimental results and analysis. Lastly, Chapter 7 presents the conclusion of this work.

## 1.8 Publication

The results of this research have been published in two international conference proceedings and submitted to a national journal:

- Jakkrit Dulayatrakul, Itthisek Nilkhamhang, Wutthiphat Covanich and Nobuhigo Sugino, “Adaptive combination board using ARM and FPGA for real-time control and image processing of a quad-rotor platform”, *5th International Conference on Information and Communication Technology for Embedded System (IC-ICTES)*, 2014

- Jakkrit Dulayatrakul, Itthisek Nilkhamhang, Toshiaki Kondo and Pawin Prasertsakul, “Robust implementation of hand gesture recognition for remote human machine interaction”, *7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2015
- Pawin Prasertsakul, Jakkrit Dulayatrakul, Itthisek Nilkhamhang and Toshiaki Kondo, “A real-time hand segmentation method using background subtraction and color information”, *Journal of the Songklanakarin Journal of Science and Technology (SJST)*, 2016 [submitted]



## Chapter 2

### Literature Review

This chapter provides the literature review concerning background knowledge and information utilized by this thesis. It includes sections on embedded systems for image processing, hand segmentation algorithms, and hand gesture recognition methods.

#### 2.1 Embedded Systems for Image Processing

This thesis will implement hand gesture recognition on an embedded system using a monocular camera. Therefore, it is necessary to understand the software and hardware architecture of embedded systems and the requirements for real-time image processing.

##### 2.1.1 Embedded System

An embedded system is essentially a computer with embedded components capable of satisfying real-time constraint requirements [9]. It is always smaller and lighter than a general purpose personal computer (PC), while consuming less power and containing limited memory resources. The programmable embedded system can be divided into two categories, which are software-based and hardware-based.

###### 2.1.1.1 Software-based Architecture

Software-based architecture usually includes a central processing unit (CPU) and an arithmetic logic unit (ALU). The CPU operates to feed the required data to the ALU by fetch/decode/execute cycle per system clock. The duty of the ALU is to sequentially compute the data received from the CPU. An example of a software-based processor is shown in Fig. 2.1 [40], where PC is a program counter register used to hold the address of the current instruction. ACC is an accumulator that holds a data value. The ALU performs some operation on the binary operands. IR is the instruction register containing the current instruction to be executed, and memory is the stored data. Although the software-based architecture is processed serially, it can provide parallel processing for multi-core systems, which has separate ALU per core and

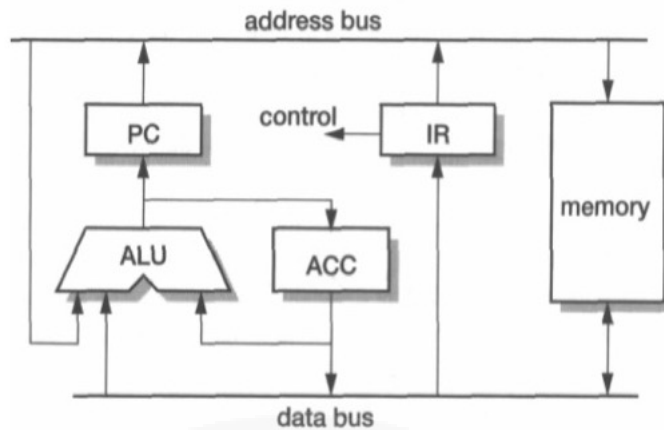


Figure 2.1 Architecture of software-based embedded system [40].

allows the developer to design parallel tasks. Examples of this architecture include advanced RISC machines (ARM) and digital signal processors (DSP). Typically, the software-based architecture is implemented directly by using computer languages such as C/C++ and assembly. Recently, micro-controllers capable of running operating systems (OS), such as Linux, allow for further compatibility with other languages, such as Python.

### 2.1.1.2 Hardware-based Architecture

Hardware-based architecture employs programmable generic circuit that can be reconfigured, and thus is not limited by the number of ALUs when compared to software-based systems [9]. For this reason, hardware-based architecture supports parallel processing. The most common systems consist of programmable logic arrays containing AND-OR interconnections (PLA), programmable array logic (PAL), and programmable read-only memory (PROM), as shown in Fig. 2.2. Other variations include complex programmable logic devices (CPLD) based on PAL or field-programmable gate arrays (FPGA) based on PROM, as shown in Fig. 2.3.

In general, FPGA is a more flexible structure than CPLD, making it the most popular hardware-based architecture for embedded systems at the current time. The basic architecture of FPGA is shown in Fig. 2.4, which includes logic blocks, interconnects, I/O blocks, configuration control, and clock control. Each logic block contains a logic cell, consisting of a look-up-table (LUT) and latches. The LUT is used

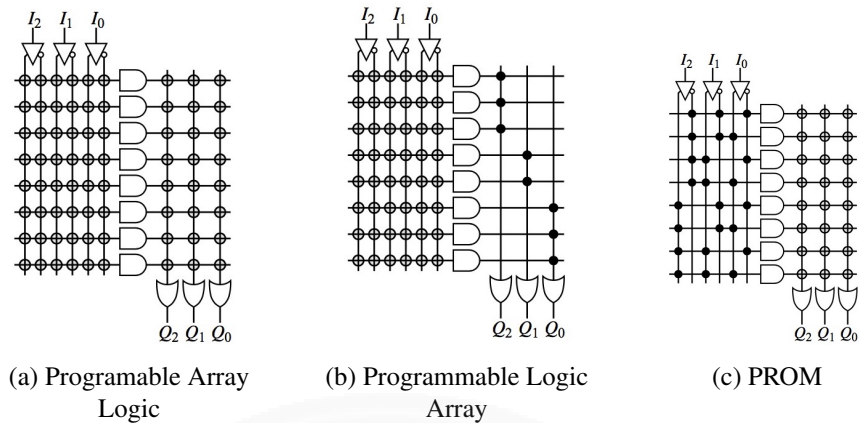


Figure 2.2 Common architecture of hardware-based embedded systems [9].

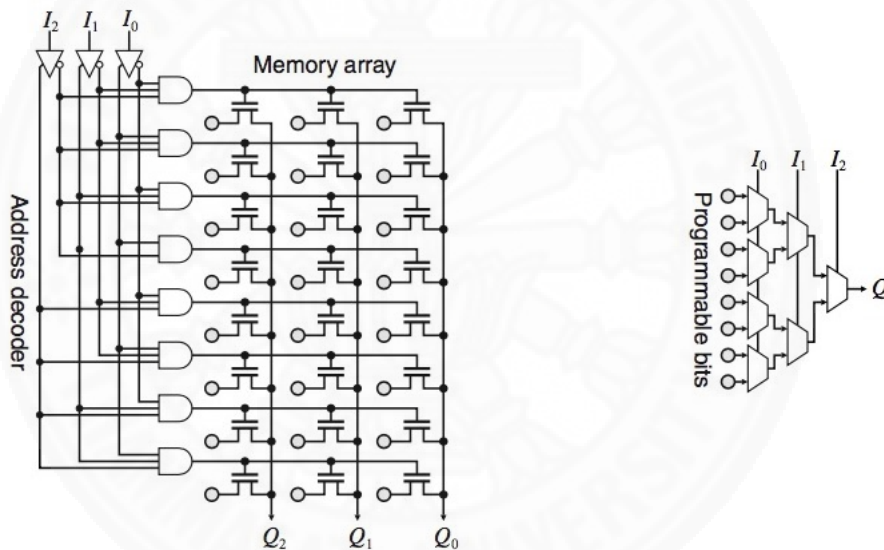


Figure 2.3 CLPD architecture [9].

to configure the function of the input between 3 to 6 inputs. The latch is used to register the output. Interconnects are the programmable interconnection array between different logic blocks. The I/O block acts as the input-output port with external devices. The clock control is used to manage synchronization with other devices and components. The configuration control stores the status of all switches, structures, and options of the FPGA. Programming the hardware-based architecture can be done using hardware languages, such as HDL, VHDL, and Verilog. Alternatively, mid- to high-cost FPGAs are compatible with MATLAB Simulink.

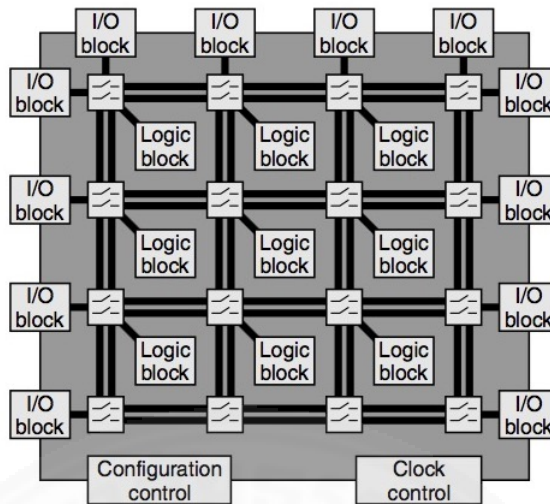


Figure 2.4 FPGA architecture [9].

### 2.1.2 Criteria of Image Processing

In [9], the criteria for image processing is described as a pyramid, shown in Fig. 2.5, with three levels: low, intermediate, and high. The lowest level is pre-processing, which is an image-to-image transformation to perform distortion corrections, contrast enhancements, filtering, or edge detection. This level considers a high volume of pixel array for calculation and thus requires more time to process than other levels. The intermediate level corresponds to segmentation, which transforms an image into region-on-interest or features. It involves less data volume than low level processing. Lastly, high level image processing employs the feature data from the intermediate level to classify and recognize objects. The data used at this level is no longer image-based.

### 2.1.3 Real-time Image Processing on Embedded system

From the previous discussion, embedded systems can be classified as either software- or hardware-based architectures. Many works have implemented image processing algorithm on both platforms in real-time. For example, You Lei et al. [46] presented a real-time hand gesture recognition algorithm implemented on an ARM Cortex-A9 quad core processor and compared the execution time to a personal computer (Intel Core i3-500, 4GB DDR3 memory), as shown in Table 2.1. The



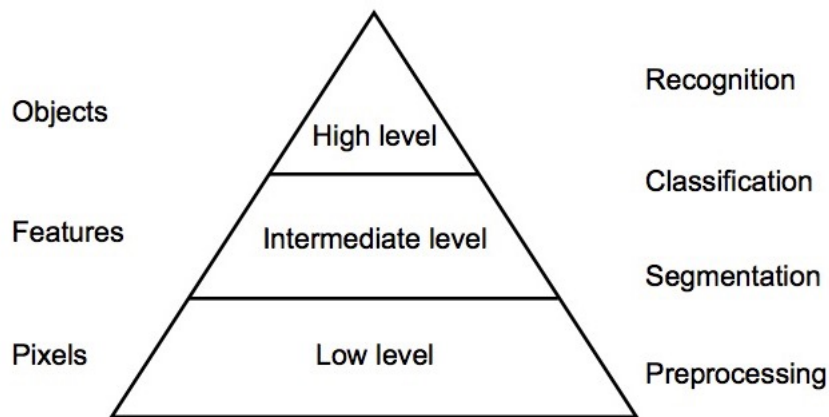


Figure 2.5 The image processing pyramid [9].

Table 2.1 Comparison of execution time between ARM and PC in [46].

	ARM	PC
Skin Model Building	12.03 sec	10.06 sec
Hand Segmentation	0.43 sec	0.38 sec
Hand Model Building	0.25 sec	0.19 sec
Hand Gesture Recognition	0.02 sec	0.02 sec

Table 2.2 The comparison of execution time between FPGA and PC in [17].

	FPGAs	PC
Skin Model Building	0.04 sec	0.289 sec

execution time performance of this software-based, serial embedded processor is similar to a general-purpose PC with much higher specifications.

Similarly, G. Liu et al. [17] performed color segmentation to detect human skin color using FPGAs. The FPGA utilized in the paper was a Cyclone 2 EP2C20F484C6 with video processor SAA7113H as an analog-to-digital converter (ADC), two 1 MB SRAMs as image buffers, and a video interface. The frame rate when implemented using FPGAs was compared to a PC (Pentium 4, 2.4 GHz, 1GB DDR memory) in Table 2.2. It is noted that FPGAs can complete the image processing algorithm in 0.04 second or 25 frames per second (fps), which is faster than the PC at 0.289 second or 3.46 fps.

In comparing the two works, both use the same algorithm for color segmentation based on Gaussian mixture model (GMM) with YCrCb. It is clear that FPGA, with its parallel architecture, provides much faster performance than ARM or PC-based

systems. In fact, the limited number of ALUs available to ARM and PC systems becomes a bottleneck for execution time, especially when dealing with large data size or images. This is most prevalent for low or intermediate level image processing. The flow chart for serial processing using ARM is compared to parallel processing using FPGAs in Fig. 2.6. However, even though FPGAs can provide much higher frame rates, it is harder to directly implement algorithms because of floating point value conversions. This is shown by the example of converting from YCrCb to RGB using ARM/PC, according to (2.1), and using FPGA, according to (2.2):

$$\begin{aligned}
 R &= Y + 1.402 \cdot (C_r - 128) \\
 G &= Y - 0.344 \cdot (C_b - 128) - 0.714 \cdot (C_r - 128) \\
 B &= Y + 1.772 \cdot (C_b - 128)
 \end{aligned} \tag{2.1}$$

$$\begin{aligned}
 16 \cdot R &= 16 \cdot Y + 22 \cdot C_r - 2871 \\
 16 \cdot G &= 16 \cdot Y - 6 \cdot C_b - 11 \cdot C_r - 2166 \\
 16 \cdot B &= 16 \cdot Y + 28 \cdot C_b - 3629
 \end{aligned} \tag{2.2}$$

Furthermore, implementation of complex, high level image processing algorithms is more difficult on FPGAs, even using high-cost boards such as Vertex-5 XC5VLX50T [38]. Therefore, it is concluded that both software- and hardware-based architectures, represented by ARMs or FPGAs respectively, have different advantages depending on the image processing task. ARM provides convenient programming tools and feedback due to its single ALU implementation but suffers from execution time bottlenecks. FPGA employs parallel processing to greatly reduce execution time and manage large data sets but become harder to manage when dealing with complex algorithms.

## 2.2 Image Processing for Hand Gesture Recognition

Hand gesture recognition refers to a computer vision technique to determine and classify hand postures from a stream of images captured by a camera. Thus, it requires both low, intermediate, and high level image processing to achieve accurate and robust

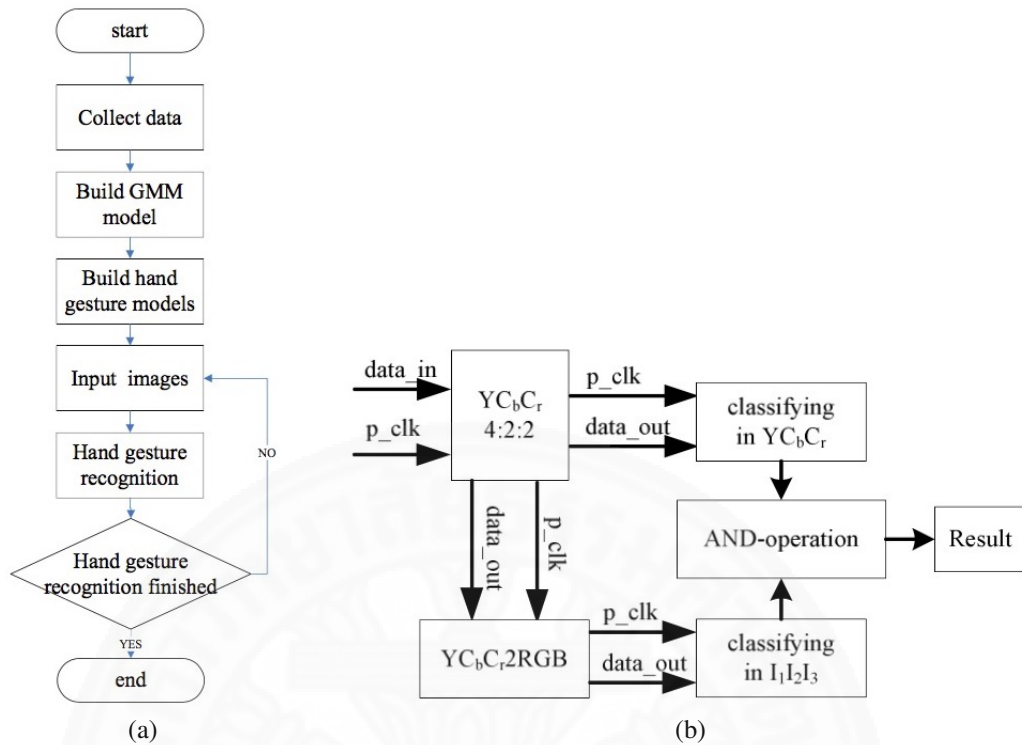


Figure 2.6 The algorithm flow chart of [46] and [17].

performance. Existing algorithms contain two main components for hand segmentation and fingertip detection.

## 2.2.1 Hand Segmentation

Hand segmentation is a method for extracting the hand region from an image frame. In general, there are two approaches based on color segmentation and background subtraction. Furthermore, these two methods can be combined for hand gesture recognition, as shown in [6], [43].

### 2.2.1.1 Color Segmentation

Color segmentation is commonly used to extract the hand from the input image by detecting each pixel that has a color value within the range of human skin. Since 1931, the International Commission on Illumination (CIE) specified the color space by adapting the standard color curves to a set of three numbers from specific spectral power distribution (SPD) [23]. RGB (red-green-blue) is the standard color space that can be transformed to other color spaces, such as YCrCb, HSV, YUV, and vice versa

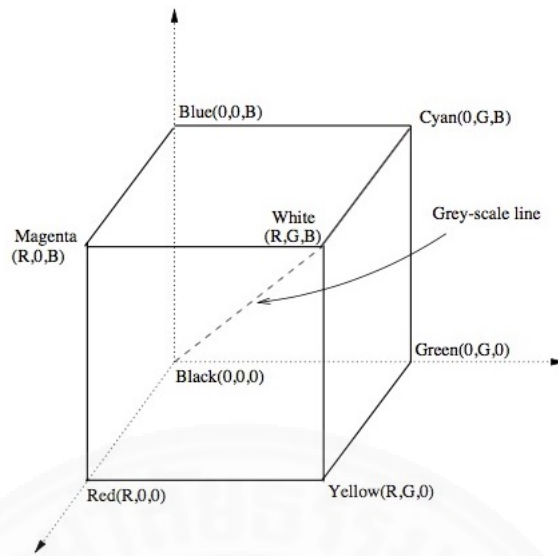


Figure 2.7 The RGB cube [23].

by mathematical conversion. Different color spaces have varying properties and advantages for video transmutation and image processing. Each color space also have different thresholds for human skin color segmentation. Normally, the standard color spaces for hand gesture recognition are RGB, YCrCb, and HSV.

RGB is a simple color space based on the primary color of light (red-green-blue). Video devices, such as a cathode ray tube (CRT), use this color space to reproduce the color for each dot component. The mixture of RGB colors in different ratio produces the cube in Fig. 2.7. The origin of the cube corresponds to black, where each RGB value is equal to zero. The maximum value of RGB is white. Various researchers have employed RGB for hand segmentation and skin color detection. In [34] and [47], the RGB color space was implemented on FPGA platforms. Alternatively, [37] present the combination of RGB and HSV for robust hand segmentation.

YCrCb color space was developed by the ITU-R BT.601-4 international standard for digital video components [22]. Y refers to the luminance or grayscale color that can be transformed from RGB. Cr and Cb represent the chroma components in red and blue. Fig. 2.8(a) shows the comparision of RGB and YCrCb cubes. Many researches on human skin detection are based on the YCrCb color space, because it improves the luminance effect by separating the brightness component from the color components [7], [38], [46].

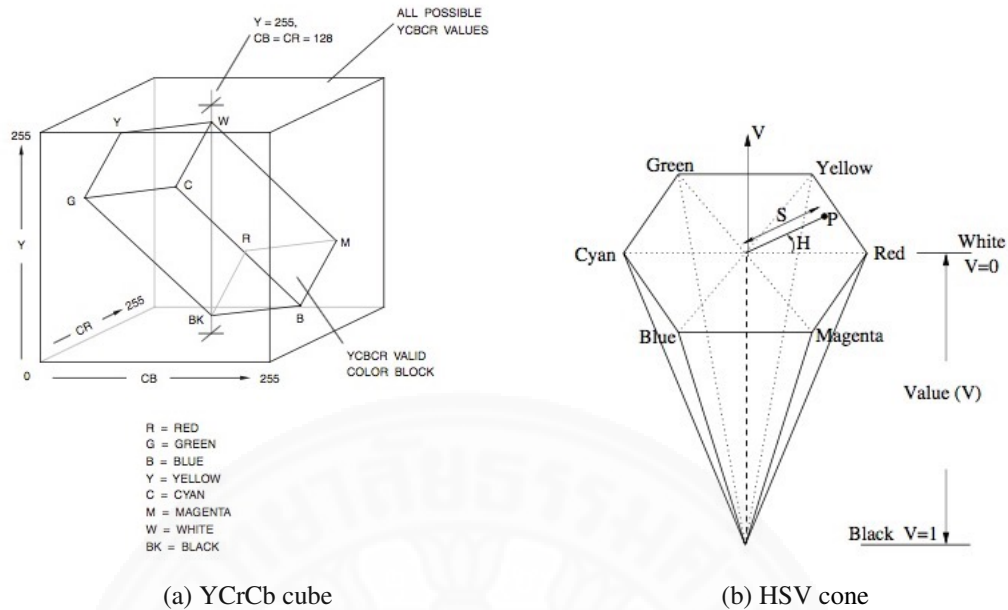


Figure 2.8 The YCrCb cube[22] and HSV cone [23].

HSV color space is designed based on human perception and interpretation of color [22]. It defines hue as a circle of colors, where each color is represented by a degree of spectral composition. Saturation is used to refer to the proportion of light and wavelength. Value corresponds to the intensity. The result is a cone-shaped color space shown in Fig. 2.8(b). Several researches explore the use of HSV for hand segmentation. Kook-Yeol Yoo [24] presents a hand segmentation method that is robust to illumination by applying the HSV color space with Y or luma. An adaptive hand segmentation method was proposed in [25] that combines HSV with TSL color space, which gives an advantage because of the wide difference between skin and non-skin color values. It improves the number of false positives by HSV color thresholding. In [48], HSV was selected for hand gesture recognition because hue is a normalized color and value can be used to distinguished between light and dark colors.

### 2.2.1.2 Background Subtraction

Background subtraction is a method for extracting a moving foreground object from a stationary background. The concept is based on the difference between frames that is measured by a distance function, such as sum-of-absolute-difference (SAD), sum-

of-squared-difference (SDD), and the Chebyshev distance according to (2.3)-(2.5). An extensive discussion of background subtraction methods can be found in [4] and [27].

$$B_{SAD} = |I_{Bg} - I_{Curr}| > SAD_{th} \quad (2.3)$$

$$B_{SSD} = (I_{Bg} - I_{Curr})^2 > SSD_{th} \quad (2.4)$$

$$B_{Chebyshev} = \max |I_{Bg} - I_{Curr}| > Chebyshev_{th} \quad (2.5)$$

where  $B_{SAD}$ ,  $B_{SSD}$ , and  $B_{Chebyshev}$  represent the difference in intensity values of each pixels between the background image  $I_{Bg}$  and the current image  $I_{Curr}$  as measured by the SAD, SSD, and Chebyshev distance functions, respectively. If the difference is greater than the threshold values specified by  $SAD_{th}$ ,  $SSD_{th}$ , or  $Chebyshev_{th}$ , then the pixel is considered a foreground object. This can be combined with color segmentation to improve the result, as [1] showed by combining SAD with HSV to produce a hand segmentation method that is robust under variable lighting conditions.

Alternatively, median or average background subtraction is an adaptive background subtraction method that considers the average value of  $N$  previous frames as the background image before applying subtraction:

$$I_{Bg,avg} = \frac{1}{N} \sum_{i=0}^N I_{Prev,i} \quad (2.6)$$

$$B_{Median} = |I_{Bg,avg} - I_{Curr}| > Median_{th} \quad (2.7)$$

where  $I_{Bg,avg}$  is the average intensity value of  $N$  previous frames and  $I_{Prev,i}$  is the intensity value of the previous frame at frame  $i$ .  $B_{Median}$  is the binary image output of median background subtraction when the difference between  $I_{Bg,avg}$  and  $I_{Curr}$  is greater than a specific threshold  $Median_{th}$ . Due to the calculation of average intensity values from  $N$  previous frames, this method requires more computational resources than conventional background subtraction methods.

Lastly, [42] presents a background subtraction method using unit-gradient-vector (UGV). UGV offers improved robustness and performance over conventional background subtraction methods under illumination changes by using intensity gradient normalization. Sobel edge detection is applied to both the background and

current frames to determine the intensity gradient vectors along the x- and y-axis:

$$I_x = I \cdot \text{Sobel operation mask in x-axis} \quad (2.8)$$

$$I_y = I \cdot \text{Sobel operation mask in y-axis} \quad (2.9)$$

The norm ( $n_x$  and  $n_y$ ) of both the background and current frames are then calculated as follows. It is sometimes necessary to avoid the divide-by-zero condition by setting a minimum value for the denominator.

$$n_x = \frac{I_x}{\sqrt{(I_x)^2 + (I_y)^2}} \quad (2.10)$$

$$n_y = \frac{I_y}{\sqrt{(I_x)^2 + (I_y)^2}} \quad (2.11)$$

The difference between the norm of the background ( $n_{Bg}$ ) and the current ( $n_{Curr}$ ) frames is determined by background subtraction along both the x- and y-axis:

$$d_x = n_{Bg,x} - n_{Curr,x} \quad (2.12)$$

$$d_y = n_{Bg,y} - n_{Curr,y} \quad (2.13)$$

The Euclidian distance is used to calculate the resulting value of UGV background subtraction. If the distance exceeds a certain threshold value, then the pixel is classified as a foreground pixel. Furthermore, UGV can be combined with HSV hand segmentation to improve robustness to dynamic lighting conditions [31].

$$d_{xy} = \sqrt{(d_x)^2 + (d_y)^2} \quad (2.14)$$

### 2.2.2 Finger Detection

Finger detection involves the determination of the tip or center of each finger by using mathematical models. This yields the position and number of fingers that can then be used to classify hand gestures. Generally, the basic methods for finger detection are convex hull, K-curvature, and center of gravity distance measurement.



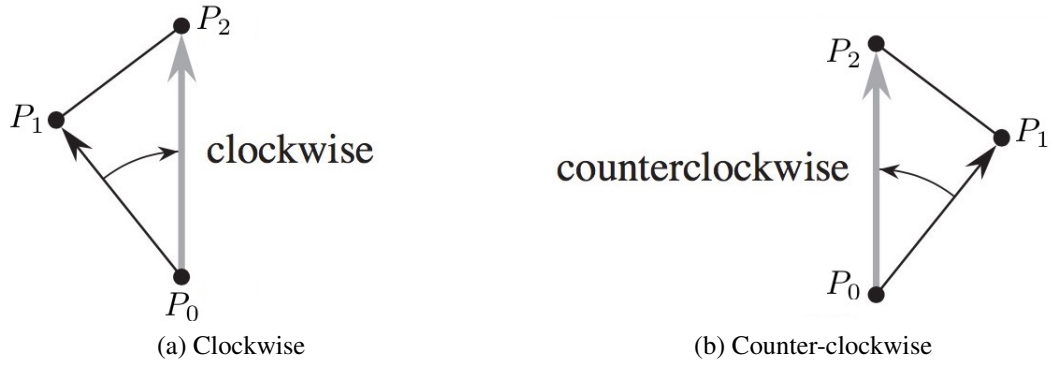


Figure 2.9 Example of clockwise and counter-clockwise direction with three reference points [41].

### 2.2.2.1 Convex Hull

The convex hull method was developed to determine the point or number of polygonal contour images by counting the number of vertex and applying the cosine law. The definition of a convex hull is the smallest convex polygon that encloses a set of points [41]. The OpenCV library utilizes a convex hull algorithm based on Graham's method that scans in the counter-clockwise direction around three reference points. From Fig. 2.9, the cross-product between  $\overrightarrow{P_0P_1}$  and  $\overrightarrow{P_0P_2}$  is obtained as:

$$\overrightarrow{P_0P_1} \times \overrightarrow{P_0P_2} = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0) \quad (2.15)$$

where  $\overrightarrow{P_0P_1}$  is the direction vector of  $P_0$  to  $P_1$ ,  $\overrightarrow{P_0P_2}$  is the direction vector of  $P_0$  to  $P_2$ ,  $P_0(x_0, y_0)$  is the common end point,  $P_1(x_1, y_1)$  is the starting reference point, and  $P_2(x_2, y_2)$  is the new reference point. The cross-product of  $\overrightarrow{P_0P_1}$  and  $\overrightarrow{P_0P_2}$  determines the clockwise or counter-clockwise direction by a positive or a negative value, respectively, as shown in Fig. 2.9.

A counter-clockwise scan is started with the minimum coordinate along the y-axis set as the starting point  $P_0$  to  $P_m$  with non-left turn, where  $m$  is the total number of points of interest. If more than one point has the same angle, the farthest point from  $P_0$  is selected and the others are removed. This process is repeated until a convex hull is formed, as shown in Fig. 2.10.



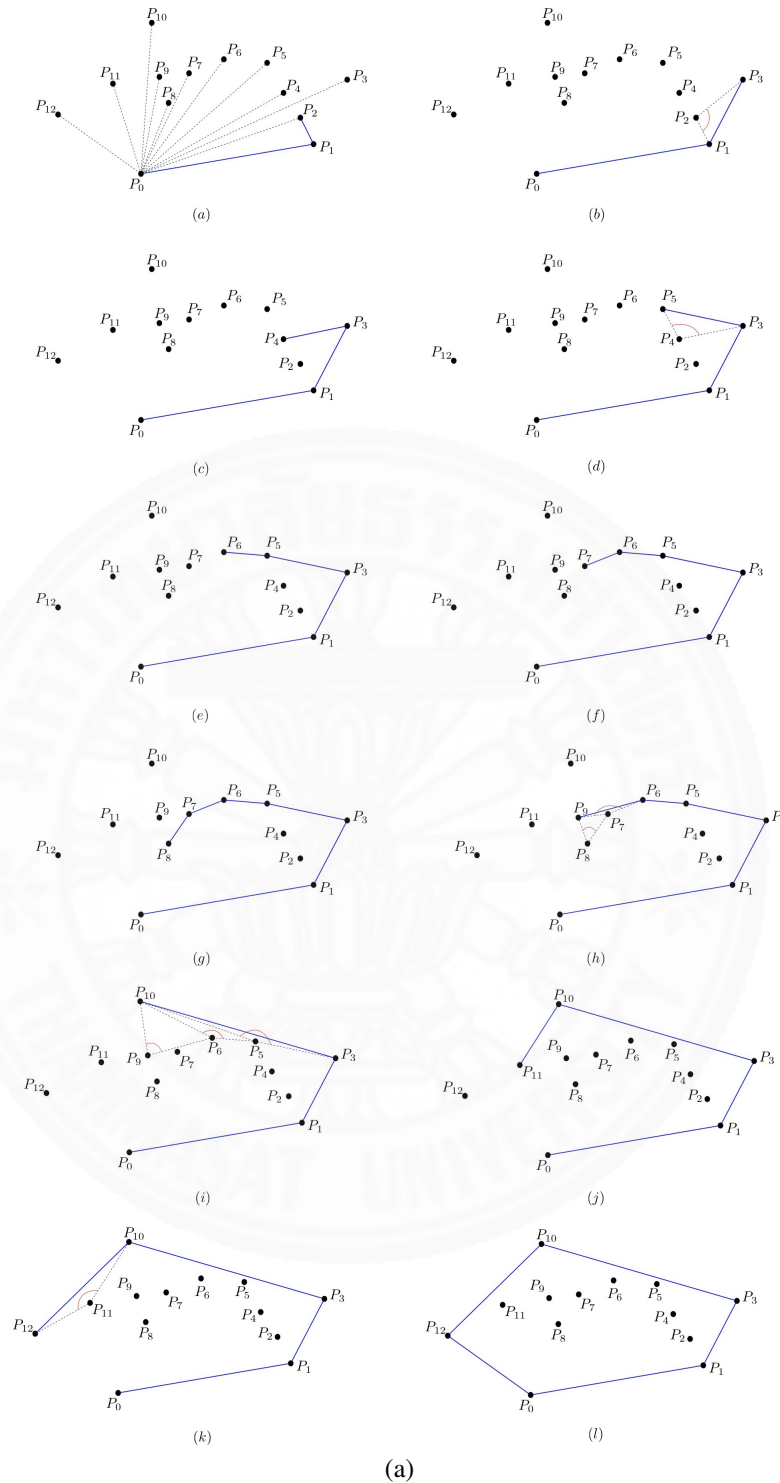


Figure 2.10 Determining a simple convex polygon [41].

Convex hull has been applied successfully for hand gesture recognition in [20] and [32]. It provides the position of each finger tip, as shown in Fig. 2.11, which can then be used to calculate the distance to the center of the palm for finger detection. However,

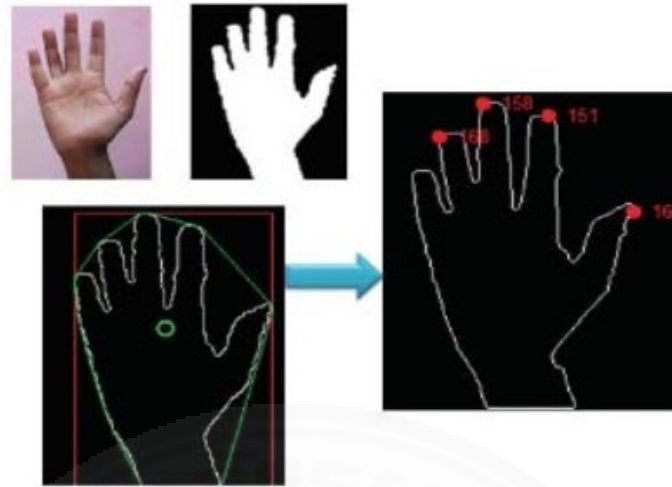


Figure 2.11 The convex hull method for finger detection [20].



Figure 2.12 Example error of convex hull method for finger detection [20].

convex hull cannot accurately determine certain hand gestures, such as a fist or when multiple fingers have the same planar angle. For example, when convex hull is applied to a fist, the knuckles are detected as multiple finger tips, as shown in Fig. 2.12.

#### 2.2.2.2 K-Curvature

The K-curvature algorithm starts from searching for the corner point of a contour and then computing the angle between two vectors that have a common end at  $P_i$  through  $P_{i+k}$  and  $P_{i-k}$ .  $k$  is a constant value corresponding to a specific magnitude of  $P_i$  to  $P_{i+k}$  and  $P_{i-k}$ , as show in Fig. 2.13. The angle is computed by the cosine law as:

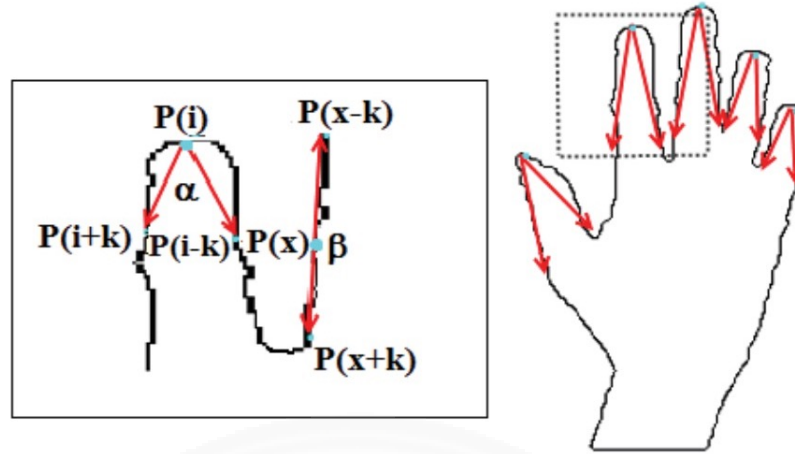


Figure 2.13 The K-curvature method for finger detection [20].

$$a^2 = b^2 + c^2 - 2bc \cos(\alpha) \quad (2.16)$$

$$m_k^2 = m_{i+k}^2 + m_{i-k}^2 - 2m_{i+k}m_{i-k} \cos(\alpha) \quad (2.17)$$

$$m_k^2 = k^2 + k^2 - 2k^2 \cos(\alpha) \quad (2.18)$$

$$\alpha = \arccos\left(\frac{2k^2 - m_k^2}{2k^2}\right)$$

where  $a$ ,  $b$ , and  $c$  are the opposing sides of  $\alpha$ ,  $\beta$ , and  $\gamma$  angles.  $k$  is a constant value.  $m_{i+k}$ ,  $m_{i-k}$  are the magnitude of the vectors  $\overrightarrow{P_i P_{i+k}}$ ,  $\overrightarrow{P_i P_{i-k}}$  and  $\overrightarrow{P_{i+k} P_{i-k}}$ . Setting the specific value of  $k$  to tracking the common end point of two vector with  $k$  magnitude will determine the peak of contour as finger and valley. Then using position classifier to separate the finger and the valley as detected  $P_i$ . An example of K-curvature is shown in Fig2.13 [20].

### 2.2.2.3 Center of Gravity Distance Measurement

The center of gravity method determines the center of each object by using the geometric moments [26]:

$$M_{pq} = \iint_{\Omega} x^p y^q I(x,y) dx dy \quad (2.19)$$

where  $I(x,y)$  is the intensity value of the image at position  $(x,y)$ ,  $M_{pq}$  is the  $(p,q)^{\text{th}}$  moment of  $I(x,y)$ . In digital image processing, the double integration can be

approximated by a summation function as:

$$\tilde{M}_{pq} = \Delta^2 \sum_{i=0}^n \sum_{j=0}^n x_i^p y_j^q I(x, y) \quad (2.20)$$

where  $\tilde{M}_{pq}$  is the  $(p, q)^{\text{th}}$  approximated geometric moment.  $\Delta$  is the sampling interval as  $x_i - x_{i-1}$  and  $y_j - y_{j-1}$ . The zero-order of the geometric moment gives the total mass as  $M_{00}$ . The first-order gives  $M_{10}$  and  $M_{01}$ , which can be divided by  $M_{00}$  to obtain the center of mass as  $\tilde{x}$  and  $\tilde{y}$ :

$$\tilde{M}_{00} = \Delta^2 \sum_{i=0}^n \sum_{j=0}^n I(x_i, y_j) \quad (2.21)$$

$$\tilde{M}_{10} = \Delta^2 \sum_{i=0}^n \sum_{j=0}^n x_i I(x_i, y_j) \quad (2.22)$$

$$\tilde{M}_{01} = \Delta^2 \sum_{i=0}^n \sum_{j=0}^n y_j I(x_i, y_j) \quad (2.23)$$

$$\tilde{x} = \frac{\tilde{M}_{10}}{\tilde{M}_{00}} \quad (2.24)$$

$$\tilde{y} = \frac{\tilde{M}_{01}}{\tilde{M}_{00}} \quad (2.25)$$

The center of gravity can be applied for finger detection by determining the center of each finger and the center of the palm. The Euclidean distance and relative position of each finger to the palm can then be used to classify each finger. This method was shown in [32].

#### 2.2.2.4 Other Methods

Other finger detection methods have also been developed, including perimeter curvature, circle detection with pixel grouping, and middle axis method. A brief discussion of each method is provided below.

The perimeter curvature method was developed in [20]. The concept of this method is based on the eccentricity of the shape at the corner points. A low eccentricity has a shape that is closer to a circle, which is similar to the fingertips. For this reason, it can be used to identify the location of each finger. This method starts by

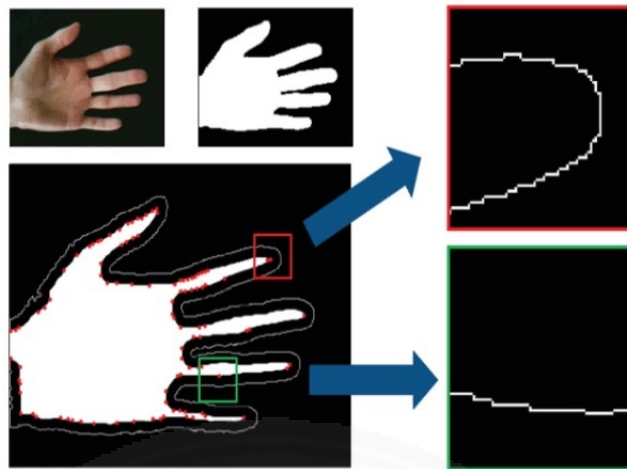


Figure 2.14 The perimeter curvature method for finger detection [20].

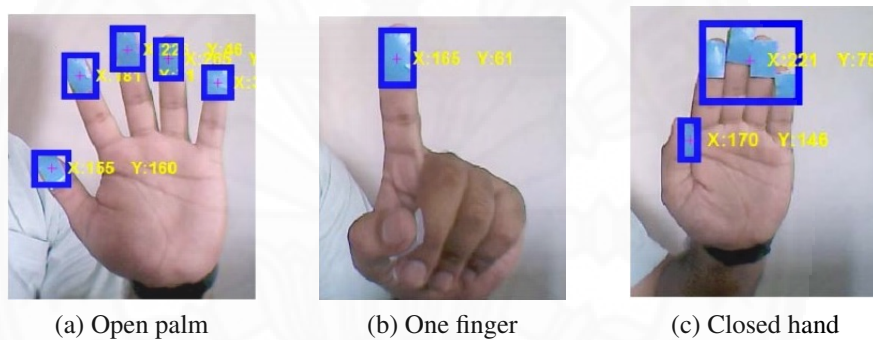


Figure 2.15 Circle detection with pixel grouping [16].

eroding the image obtained from hand segmentation using distance transform, then finding the perimeter of the hand. The position of the corner can be found by the output of distance transform. The section of the detected corner is cropped to calculate the perimeter and the eccentricity of the corner point, as shown in Fig. 2.14.

The circle detection with pixel grouping method identifies a finger by examining groups of pixels that have a circular structured element. If the number of pixels in each group is lower than a certain threshold, it is classified as a fingertip. This method is sensitive to changes in distance between the user and the camera, as the distance effects the size of the object pixel, leading to false detection error. The experiment using this method was presented in [16] and shown in Fig. 2.15.

Lastly, the middle axis method scans the vertical position of the hand segmentation image to determine the middle axis of each finger, as shown in Fig. 2.16.

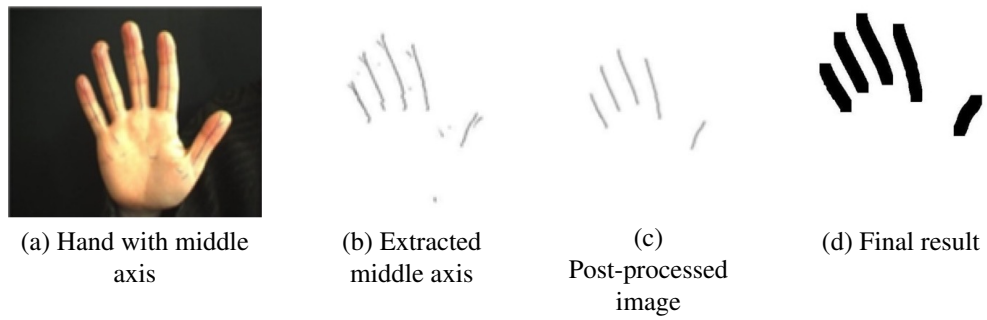


Figure 2.16 Middle axis method [18].

It then filters the image for specific row distances that corresponds to the finger width. The mean value is determined and marked as the middle axis. Morphological erosion and dilation is performed to obtain the final result. This method is a simple algorithm for counting the number of fingers [18].

## Chapter 3

### System Design

This chapter discusses the design of the proposed remote control system for an mp3 player using hand-gesture recognition. Firstly, application design and related concepts are presented, including an explanation of system inputs and outputs. The system hardware consists of an embedded controller and a camera module. The required libraries and application-programming interface (API) are presented as part of the software design. Finally, the environment setting is discussed, including the region of interest used for image processing and the distance between the user and the system.

#### 3.1 Application Design

According to the motivation discussed in Chapter 1, the main concept that will be presented in this work is a real-time, stand-alone remote controller using robust hand-gesture recognition as part of a man-machine interface that can operate in dynamic lighting conditions. The system that will be controlled is an mp3 player or other media playback device.

The application design starts with a user who can position their left or right hand in various gestures, as shown in Fig. 3.1, which also describes the relationship between each hand gesture and the corresponding media control command. The gestures that are considered in the work are open palm (OP), forefinger (FF), fist (FS), forefinger and thumb (FT), leftward thumb (TL), and rightward thumb (TR). The open palm is used as an initialization/termination command to prompt media playback control. The forefinger starts media playback, the fist stops playback and exits the track, and the forefinger and thumb together pauses the media temporarily. The leftward and rightward thumbs are used to skip to the next track or return to the previous track, respectively.

The three main components of the system are the image processing hardware and algorithm, controller interface, and media playback functionality. It is important to design the software and hardware to achieve real-time operations.

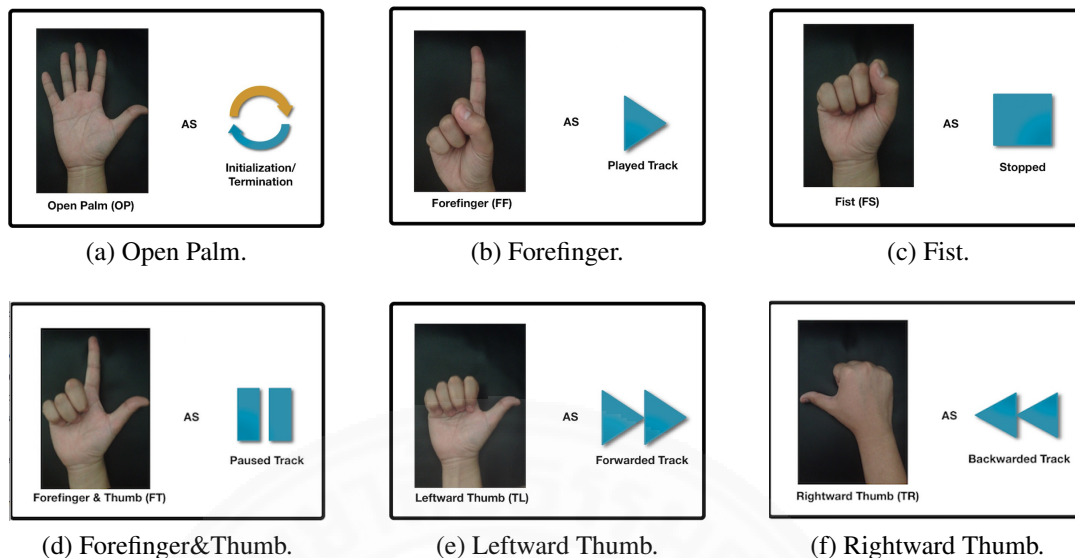


Figure 3.1 Hand gesture pattern for MP3 player.

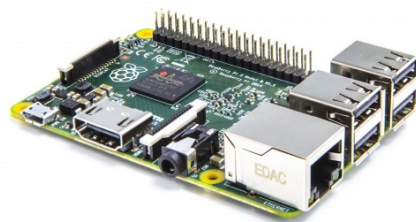
### 3.2 Hardware Design

Hand-gesture recognition softwares are usually implemented on personal computers with high-performance processors that support high-resolution images via a camera. These computers are invariably of large size, heavy weight, and high power consumption. Alternatively, this research requires a small, portable, and power-efficient hardware. An embedded system would fulfill these requirements.

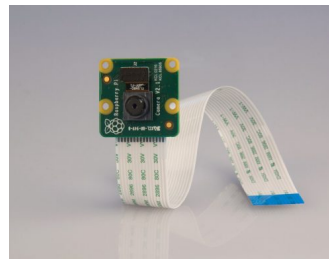
Currently, two types of cameras can be used for hand-gesture recognition: monocular and stereo. Monocular cameras have a single lens that produces a 2-dimensional image. Stereo cameras have two lens that can be used to construct 3-dimensional images, containing depth information. As a 3-dimensional image would contain more data than a 2-dimensional image, the resulting file size is also much larger. The size of the image file has a significant effect on the computational time of any image processing algorithm as it would require longer execution. Therefore, this work chooses to employ a monocular camera, because the image size is more suitable for processing on an embedded system in real-time.

The hand-gesture recognition algorithm will be implemented on an embedded system with a main controller unit (MCU) that will handle all image processing tasks and media playback functions. The MCU will also interface with a monocular camera





(a) Raspberry Pi2 Module B [13]



(b) Raspberry Pi camera [12]

Figure 3.2 Raspberry Pi2 Module B and Pi camera.

to acquire the input image and display the results on an external monitor. The hardware model used in this study is the Raspberry Pi2 Module B and its compatible Raspberry Pi camera. More details and other specifications will be provided in the following subsection.

### 3.2.1 Raspberry Pi2 Module B

Raspberry Pi is a popular, low-cost embedded platform running Debian LINUX operating system. This work uses the Raspberry Pi2 Module B, shown in Fig. 3.2(a), from the Raspberry Pi family because it was the newest module and highest performance model at the time. This module includes a 900MHz, 32-bit quad-core ARM Cortex-A7 processor and 1GB of RAM. Due to the quad-core processor, it is possible to make use of the multi-core architecture for this application by multi-thread parallel processing to decrease execution time. Other features of Raspberry Pi2 Module B are provided in [39].

### 3.2.2 Raspberry Pi camera

The Raspberry Pi camera, shown in Fig. 3.2(b), is a high-definition camera that is compatible with all of the Raspberry Pi family. It uses an Omnivision 5647 CMOS image sensor with fixed-focus lens and integrated IR filter, and has a resolution of 5-megapixel. The maximum still-image resolution that it can capture is  $2592 \times 1944$ . For video streaming, it can capture 1080p at 30 frames per second (fps), 720p at 60 fps, and  $640 \times 480$ p at 60 to 90 fps.



Figure 3.3 32 GB Micro SD card for Raspberry Pi2 module B

### 3.2.3 Expand memory of Raspberry Pi2 module B

The memory of Raspberry Pi2 module B is informed of the external 8 GB micro SD card as the minimum requirement. The memory card used to support the OS, the open-source library component and memory space. To prevent the memory run out, the author has expanded the memory size to a 32 GB micro SD card. The compatible memory card for Raspberry Pi can see in [14]. Thus using the 32 GB Kingston micro SDHC 10/32 GB 314336-016 A00LF Taiwan card for the implementation in this work as shown in Fig 3.3.

After the installation of OS to the memory card, then going to Raspberry Pi configuration then "Expand File system" command and reboot. Then the OS will set the memory space as 32 GB instead of the original memory size 8 GB.

### 3.3 Software Design

The software design section describes the program and system architecture of this work. The program structure consists of the operating system, application programming interface (API), and required libraries. The system architecture shows the block diagram of this work and its synchronization. Both are shown in Fig. 3.4.

The operating system used in this work is Wheezy Debian LINUX that supports C and C++ programming languages through the CMake compiler under MinGW. The details of CMake and MinGW can be found in [8] and [28]. The API necessary for interfacing between the MCU and camera is based on Userland library. This research also employs OpenCV 3.0 beta library for basic image processing functions. Media playback functionalities on the mp3 player are handled by LibAO and LibMPG123. The last component is OpenMP library for multi-thread programming in C and C++.

The installation of Userland API and OpenCV follow the instruction in [36]. Although, this instruction is not compatible with the latest version OpenCV 3.0 beta thus it requires to debug the protocol between them. Besides, video recoding and reading with OpenCV need an additional library as FFMPEG and LibFAAC. Furthermore, OpenCV allows OpenMP for multi-thread purpose. Thus it should be reconfigured and reinstalled.

The system architecture is shown in Fig. 3.4. The three main components are the image processor, control interface, and mp3 player. The image processor uses both streaming and still images from the camera output for hand-gesture recognition. The outputs of the image processor are the six recognizable hand postures. The control interface acts as an intermediary between the image processor and the mp3 player, and is constructed as a finite state machine shown in Fig. 3.5. This work defines two states as idle and recognition. The system is initialized in the idle state and waits for the open palm gesture to transition to the recognition state. While in the recognition state, the algorithm will process the recognized hand posture to control the mp3 playback. The machine will return back to the idle state when another open palm gesture is detected.

### **3.4 Acquiring an Image from Camera**

As know, Raspberry Pi camera is compatible with Raspberry Pi2 module B and can be accessed to capture the image directly from the command line function. The command line function is not flexible to implement and convenient for image processing. In this work, the implementation uses the specific library as Userland API to corresponds with OpenCV library. The acquiring image from the camera by using both library shows as a flow chart in 3.6.

First, the initialize step Multi-Media Abstraction Layer(MMAL) port to prepare the camera connection which includes video and still image camera port. Setting the capturing image that consists of resolution of the image, frame rate, bitrate and capturing time as presents in Table 3.1. Camera setting for this implementation is set as the default of Raspberry Pi camera. After that, Raspberry Pi camera capture and sent the image data as YUV color space in three channels to the callback buffer then

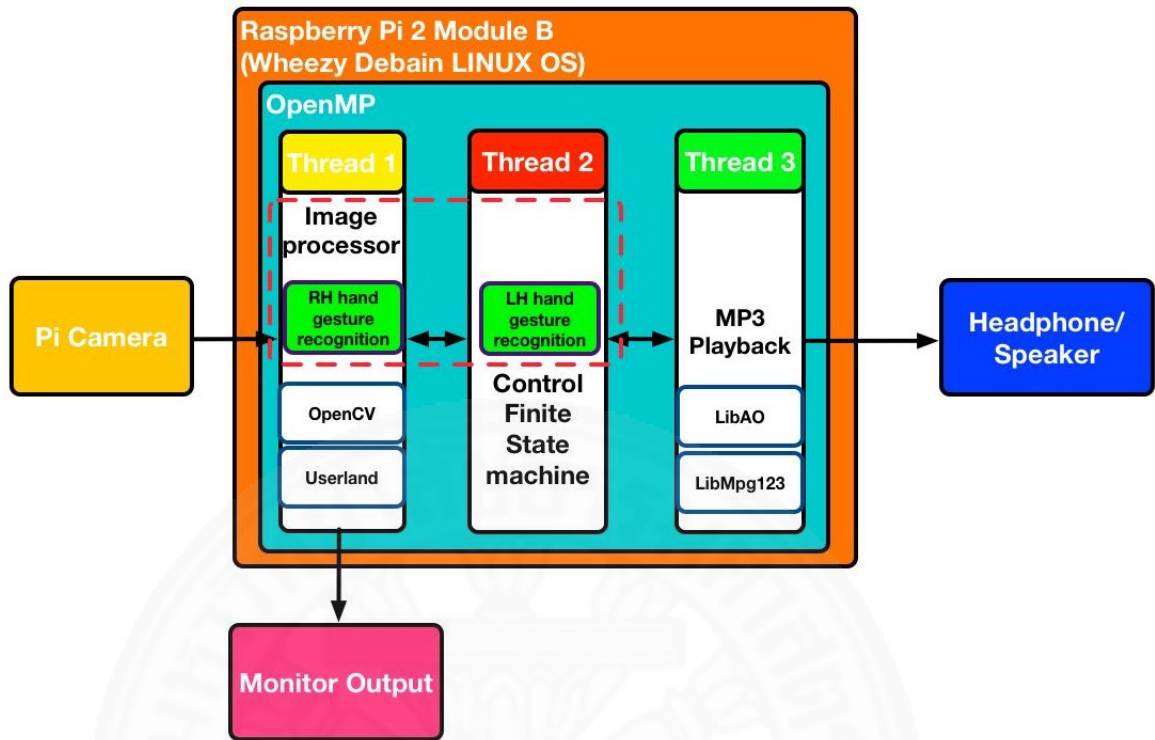


Figure 3.4 Software design flowchart.

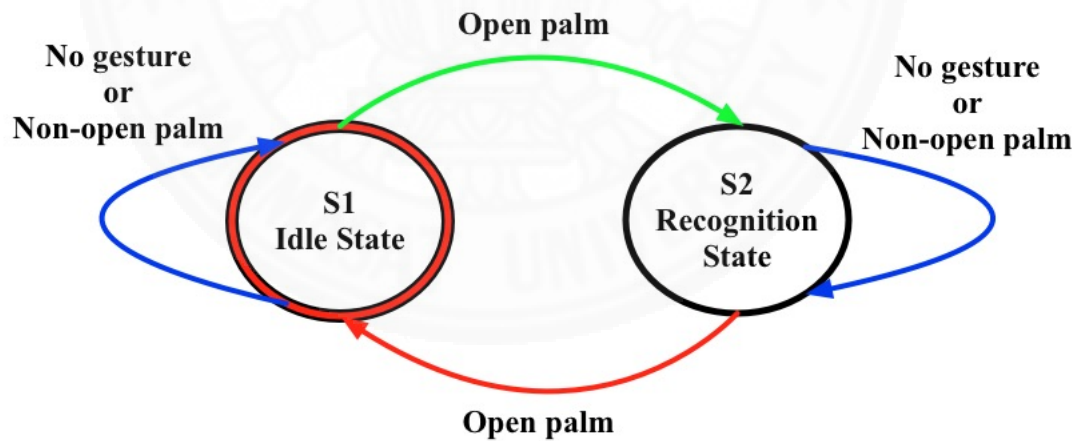


Figure 3.5 Control finite state machine.

convert the data to a matrix with OpenCV. Matrix format gives the comfortable to work with the standard image processing function of OpenCV. Also, the image processing can be implemented on this buffer as frame by frame computation and send the output matrix image to the video port. At the end of capturing time, the port and component will disable after video capture is finished.

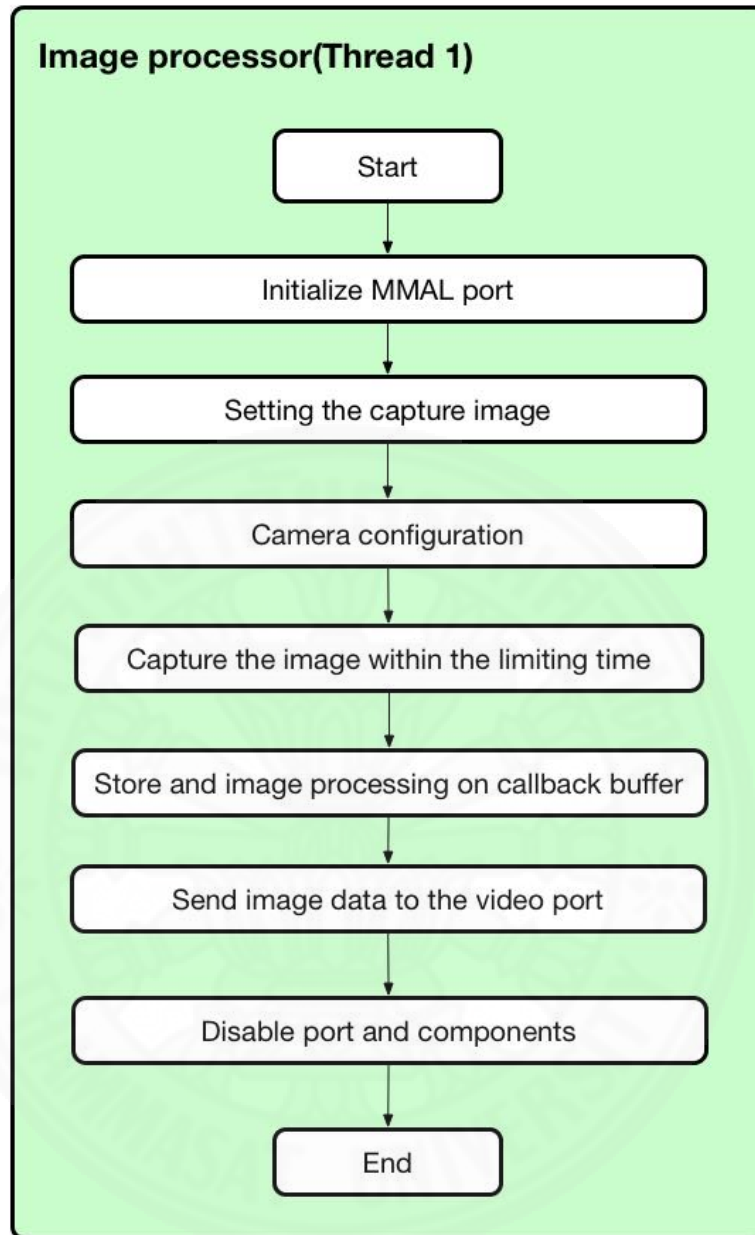


Figure 3.6 The image acquisition process by using Userland and OpenCV

Table 3.1 The setting parameter for capturing the image.

Element Setting	Value
Image width	320
Image height	240
Frame rate	30
Bitrate	17,000,000
Capturing time	36000 ms

### 3.5 Environment Setting

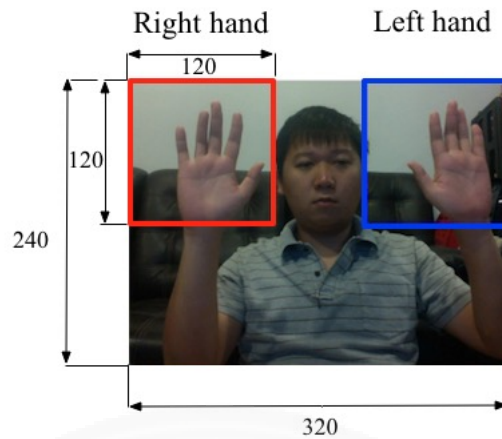
The experiment will be conducted under three lighting levels in a room with fluorescent lamps that can be adjusted for bright, dim, and dark. This will be used to study the robustness of the hand-gesture recognition algorithm under dynamic lighting conditions. More details of the lighting conditions are given in Chapter 6. The image resolution is limited to  $320 \times 240$  in order to achieve real-time processing at 30 fps using Raspberry Pi2 Module B with Userland API. Two regions of interest (ROIs) are defined for both the left and right hands, where the left ROI is located between (200,0) to (320,120) and the right ROI is located between (0,0) to (120,120). The left and right ROIs are indicated by red and blue boxes in Fig. 3.7(a). Each region has a size of  $120 \times 120$ , which corresponds to an approximate distance between the hand and camera of 50 to 80 centimeters. The distance between the user and the camera is shown in Fig. 3.7(b). Furthermore, this resolution helps to ensure that the hand gesture recognition can be completed faster and in real-time than higher resolution images.

### 3.6 Multithreading Implementation

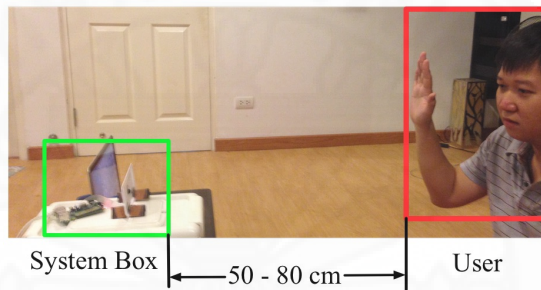
From the software design present the benefit to using the multithreading programming with OpenMP to process three module that including of image processing, controller interface and MP3 player independently. To design, the multithreading application should be concerned about the number of thread, shared variable and global variable. The number of thread is the number of processors that use to execute the program. Shared variable is usually used as the shared input data that for each thread but the global variable can as both as input and output.

The implementation of multithreading in this work start from enabling the parallel processing by develops under the “#pragma omp parallel”. Then allow the multithreading with independent execution with “#pragma omp section nowait” after that implement the image processing, controller interfaces and MP3 playback in a different thread under each of “#pragma omp section”. Finally, three thread will join back to process the serial processing again at the end of parallel processing section.





(a) Region of interest



(b) Distance between user and camera

Figure 3.7 Region of interest and the distance between the user and the camera.

When applying the multithreading to the image processor, it can perform to recognize two hand gesture recognition simultaneously by parallel processing on two threads. The process is starting at the acquiring the image and pre-image processing with serial processing then fork two regions of hand image two thread and apply the hand gesture recognition. Joining them together at the end of the algorithm then send the output to the controller interface which locates in the second thread.

### 3.7 Controller Interface

Controller interface is the connector between the hand gesture recognition part and MP3playback part as introducing in Chapter 3. The implementation of its is based on the control finite state machine which consists of two states as an idle state and recognition state and show in Fig3.8. In general, the finite state machine is always implemented inform of a switch-case statement by setting state of finite state machine

as the condition.

At the idle state, it is starting with reset the hand posture input then giving the delay in 1 second. Next is the OP classification to transmute the hand gesture input to enter the state of machine. Then, the system will go to recheck the OP hand posture again with additional delay as same as the concept of software debounce [10]. The software debounces using to confirm the input is correct in specific time. If the input is not always as OP until the recheck, the state will stay at idle state else the state will transit to the recognition state.

When entering to the recognition state, the input should be reset to prompt the new incoming input for MP3. The construct of the recognition state is more complex when compared to the idle state. Because of this state decide to six hand postures which are OP, FF, FT, FS, TL, and TR to response to six MP3player commands as initialization/termination, play, pause, stop, FWD, and BWD. The construction of them also using the software debounce but the state stays at recognition state instead except the OP which goes back to the idle state.

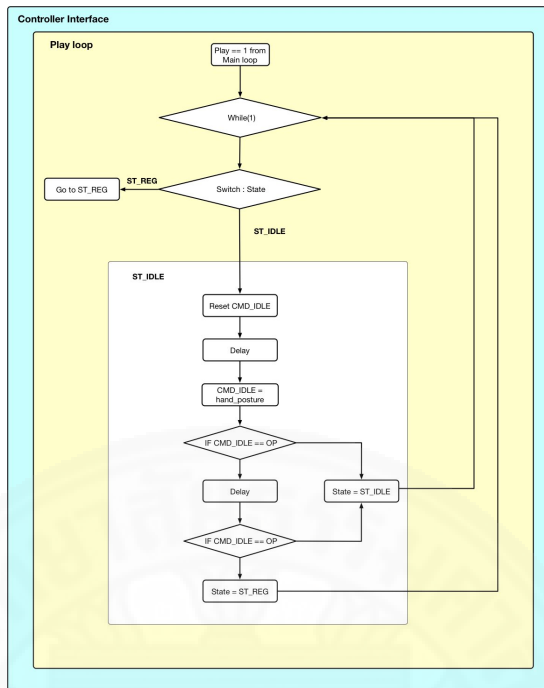
### **3.8 MP3playback Application**

This thesis has implemented the MP3 player which has six commands as initialization/termination, play, pause, stop, forward track(FWD) and backward track(BWD) to correspond to the designed six hand postures. The software flow chart of the music player is presented in 3.9(a) to (c).

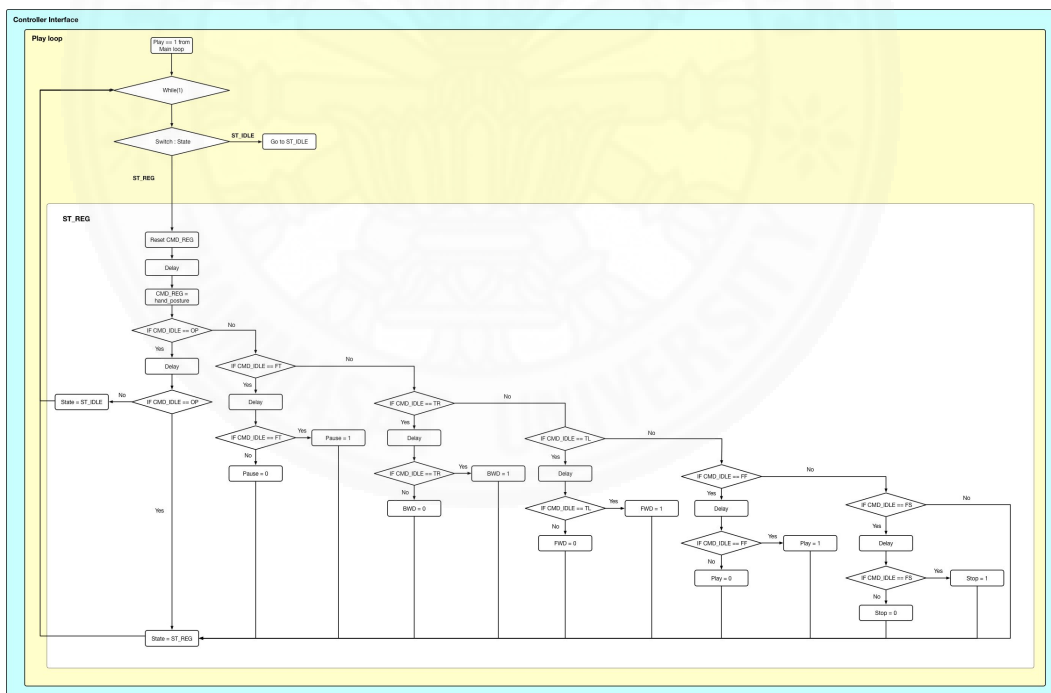
Following the flow chart has designed in three main loop as MP3player-loop, play-loop, and pause-loop. Starting with the main loop as MP3player-loop, this loop is the infinite loop to provide the MP3player independently run at all time on the third thread. The main loop has initialized the MP3playback at first then checking play command if play the system will go to play-loop else still at this loop respectively. Furthermore, this loop allows the user to FWD or BWD the track during play command does not activate.

In play-loop, it starts with the initialization of the music track before the playingtrack-loop inside. The playingtrack-loop has a duty to play music track until





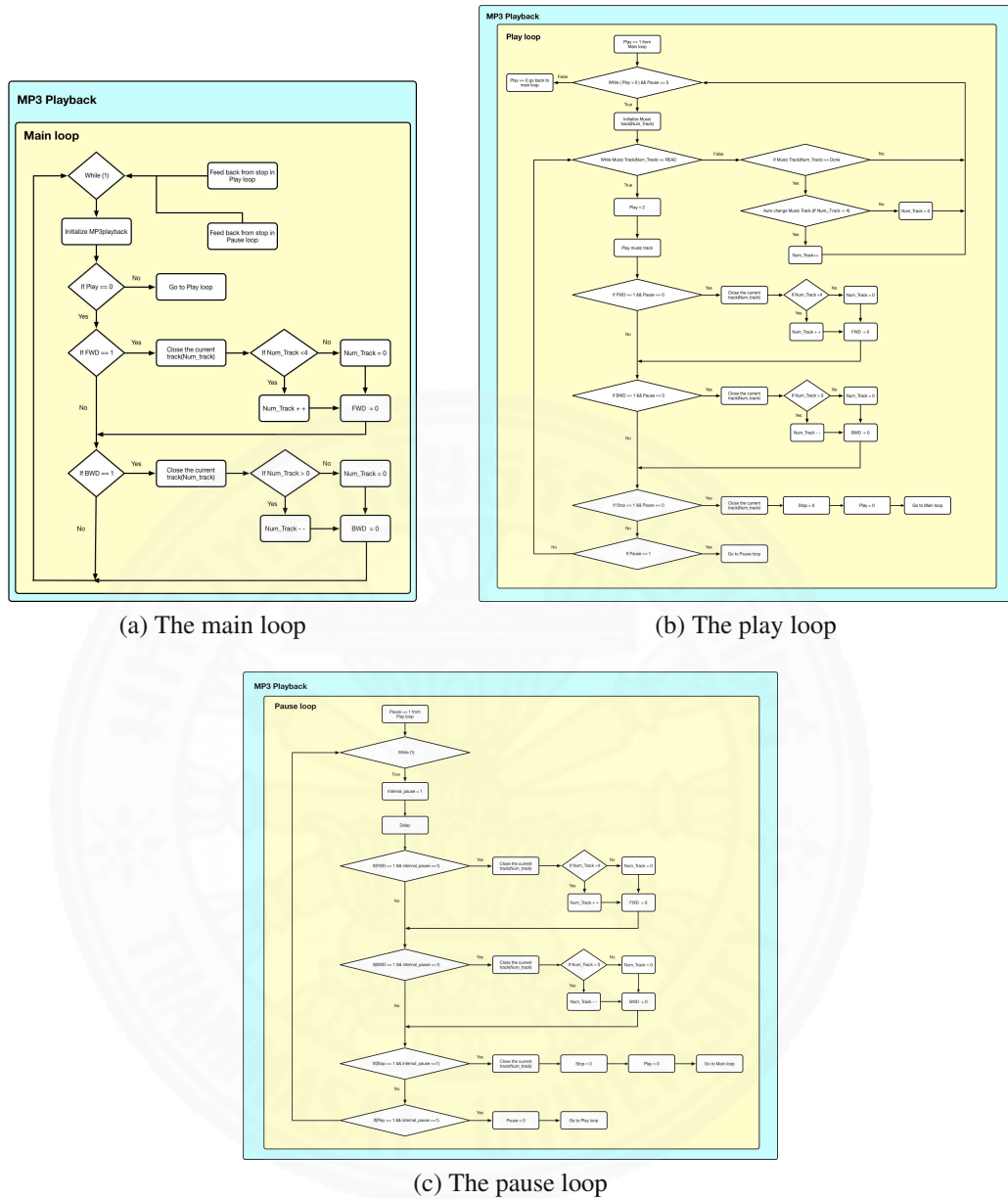
(a) The Idle state



(b) The recognition state

Figure 3.8 Controller interface software flow chart

the music has ended or has the interrupt of MP3 player command. If the music track was ended, the author design the automatical next track as the general music playback. Discussing of the interrupt command in the playingtrack-loop, it can be divided in two



(a) The main loop

(b) The play loop

(c) The pause loop

Figure 3.9 The MP3player flow chart.

conditions as non-pause and pause command. Non-pause command is including of FWD, BWD, and stop the music track. FWD and BWD will close the track first then break the playingtrack-loop through the play-loop to initialize a new music track which supports FWD or BWD command then play the track. On the others hand stop command is different, stop command will close the track then go out the play-loop back to the main loop. Pause command will jump to the pause-loop but this loop still inside the playingtrack-loop then paused the music track at that time.

Last, the pause-loop is the loop which the music has temporary stop playing until the first input command as play command to run the music track at that time again. If the receiving command is not played such as pause, stop, FWD, and BWD. The system does not react to the pause command during pause-loop is active. Also to the playingtrack-loop, stop command will close the track then exit the pause-loop to the main-loop. FWD and BWD will close the song as same as in playingtrack-loop, but it is still in the pause loop thus the music can not play until the play command is received.



## Chapter 4

### Hand Segmentation

Hand segmentation is an integral part of this research that allows the relevant hand region to be extracted from the image. This begins with pre-processing of the raw, intensity-based image obtained from the camera. Color segmentation is then used to identify pixels with skin color information, and the performance for different color spaces are compared under various lighting conditions. Background subtraction using both conventional sum-of-absolute-difference (SAD) and proposed unit-gradient-vector (UGV) is also used to isolate the hand region. The results from color segmentation and background subtraction are then combined together using the AND operation to obtain the complete hand image. In the final step, post-processing is used to filter out false positives/negatives and smooth the image. Details of each procedure will be presented in this chapter.

#### 4.1 Overview of Hand Segmentation

The flowchart of hand segmentation is shown in Fig. 4.1 with the following sequence: pre-processing, color segmentation, background subtraction, AND operation, and post processing. It will be handled as part of the image processing software described in Chapter 3. Both right- and left-hand gesture recognition is supported, with each hand treated separately using multi-threading in OpenMP, such that Thread 1 and Thread 2 correspond to the right- and left-hands, respectively.

##### 4.1.1 Pre-Processing

Pre-processing is the initial step to prepare the image for color segmentation and background subtraction. It involves acquiring the original image from the camera, determining the region of interest (ROI), and conversion into an intensity-based image.

The original image is captured from the Pi camera using Userland API in YUV, at a resolution of  $320 \times 240$  and 30 FPS. YUV color space is widely used in cameras, analog, and digital television because it gives more natural colors for human vision

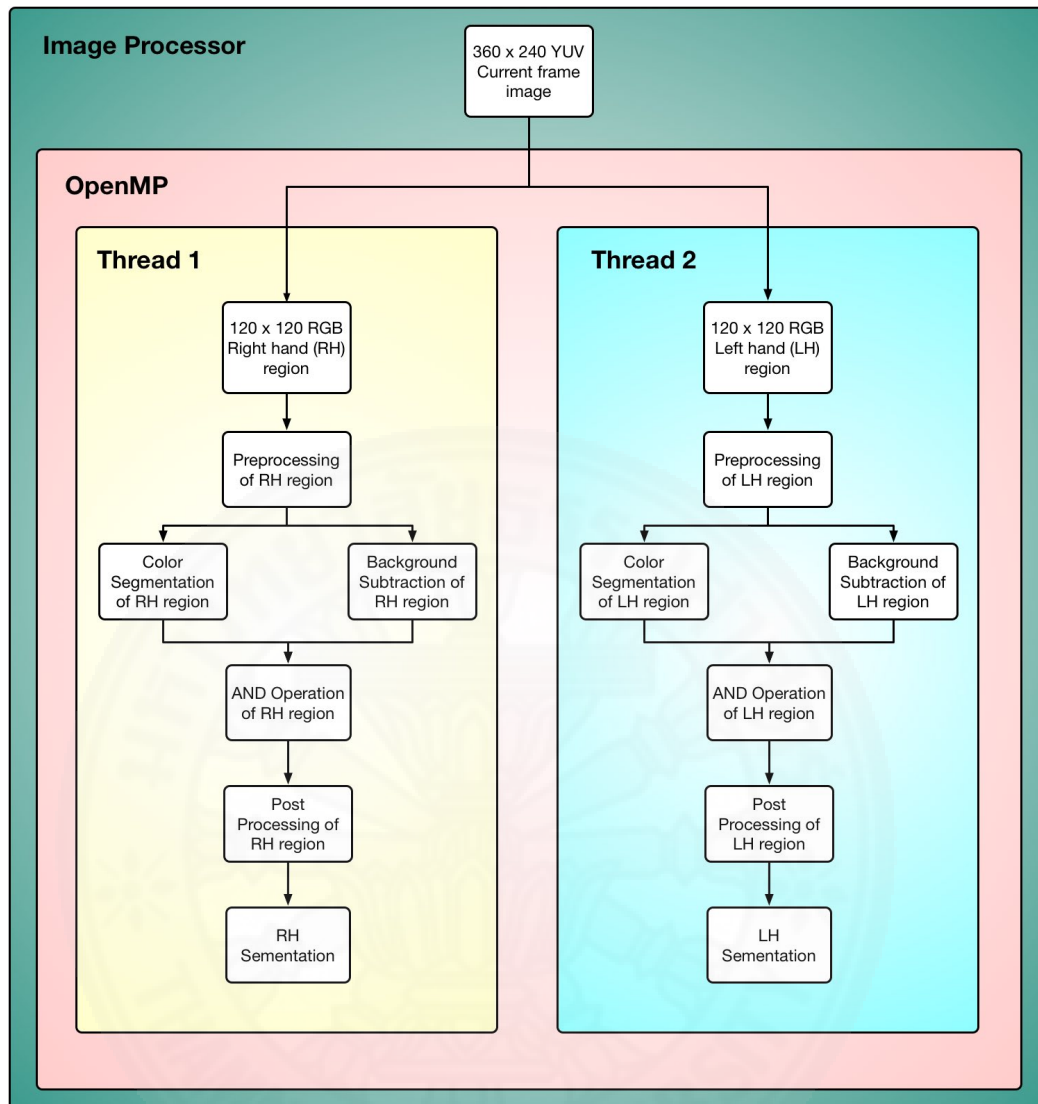


Figure 4.1 Hand segmentation flow chart

than RGB color space [22] It consists of three channels for luma or intensity-based image (Y), chrominance component (U), and color information (V). YUV color space is sometimes called YPbPr in an analog component video and YCrCb in a digital component video. UV are the deviations of gray in blue-yellow and red-cyan axes. It is important to first convert YUV to RGB in order to be compatible with OpenCV.

Two ROIs are defined according to Chapter 3 for the right- and left-hands, as shown in Fig. 4.2(a). Each ROI has a resolution of  $120 \times 120$  in the RGB color space. The hand segmentation and detection algorithm is similar for both regions, thus this work will proceed to discuss the proposed method only for the right-hand.

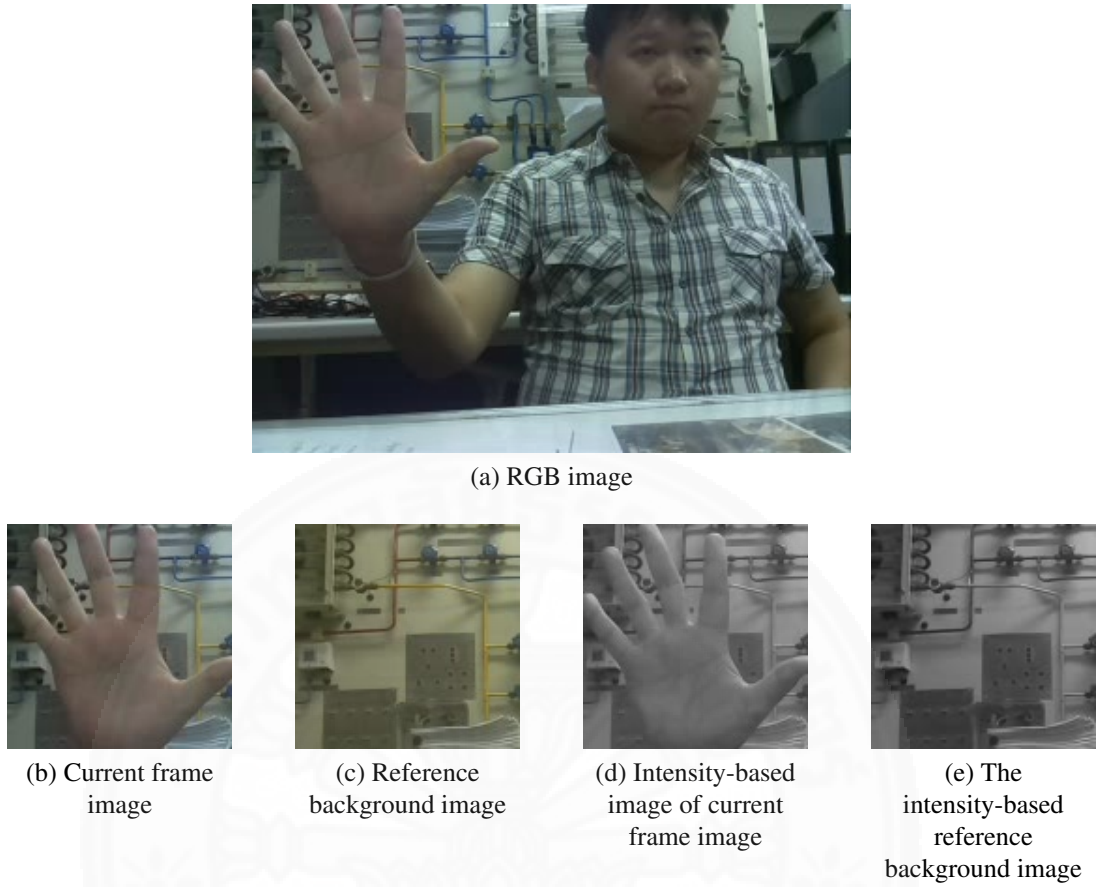


Figure 4.2 Pre-processing image.

The background and current RGB image frames are defined as  $F_{Bg}$  and  $F_{Curr}$ , respectively, as shown in Fig. 4.2 (b) and (c). It is also necessary to convert the RGB images to intensity-based images  $I_{Curr}$  and  $I_{Bg}$  using (4.1), which is referenced from[45]:

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (4.1)$$

where  $I$  is the intensity-based pixel value,  $R$  is the red-channel pixel value,  $G$  is the green-channel pixel value, and the  $B$  is blue-channel pixel value of the image.

#### 4.1.2 Color Segmentation

Color segmentation is used to extract the hand region from the frame by using human skin color in any of the color spaces. This work examines three different color spaces: RGB which is the most basic color space; YCrCb which is widely used for human skin detection; and HSV which has strong robustness to dynamic lighting

Table 4.1 Human skin threshold color in three different color space

Color Space	Channel	Value	
		Min	Max
RGB	$R_{th}$	45	255
	$G_{th}$	40	150
	$B_{th}$	40	150
YCrCb	$Y_{th}$	0	255
	$Cr_{th}$	77	127
	$Cb_{th}$	133	255
HSV	$H_{th}$	0	54
	$S_{th}$	10	255
	$V_{th}$	0	255

conditions. Therefore, it is necessary to convert the RGB image into YCrCb and HSV images before color segmentation is performed. The conversions are conducted using (4.2) and (4.3), as referenced in [29]:

$$\begin{aligned}
 Y &= 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \\
 Cr &= R - Y \cdot 0.713 + 128 \\
 Cb &= (B - Y) \cdot 0.564 + 128 \\
 f_{YCrCb} &= f(Y, Cr, Cb) \\
 V &= \max(R, G, B) \\
 S &= \begin{cases} V - \min(\frac{R, G, B}{V}) & , \text{if } V \neq 0 \\ 0 & , \text{Otherwise} \end{cases} \\
 H &= \begin{cases} \frac{60(G-B)}{(V-\min(R, G, B))} & , \text{if } V = R \\ \frac{120+60(B-R)}{V-\min(R, G, B)} & , \text{if } V = G \\ \frac{240+60(R-G)}{V-\min(R, G, B)} & , \text{if } V = B \end{cases} \\
 f_{HSV} &= f(H, S, V)
 \end{aligned} \tag{4.2}$$

$$\tag{4.3}$$

Each color space has different threshold values for human skin color segmentation. These values are shown in Table 4.1. The range of threshold values for each channel are represented as  $R_{th}$ ,  $G_{th}$ ,  $B_{th}$ ,  $Y_{th}$ ,  $Cr_{th}$ ,  $Cb_{th}$ ,  $H_{th}$ ,  $S_{th}$ , and  $V_{th}$ . The output of color segmentation is a binary image according to (4.4). If the values of a pixel are in the

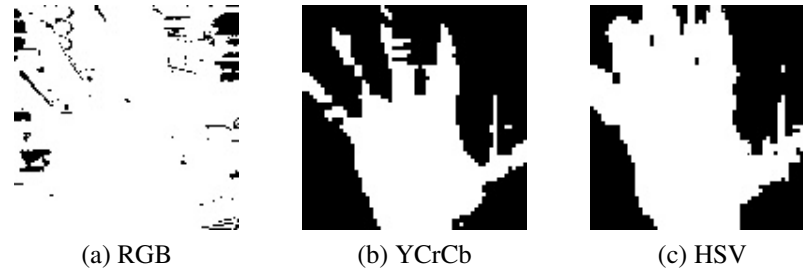


Figure 4.3 Human skin color segmentation

threshold range, the pixel will become white to indicate the detected pixel, otherwise it will become black. The outputs of color segmentation are shown in Fig. 4.3 for RGB, YCrCb, and HSV.

$$\begin{aligned}
 B_{RGB} &= \begin{cases} 255 & ,\text{if } R \in R_{th}, G \in G_{th}, \text{ and } B \in B_{th} \\ 0 & ,\text{Otherwise} \end{cases} \\
 B_{YCrCb} &= \begin{cases} 255 & ,\text{if } Y \in Y_{th}, Cr \in Cr_{th}, \text{ and } Cb \in Cb_{th} \\ 0 & ,\text{Otherwise} \end{cases} \\
 B_{HSV} &= \begin{cases} 255 & ,\text{if } H \in H_{th}, S \in S_{th}, \text{ and } V \in V_{th} \\ 0 & \text{Otherwise} \end{cases} \quad (4.4)
 \end{aligned}$$

where  $B_{RGB}$ ,  $B_{YCrCb}$ ,  $B_{HSV}$  are binary images resulting from color segmentation using RGB, YCrCb, and HSV, respectively.

From Fig. 4.3, it is observed that RGB yields the most false detections because of its characteristic mixture of red, green, and blue that gives a wide range of threshold values. YCrCb and HSV are both able to accurately detect the hand shape.

### 4.1.3 Background Subtraction

Background subtraction is an image processing method for classifying a moving object from the background image. This research explores both the conventional sum-of-absolute-difference (SAD) and unit-gradient-vector (UGV) algorithms for background subtraction.



1	2	3	2	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
-1	-2	-3	-2	-1

(a) horizontal mask

1	0	0	0	-1
2	0	0	0	-2
3	0	0	0	-3
2	0	0	0	-2
1	0	0	0	-1

(b) vertical mask

Figure 4.4  $5 \times 5$  custom Sobel operator mask

SAD is a simple background subtraction algorithm that uses an intensity-based image of the background as a reference and subtracts it with an intensity-based image of the current frame according to (4.5):

$$B_{SAD} = \begin{cases} 255 & ,\text{if } |I_{Bg} - I_{Curr}| > SAD_{th} \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.5)$$

where  $B_{SAD}$  is the binary image output from SAD background subtraction,  $I_{Bg}$  is the intensity-based reference background image pixel,  $I_{Curr}$  is the intensity-based current frame image pixel, and  $SAD_{Th}$  is the threshold value of SAD algorithm to classify foreground pixel from background pixel. This work uses 15 as the threshold value. If the SAD output is greater than  $SAD_{Th}$ , that pixel will become the foreground pixel, otherwise it will become the background pixel.

UGV-based background subtraction is described in [42]and [31] for the purpose of hand segmentation. This method starts by obtaining the partial derivative of  $I_{Bg}$  in both the x- and y-axis using a  $5 \times 5$  custom horizontal and vertical mask shown in Fig. 4.4. The partial derivative equation is shown in (4.6) for the reference background image.

$$\begin{aligned} I_{Bg,x} &= I_{Bg} \cdot \text{the } 5 \times 5 \text{ custom horizontal mask} \\ I_{Bg,y} &= I_{Bg} \cdot \text{the } 5 \times 5 \text{ custom vertical mask} \end{aligned} \quad (4.6)$$

where  $I_{Bg,x}$  is the partial derivative of  $I_{Bg}$  in the x-axis and  $I_{Bg,y}$  is the partial derivative of  $I_{Bg}$  in the y-axis. Note that the value of  $I_{Bg,x}$  and  $I_{Bg,y}$  should be divided by 4590 before determining the norm because the intensity-based image in OpenCV has a maximum value of 255 and minimum as 0, thus after the dot product the maximum value becomes 4590.

Next, the norms of  $I_{Bg,x}$  and  $I_{Bg,y}$  are calculated according to 4.7 and eq 4.8:

$$n_{Bg,x} = \begin{cases} \frac{(I_{Bg,x})}{\sqrt{(I_{Bg,x})^2+(I_{Bg,y})^2}} & ,\text{if } \sqrt{(I_{Bg,x})^2 + (I_{Bg,y})^2} > 0.035 \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.7)$$

$$n_{Bg,y} = \begin{cases} \frac{(I_{Bg,y})}{\sqrt{(I_{Bg,x})^2+(I_{Bg,y})^2}} & ,\text{if } \sqrt{(I_{Bg,x})^2 + (I_{Bg,y})^2} > 0.035 \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.8)$$

where  $n_{Bg,x}$  is the normalized UGV of  $I_{Bg}$  in the x-axis, and  $n_{Bg,y}$  is the normalized UGV of  $I_{Bg}$  in the y-axis. The value of 0.035 prevents the denominator from becoming zero. If the denominator value is lower than this threshold, the value of  $n_{Bg,x}$  and  $n_{Bg,y}$  will be set to zero immediately.

This process is repeated for  $I_{Curr}$  to obtain the partial derivative of  $I_{Curr}$  in the x- and y-axis, according to (4.9):

$$\begin{aligned} I_{Curr,x} &= I_{Curr} \cdot \text{the } 5 \times 5 \text{ custom horizontal mask} \\ I_{Curr,y} &= I_{Curr} \cdot \text{the } 5 \times 5 \text{ custom vertical mask} \end{aligned} \quad (4.9)$$

where  $I_{Curr,x}$  is the partial derivative of  $I_{Curr}$  in the x-axis and  $I_{Curr,y}$  is the partial derivative of  $I_{Curr}$  in the y-axis. Similar to  $I_{Bg,x}$  and  $I_{Bg,y}$ ,  $I_{Curr,x}$  and  $I_{Curr,y}$  should also be divided by 4590.

Finding the UGV by using the norm calculation of  $I_{Curr,x}$  and  $I_{Curr,y}$  that shown in eq 4.10 and 4.11.

$$n_{Curr,x} = \begin{cases} \frac{(I_{Curr,x})}{\sqrt{(I_{Curr,x})^2+(I_{Curr,y})^2}} & ,\text{if } \sqrt{(I_{Curr,x})^2 + (I_{Curr,y})^2} > 0.035 \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.10)$$

$$n_{Curr,y} = \begin{cases} \frac{(I_{Curr,y})}{\sqrt{(I_{Curr,x})^2 + (I_{Curr,y})^2}} & ,\text{if } \sqrt{(I_{Curr,x})^2 + (I_{Curr,y})^2} > 0.035 \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.11)$$

where  $n_{Curr,x}$  is the normalized UGV of  $I_{Curr}$  in the x-axis,, and  $n_{Bg,y}$  is the normalized UGV of  $I_{Curr}$  in the y-axis. Likewise, a threshold value of 0.035 is specified for the denominator to avoid the divide-by-zero condition.

The resulting images  $n_{Bg,x}$  and  $n_{Bg,y}$  are subtracted with  $n_{Curr,x}$  and  $n_{Curr,y}$  as shown in (4.12) and (4.13):

$$d_x = n_{Bg,x} - n_{Curr,x} \quad (4.12)$$

$$d_y = n_{Bg,y} - n_{Curr,y} \quad (4.13)$$

where  $d_x$  is the differential result of UGV-based background subtraction in the x-axis and  $d_y$  is the differential result of UGV-based background subtraction in the y-axis.

Furthermore,  $d_x$  and  $d_y$  are evaluated using Euclidian distance in (4.14). The result is the differential UGV background subtraction image between x- and y-axis,  $d_{xy}$ . The final output is converted into a binary image representing UGV-based background subtraction  $B_{UGV}$ , where the threshold value is specified as 0.24 to correspond to 13.48 degree. If each pixel in  $B_{UGV}$  is greater than the threshold value, then that pixel becomes a foreground pixel, else is the background pixel, as shown in (4.15).

$$d_{xy} = \sqrt{d_x^2 + d_y^2} \quad (4.14)$$

$$B_{UGV} = \begin{cases} 255 & ,\text{if } d_{xy} > 0.24 \\ 0 & ,\text{Otherwise} \end{cases} \quad (4.15)$$

The output images of SAD and UGV background subtraction are shown in Fig. 4.5(a) and 4.5(b). It is noted that SAD produces many false detections when the foreground and background images have the same intensity value. UGV consistently produces better results for hand segmentation.

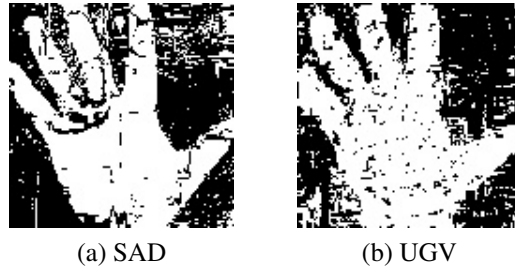


Figure 4.5 Background subtraction output image.

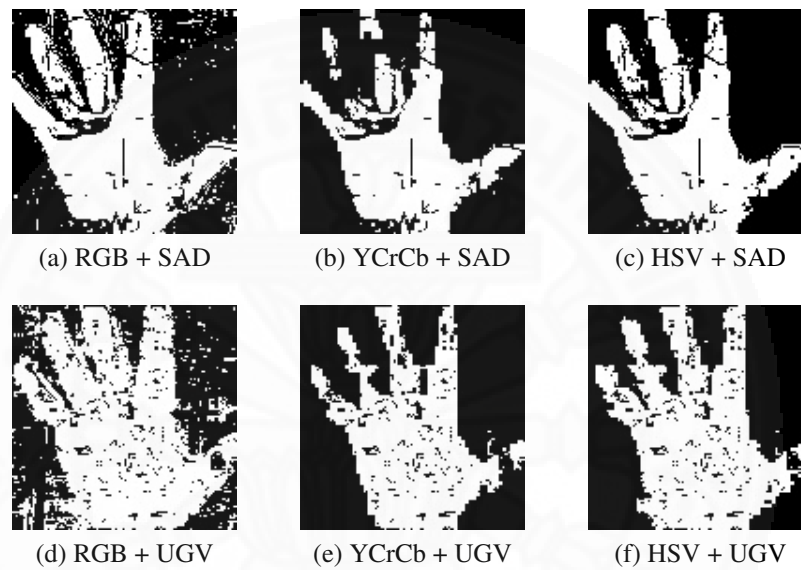


Figure 4.6 Output image from combination of color space segmentation and background subtraction methods.

#### 4.1.4 AND Operation

The AND operation is used to integrate the results from color segmentation and background subtraction by providing an intersection of pixels obtained from both methods. The advantage of doing so is the removal of false positives and negative pixels that often occur during experimentation. This operation is expressed as (4.16):

$$B_{AND} = B_{Color} \cap B_{BgSubtraction} \quad (4.16)$$

where  $B_{AND}$  is the binary image produced by the AND operation,  $B_{Color}$  is the color segmentation binary output image,  $B_{BgSubtraction}$  is the background subtraction binary output image of both SAD and UGV. Various combinations of color space and

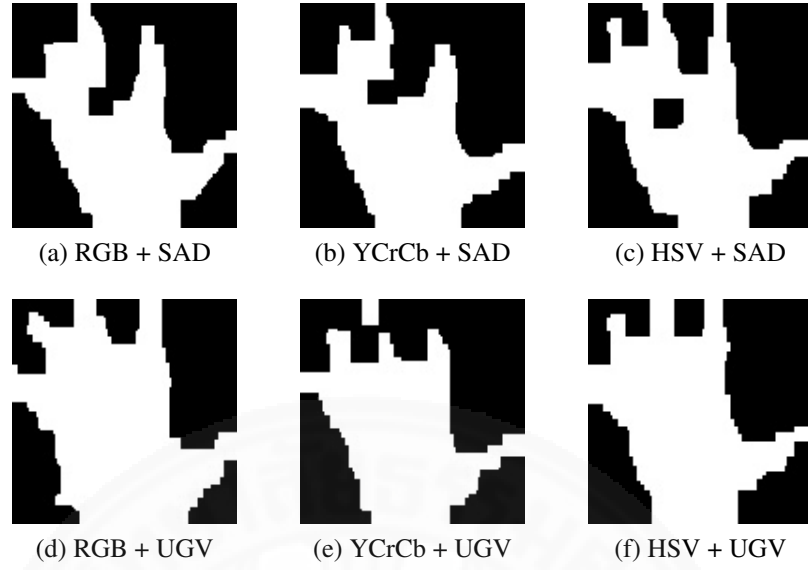


Figure 4.7 Post-processed output image from combination of color space segmentation and background subtraction methods.

background subtraction methods will be evaluated. The AND operation is shown in Fig.4.6.

#### 4.1.5 Post-Processing

As the output image from the AND operation often contain noise, post-processing is applied as a final step to smoothen the image. This involves morphological operations such as opening and closing. First, the opening operation is performed by using a square  $5 \times 5$  structure element to remove small noises from the background. The closing operation is then conducted using a square  $15 \times 15$  structure element to fill in the holes of the foreground image. This is shown in (4.17):

$$B_{Post} = (B_{AND} \circ SE_{5 \times 5}) \bullet SE_{15 \times 15} \quad (4.17)$$

where  $B_{Post}$  is the binary output image after post-processing,  $B_{AND}$  is the binary output image of the AND operation,  $SE_{5 \times 5}$  is the square  $5 \times 5$  structure element, and  $SE_{15 \times 15}$  is the square  $15 \times 15$  structure element. The final output image for each combination of color segmentation methods and background subtraction are shown in Fig.4.7.

## 4.2 Discussion of the Hand Segmentation Output

Analysis of the different combinations of color segmentations and background subtraction methods for hand segmentation indicate that SAD yields poor results, such as a hole in the foreground hand image. It can be concluded that UGV-based background subtraction is better suited for hand segmentation. Furthermore, the effects of lighting conditions and different skin colors will be explored in Chapter 6 using 7920 numerical results from experimentation.



## Chapter 5

### Finger Detection and Gesture Recognition

Chapter 5 describes the algorithm for finger detection and hand gesture recognition. Finger detection is accomplished through top-hat transform, which is a simple and effective method for determining finger positions. The hand gesture recognition is capable of identifying six hand postures, consisting of open palm (OP), forefinger (FF), forefinger and thumb (FT), fist (FS), leftward thumb (TL), and rightward thumb (TR).

#### 5.1 Finger Detection by Using Top-Hat Transform.

The flowchart for detecting the fingers and palm is shown in Figure 5.1. The proposed method applies top-hat transform to extract the finger and palm regions from the hand segmentation image. Processing for both the left and right hands can be executed simultaneously using multi-thread programming. The procedures include: 1) top-hat transform, 2) distance transformation, 3) center of gravity determination, and 4) distance measurement.

##### 5.1.1 Top-Hat Transform

Top-hat transform is a morphological image processing algorithm to detect the particle that has a smaller size than the structure element (SE) of the input image. Therefore, this method is suitable for separating the finger regions from the palm region of the hand segmentation image. The SE is defined as a circular structure with approximately the same size of the average palm region. In this work, the size of SE was determined experimentally as a circular shape with a radius value of 17 pixels. The fingers and palm regions are then classified according to (5.1) and (5.2).

$$B_{Palm} = B_{Hand} \circ SE \quad (5.1)$$

$$B_{Finger} = B_{Hand} - B_{Palm} \quad (5.2)$$

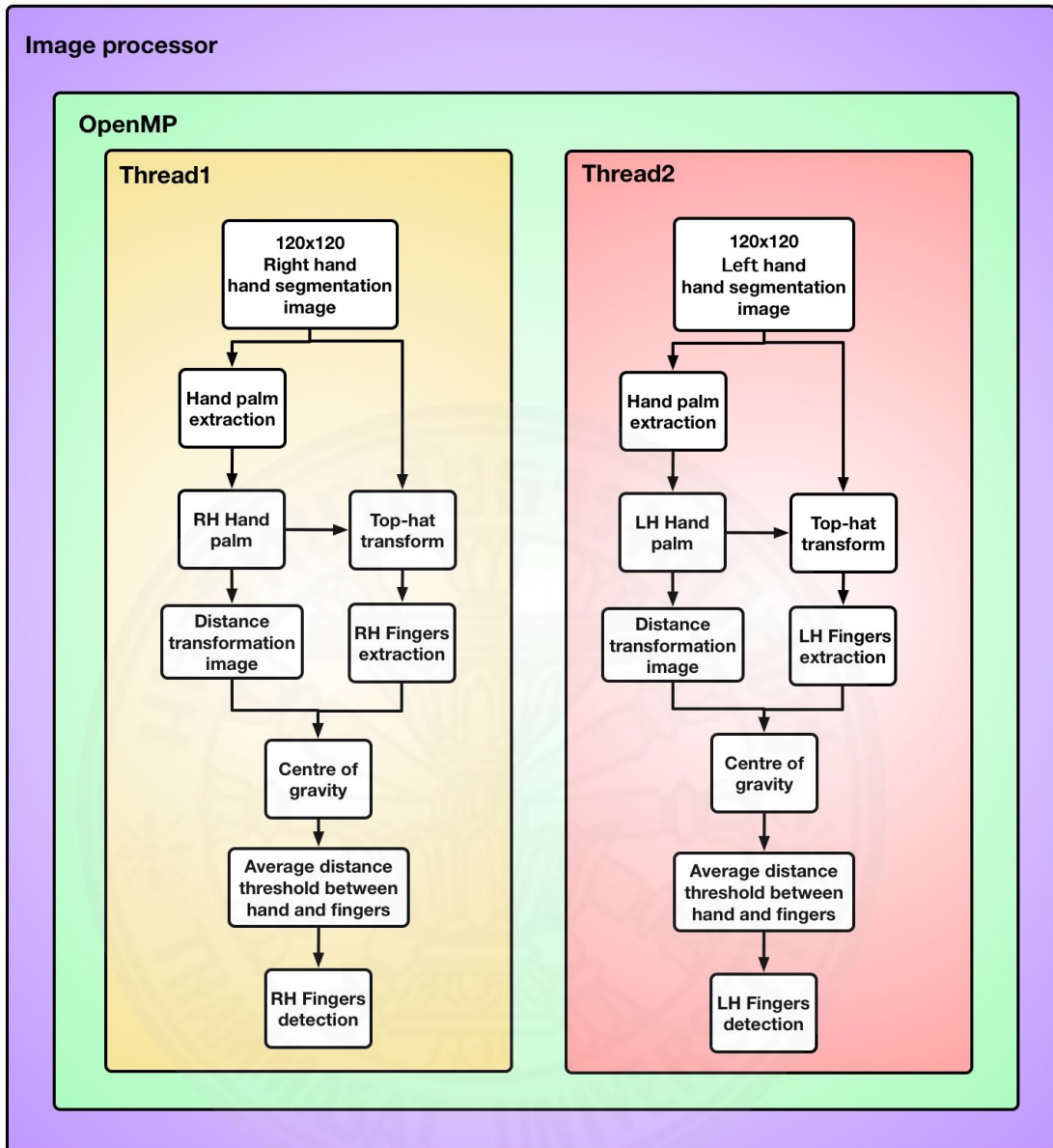


Figure 5.1 Finger detection flow chart

where  $B_{Hand}$  is a binary image from the hand segmentation input,  $SE$  is the circular structure element of size 17,  $B_{Palm}$  is a binary output image representing the palm region, and  $B_{Finger}$  is a binary output image of the finger regions.

The morphological opening operation is used to identify the palm region by locating the element with a size bigger or equal to  $SE$ . Top-hat transform then subtracts the palm region from the hand segmentation input image to obtain the finger positions. An example of input and output images using top-hat transform is shown in Figure 5.2.



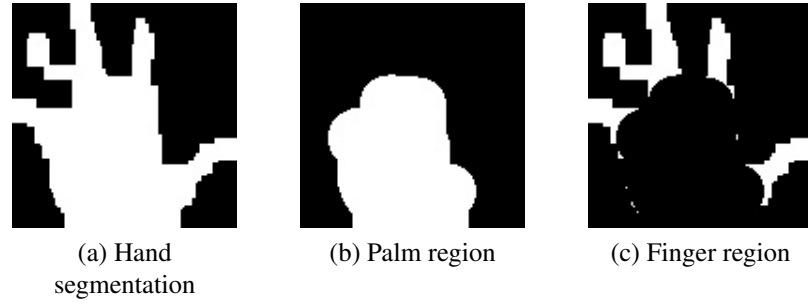


Figure 5.2 Finger detection using top-hat transform.

2.8	2.19	2	2.19	2.8
2.19	1.4	1	1.4	2.19
2	1	0	1	2
2.19	1.4	1	1.4	2.19
2.8	2.19	2	2.19	2.8

Figure 5.3 Euclidean distance transform  $5 \times 5$  mask.

### 5.1.2 Distance Transformation

Distance transformation is used to determine the distance between each pixel in the palm to the center of the region. It uses a custom  $5 \times 5$  Euclidean distance transform mask as a filter, as shown in Figure 5.3. The distance transform is expressed as (5.3).

$$B_{Palm,Dist} = B_{Palm} \cdot M_{Euc} \quad (5.3)$$

where  $B_{Palm}$  is the binary input image containing the palm region,  $M_{Euc}$  is the  $5 \times 5$  Euclidean distance transform mask, and  $B_{Palm,Dist}$  is the output image containing the distance information. An example of distance transformation is illustrated in Figure 5.4. The input image is the palm region, shown in Figure 5.4(a). The output image represents the distance between each pixel and the center of the palm, as shown in Figure 5.4(b), where the closest distance is represented as a pure white pixel. The distance transformation increases the accuracy of determining the center of the palm region in the next procedure.

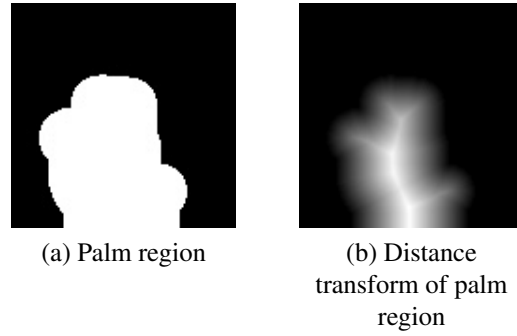


Figure 5.4 Comparison between palm region and distance transform of palm region.

### 5.1.3 Center of Gravity

To determine the positions of each finger and the palm, the center of gravity method is applied to both  $B_{Finger}$  and  $B_{Palm,Dist}$  to determine the moments of each region according to (5.4) and (5.5).

$$\begin{aligned}
 MM_{00} &= \sum_x \sum_y B_{Finger}(x,y) \\
 MM_{10} &= \sum_x \sum_y x B_{Finger}(x,y) \\
 MM_{01} &= \sum_x \sum_y y B_{Finger}(x,y) \\
 x_{c,Finger} &= \frac{MM_{10}}{MM_{00}} \\
 y_{c,Finger} &= \frac{MM_{01}}{MM_{00}}
 \end{aligned} \tag{5.4}$$

where  $B_{Finger}$  is the binary image of the finger regions,  $(x_{c,Finger}, y_{c,Finger})$  are the coordinates of the center of mass for each finger.

$$\begin{aligned}
 MM_{00} &= \sum_x \sum_y B_{Palm,Dist}(x,y) \\
 MM_{10} &= \sum_x \sum_y x B_{Palm,Dist}(x,y) \\
 MM_{01} &= \sum_x \sum_y y B_{Palm,Dist}(x,y) \\
 x_{c,Palm} &= \frac{MM_{10}}{MM_{00}} \\
 y_{c,Palm} &= \frac{MM_{01}}{MM_{00}}
 \end{aligned} \tag{5.5}$$

where  $B_{Palm,Dist}$  is the output from distance transformation of the palm,  $(x_{c,Palm}, y_{c,Palm})$  are the coordinates of the center of mass for the palm region. The centers of mass are defined as the reference coordinates of the fingers and palms in 5.6 to 5.7 as:

$$Finger_c(x,y) = (x_{c,Finger}, y_{c,Finger}) \quad (5.6)$$

$$Palm_c(x,y) = (x_{c,Palm}, y_{c,Palm}) \quad (5.7)$$

#### 5.1.4 Distance Measurement

In order to identify the hand gestures, it is necessary to measure the distances between each finger region  $Finger_c(x,y)$  and the palm  $Palm_c(x,y)$ . The Euclidean distance is defined as:

$$d_{Finger \leftrightarrow Palm} = \sqrt{(x_{c,Palm} - x_{c,Finger})^2 + (y_{c,Palm} - y_{c,Finger})^2} \quad (5.8)$$

where  $d_{Finger \leftrightarrow Palm}$  is the distance between the center of each finger and the center of the palm region.

From experimentation, the distance between the edge of the palm region and  $Palm_c(x,y)$  is approximately 30 pixels. This distance can be used as the threshold to classify the finger regions. However, many false positives are observed in the case of FS, TL, and TR. To solve this problem, an additional 7 pixels should be added to the distance threshold. Furthermore, the accuracy of the algorithm can be improved by using physical information of the human hand. When a hand is raised in the air, the finger at the lowest position is the thumb, which is also usually lower than the center of the palm region. From this reasoning, the center of mass of the palm should also be compensated by an additional 17 pixels. This means that for a region to qualify as a finger, the position must be higher than  $Palm_c(x, y + 17)$  and at a distance of at least 37 pixels or more.

Figures 5.5(a) and 5.5(b) shows the experimental results of finger detection for FS and OP. The blue circle represents the detected center of the palm, the green circle is the edge of the palm region, and the magenta circle is the distance threshold of the palm. FS shows that the palm is detected without any finger regions. OP indicates each

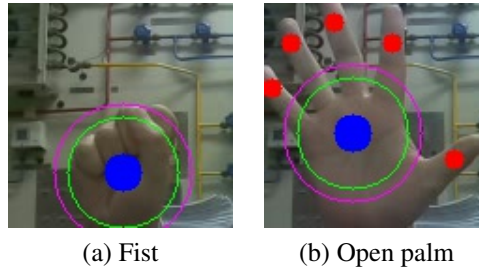


Figure 5.5 The example image of finger detection.

finger by a red circle, which must also be outside of the distance threshold from the center of the palm and situated higher than  $Palm_c(x, y + 17)$ .

## 5.2 Hand Gesture Recognition

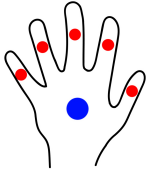
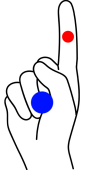
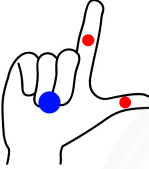

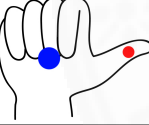
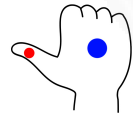
The hand gestures that will be recognized in this work includes open palm (OP), forefinger (FF), forefinger and thumb (FT), fist (FS), leftward thumb (TL), and rightward thumb (TR). This section describes the method used to classify each gesture. The algorithm is based upon the assumption of an anatomically correct human hand with five fingers. Each gesture is determined by counting the number of fingers and their relative positions from the center of the palm region, as shown in Table 5.1.

According to Table 5.1, the detection of the palm can be used to classify a hand gesture from non-hand images. The number of fingers can be used to recognize 4 groups of gestures, namely OP with more than two detected fingers, FT with exactly two detected fingers, FS with no detected fingers, and either FF, TL, or TR with one detected finger. As show in the related equation (5.9)

$$Handgesture = \begin{cases} OP & ,\text{if (number of finger} > 2) \\ FT & ,\text{if (number of finger} = 2) \\ FF \vee TL \vee TR & ,\text{if (number of finger} = 1) \\ FS & ,\text{if (number of finger} = 0) \end{cases} \quad (5.9)$$

Where *Handgesture* as a hand gesture recognition output, *OP* as the open palm, *FT* as the forefinger and thumb, *FF* as the forefinger, *TL* as the leftward thumb, *TR* as the

Table 5.1 The relationship between number of finger and hand gesture

Hand Gesture	Description
	<ul style="list-style-type: none"> <li>OP is recognized when the hand has one palm region and more than two detected fingers.</li> </ul>
	<ul style="list-style-type: none"> <li>FF is recognized when the hand has one palm region and exactly one detected finger with a position above the specific y-axis coordinate of the edge of palm.</li> </ul>
	<ul style="list-style-type: none"> <li>FT is recognized when the hand has exactly one palm and two detected fingers.</li> </ul>
	<ul style="list-style-type: none"> <li>FS is recognized when the hand has one palm and no detected fingers.</li> </ul>
	<ul style="list-style-type: none"> <li>TL is recognized when the hand has exactly one palm and one detected finger that is not in the forefinger position, and its x-axis coordinate is greater than the x-axis coordinate of the palm.</li> </ul>
	<ul style="list-style-type: none"> <li>TR is recognized when the hand has exactly one palm and one detected finger that is not in the forefinger position, and its x-axis coordinate is less than the x-axis coordinate of the palm.</li> </ul>
	<ul style="list-style-type: none"> <li>Non-hand is recognized when no palm and finger regions are detected.</li> </ul>

rightward thumb and FS as the fist.

To further differentiate between FF, TL, and TR, the position of the detected finger must be considered. Firstly, the forefinger can be separated from the thumb using the assumption that FF must have a position higher than 30 pixels above the palm along the y-axis. Otherwise, the image contains a thumb in either TL or TR. TL and TR can then be classified by examining the position along the x-axis, when TL is towards the left (greater than) and TR is towards the right (less than) of the palm center. The

classification of one finger can present as the algorithm in (5.10)

$$Handgesture = \begin{cases} FF & ,\text{if } (Finger_{c,y} < Palm_{c,y-30}) \\ TL & ,\text{if } (Finger_{c,y} > Palm_{c,y-30}) \\ & \wedge (Finger_{c,x} > Palm_{c,x}) \\ TR & ,\text{if } (Finger_{c,y} > Palm_{c,y-30}) \\ & \wedge (Finger_{c,x} < Palm_{c,x}) \end{cases} \quad (5.10)$$

Where *Handgesture* as a hand gesture recognition output, *FF* as the forefinger, *TL* as the leftward thumb, and *TR* as the rightward thumb.

### 5.3 Discussion of Finger Detection and Hand Gesture Recognition

The proposed method uses top-hat transform and center of gravity for finger detection. Furthermore, the accuracy of the algorithm is improved by using distance transformation. The center of gravity method also determines the position of each finger and palm. A region is classified as a finger if it is located at least 37 pixels or more away from the compensated center of the palm.

Hand gesture recognition is developed for 7 postures by counting the number of detected fingers and the relative position to the palm. Finger count alone allows the determination of OP, FT, FS, and non-hand. FF, TL, and TR, which all have only one detected finger, are further classified according to the position of the finger when compared to the palm center. Because of the algorithm using the position of finger and palm thus it is suitable to the upward direction of hand only as related to the natural hand gesture position.

Analysis of the hand gesture recognition will be described in Chapter 6 with numerical results from 7920 experiments under dynamic lighting condition, using different color spaces, background subtraction methods, and skin color.

## Chapter 6

### Experimental Results

This chapter presents experimental verification and analysis of the hand gesture recognition algorithm described in Chapters 4 and 5. Performance, robustness, and execution time of the proposed method are examined subjected to dynamic lighting conditions and variation in user skin color.

#### 6.1 Dynamic Lighting Condition

Dynamic lighting condition is a real-world problem for many color segmentation and background subtraction methods. Changing levels of illumination can effect perceived colors and cause false detections. Therefore, the experiments will be conducted for varying light intensity and analyzed in terms of color space and background subtraction methods.

##### 6.1.1 Light Intensity

The perceived change in color under dynamic lighting conditions depends on the intensity value. The mean image intensity value is given by (6.1):

$$I_{mean} = \frac{1}{N} \sum_x \sum_y I(x,y) \quad (6.1)$$

where  $I_{mean}$  is the mean image intensity value,  $I(x,y)$  is the intensity value of the pixel  $(x,y)$ , and  $N$  is the total number of pixels of the intensity image.

The environment setting, as described in Chapter 3, consists of a room under fluorescent lighting that can be adjusted to three levels (bright, dim, dark), as shown in Figure 6.1. The mean intensity values were calculated from 5-10 seconds of video captured under these three lighting conditions, and the maximum/minimum values are presented in Table 6.1 with corresponds to the measured illuminance SI photometry quantities value from a light meter. It is clear that the dark image shows a nearly black scene with insufficient information for color segmentation or object detection.

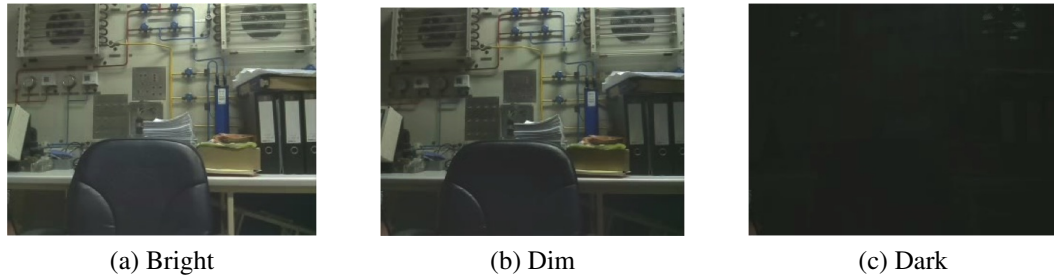


Figure 6.1 Three different light intensity levels.

Table 6.1 Light intensity

Level of light intensity	Min	Max	Illuminance (lux)
Brightness	88	92	304
Dim	65	70	30
Darkness	33	34	0

Therefore, only bright and dim images will be considered in the remaining experiments. Example images of the six hand gestures and recognition results are shown in Figure 6.2 and Figure 6.3 for bright and dim conditions, respectively.

### 6.1.2 Performance Analysis

The performance analysis is conducted for various combinations of background subtraction and color segmentation methods. Classification results are measured by a confusion matrix, which is a performance classifier that is widely used in machine learning to determine accuracy, sensitivity, specificity, and precision. The components of the confusion matrix consists of true positive (TP), true negative (TN), false positive (FP), and false negative (FN), defined in this work as:

- TP is the number of specified gesture that was correctly detected.
- TN is the number of other gestures that was correctly not detected.
- FP is the number of other gestures that was incorrectly detected as a specified gesture.
- FN is the number of specified gestures that was not detected.

A total number of 1320 images were captured from the camera for the six hand gestures under bright and dim lighting conditions (110 images per gesture per lighting



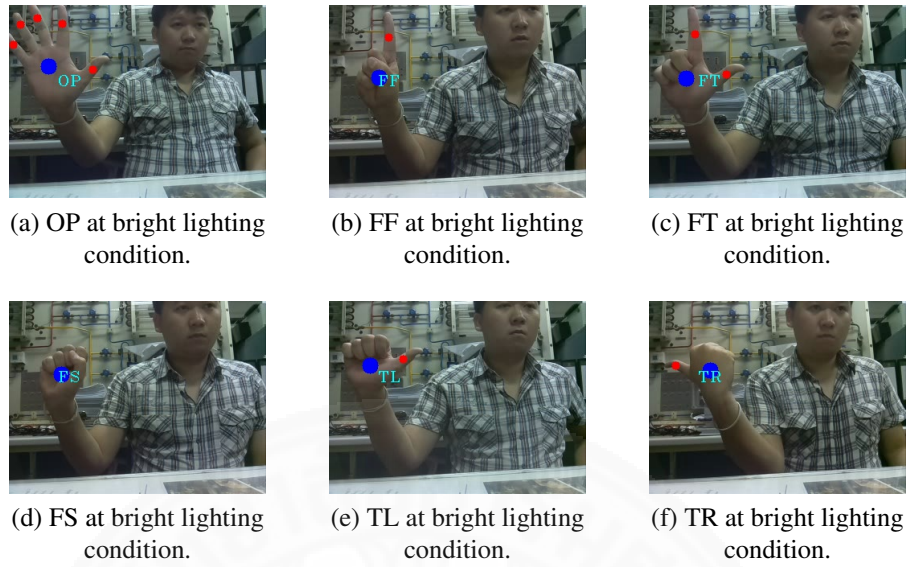


Figure 6.2 Six different hand gestures under bright lighting conditions.

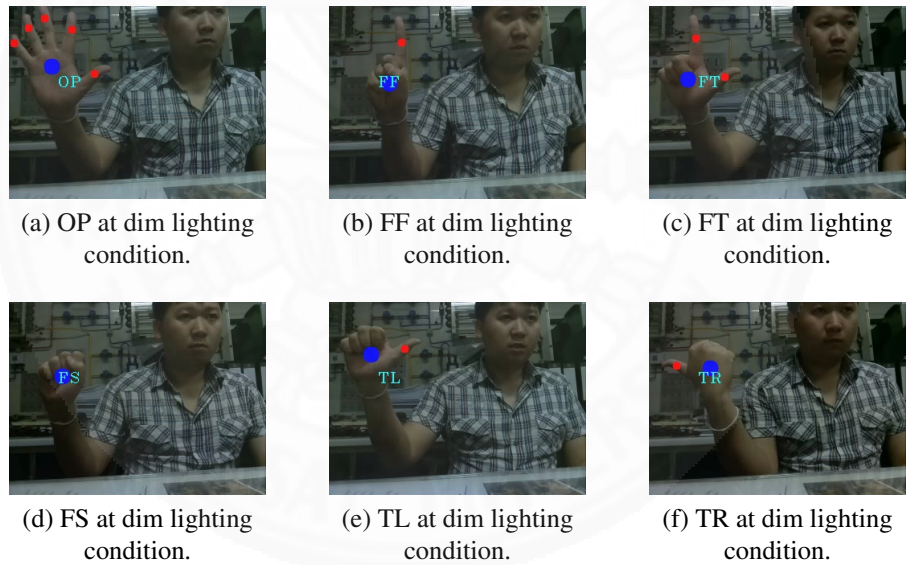


Figure 6.3 Six different hand gestures under dim lighting conditions.

condition). The robustness of each algorithm is determined by the sensitivity, which is calculated as:

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% \quad (6.2)$$

Table 6.2 Sensitivity of RGB Color Space

Level of light intensity	OP	FF	FT	FS	TL	TR
<i>RGBSAD<sub>Bright</sub></i>	82.72	19.09	21.81	30.9	30	1.81
<i>RGBSAD<sub>Dim</sub></i>	57.27	28.18	0	1.81	0	59.09
<i>RGBUGV<sub>Bright</sub></i>	86.63	29.09	0	21.81	18.18	7.27
<i>RGBUGV<sub>Dim</sub></i>	60.9	60.9	4.54	3.63	0	0

Table 6.3 Sensitivity of YCrCb Color Space

Level of light intensity	OP	FF	FT	FS	TL	TR
<i>YCrCbSAD<sub>Bright</sub></i>	55.45	54.54	100	35.45	7.27	0.9
<i>YCrCbSAD<sub>Dim</sub></i>	7.27	13.63	19.09	0	8.18	0
<i>YCrCbUGV<sub>Bright</sub></i>	99.09	100	100	100	80.9	13.63
<i>YCrCbUGV<sub>Dim</sub></i>	65.45	97.27	99.09	42.72	21.81	31.81

Table 6.4 Sensitivity of HSV Color Space

Level of light intensity	OP	FF	FT	FS	TL	TR
<i>HSVSAD<sub>Bright</sub></i>	80.9	82.72	68.18	53.63	84.54	10
<i>HSVSAD<sub>Dim</sub></i>	21.81	80	88.18	90.9	62.72	10
<i>HSVUGV<sub>Bright</sub></i>	100	100	100	97.27	87.27	51
<i>HSVUGV<sub>Dim</sub></i>	100	98.18	100	97.27	75.45	85.45

The sensitivity results are presented for the three color spaces (RGB, YCrCb, HSV) in Tables 6.2, 6.3, and 6.4, using either the sum-of-absolute-difference (SAD) or unit-gradient-vector (UGV) methods for background subtraction.

### 6.1.2.1 Background Subtraction

First, the performance of each background subtraction method is compared under bright lighting conditions. By examining the sensitivity values presented in Tables 6.2, 6.3, and 6.4, it is noted that UGV-based background subtraction performs significantly better than SAD, except in the RGB color space. This is because UGV considers the intensity vector of the image whenever a new object is introduced into the frame, whereas SAD uses the intensity values directly to determine foreground objects. The disadvantage of SAD is the false negatives that occur when the foreground object has the same intensity value as the background. The strength of UGV is clearly seen when the lighting intensity is reduced from bright to dim. The sensitivity of SAD drops drastically when compared to UGV, because the lighting condition changes greatly

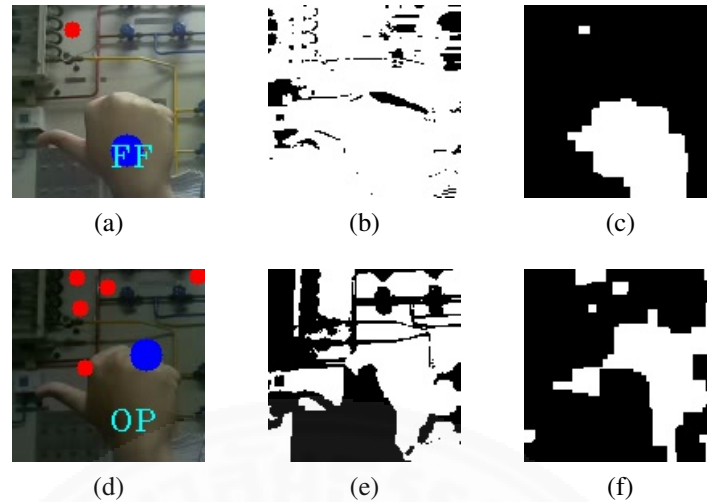


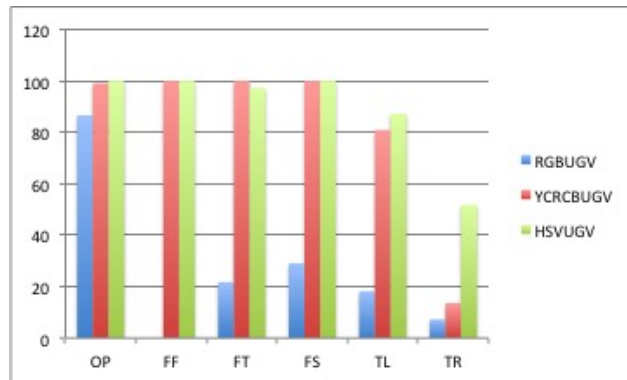
Figure 6.4 RGB color segmentation under bright (a)-(c) and dim (d)-(f) conditions.

effect the intensity values of all pixels. For these reasons, UGV-based background subtraction is utilized in the proposed method.

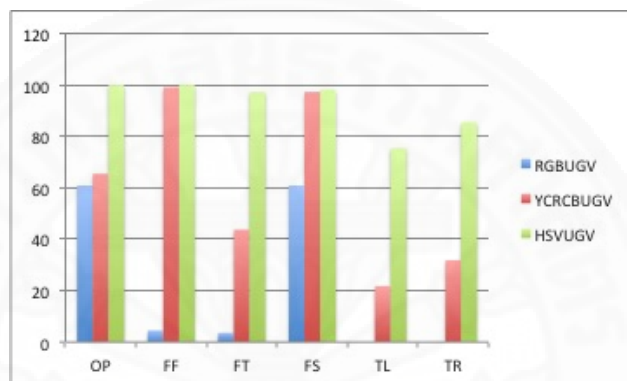
### 6.1.2.2 Color Segmentation

The next section analyzes the robustness of each color space against dynamic lighting conditions. The sensitivity result for the three color spaces (RGB, YCrCb, HSV) are shown in Tables 6.2, 6.3 and 6.4. Considering only UGV background subtraction, a comparison between RGB, YCrCb, and HSV is shown by the column graph in Figure 6.5 for both bright and dim conditions.

Under bright conditions, RGB demonstrate poor sensitivity for all hand gestures, except for OP. This is because the RGB color space does not directly support color illumination, leading to a wider color threshold range and many false detections. An example of incorrect detection is shown in Figure 6.4, where the TL gesture was wrongly recognized as FT. The result of RGB color segmentation shows that all hand pixels were correctly identify. However, the wide color threshold of RGB meant that a significant portion of the background was falsely detected as well. Post-processing and UGV background subtraction were not able to completely eliminate some background regions, leading to incorrect classification. YCrCb achieved higher than 80% sensitivity for all hand gestures, except for TR (13%). HSV also produced good results



(a) Bright



(b) Dim

Figure 6.5 Comparison of sensitivity for each color space using UGV-based background subtraction method.

that were higher than 87% for all hand gestures, again except for TR (51%). The reason for the poor performance of TR is because it mainly shows the back of the hand, rather than the front, which has the different twist angle of hand posture image to give the error in the distance measurement. Nevertheless, it can be concluded that both YCrCb and HSV color segmentation yield higher sensitive than RGB under bright conditions.

The performance of YCrCb color segmentation decreases drastically when the lighting intensity changes. This is demonstrated using the OP hand gesture for both bright and dim conditions in Figure 6.6. When the image is sufficiently bright, YCrCb correctly determines OP with a sensitivity of 99.09%, as shown in Figure 6.6(a)-(c). By examining the segmented images, it is clear that the outline information of the palm and all fingers was preserved. However, Figure 6.6(d)-(f) shows that with dim lighting, some of the fingers were not correctly detected. Following post-processing,

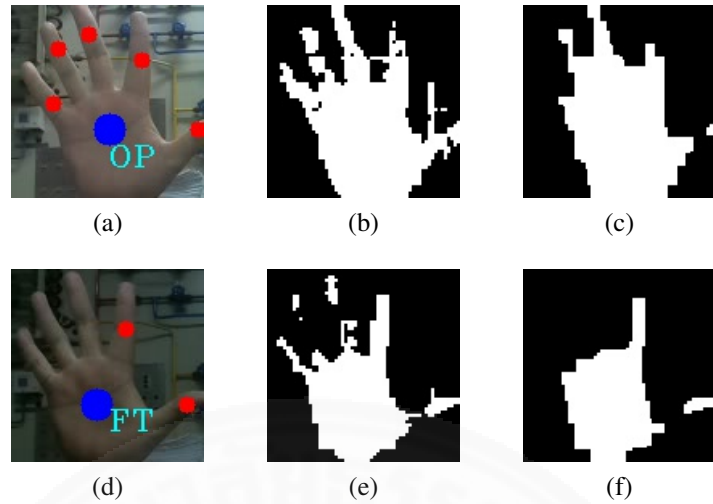


Figure 6.6 YCrCb color segmentation under bright (a)-(c) and dim (d)-(f) conditions.

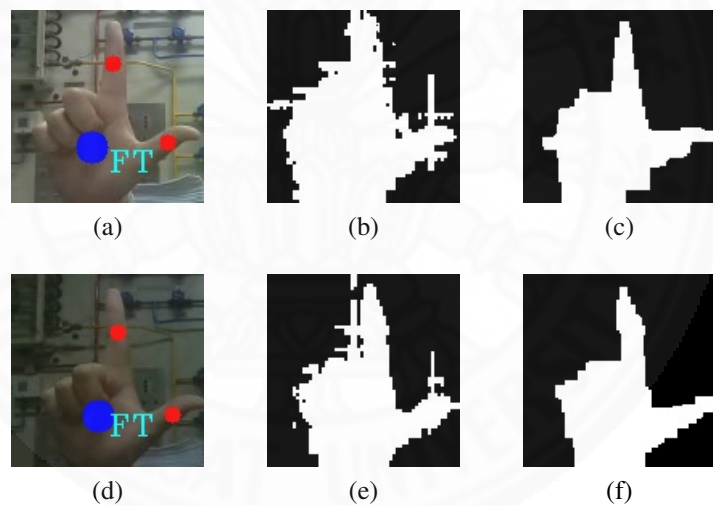


Figure 6.7 HSV color segmentation under bright (a)-(c) and dim (d)-(f) conditions.

the middle, ring, and little fingers disappeared, leading to a false classification as FT.

In comparison, HSV color segmentation maintains good detection performance under dim conditions, with sensitivity values of greater than 75.45%. This is because of the inherent robustness of HSV to changes in illumination, as the saturation and value is capable of describing the brightness/darkness of a color without effecting the color range or hue. This can be seen in Figure 6.7(a)-(c) and (d)-(f). HSV correctly detects the forefinger and thumb for both situations.



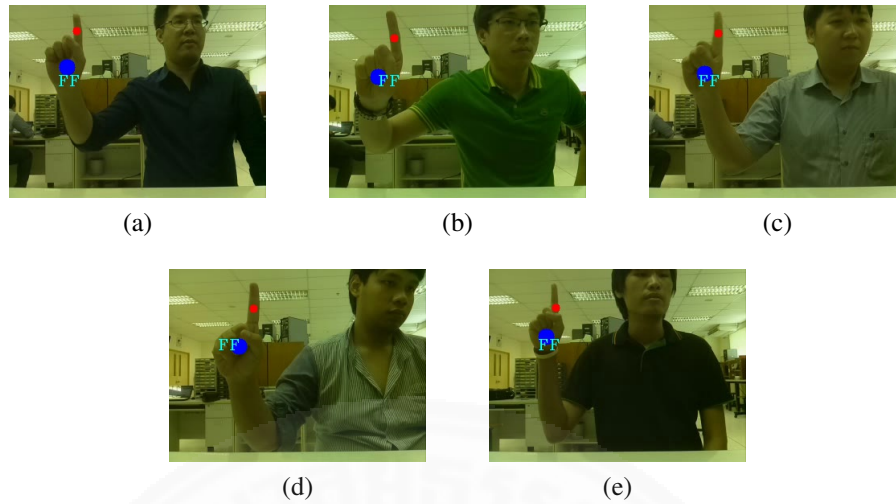


Figure 6.8 FF hand-gesture recognition of 5 users with different skin color.

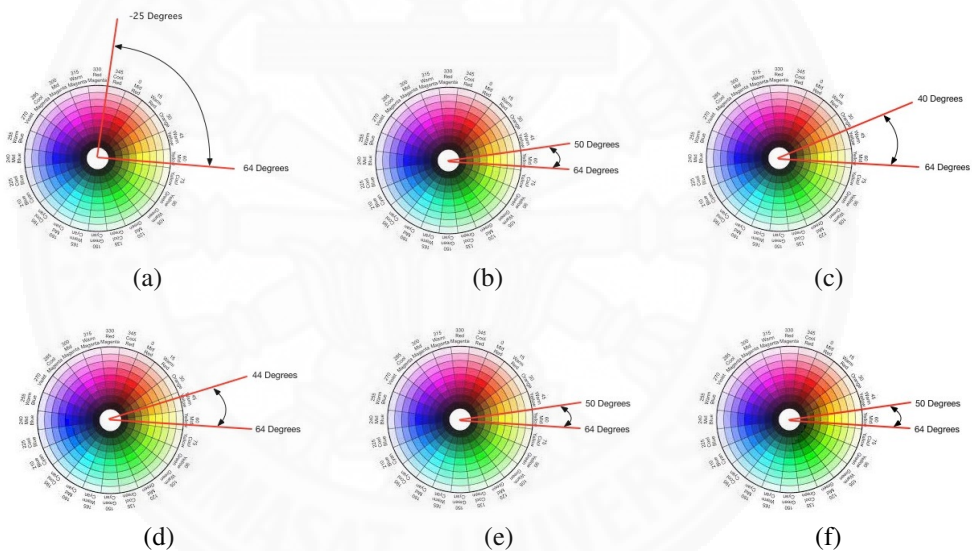


Figure 6.9 Hue circle and threshold values for 5 users with different skin color.

## 6.2 Effect of Skin Color

This section discusses the variation of user skin color on the proposed method. From the previous sections, the experimental results will focus only on HSV color segmentation with UGV background subtraction. Sample images were captured from five users of Asian ethnicity but with varying skin color, as shown in Figure 6.8.

The hue threshold values were determined experimentally for each user and compared to range used by the proposed method in Figure 6.9. The threshold minimum value was 40 for User B with the palest skin color, 44 for user C, and 50 for

Table 6.5 Execution time of each process.

Execution Time	Min ( $\mu$ s)	Max ( $\mu$ s)
Color Segmenatation	1,469	3,173
UGV-based Background Subtraction	31,481	33,232
AND Operation	100	124
Post-Processing	9,724	9,952
Palm Detection with Opening	242,753	243,135
Finger Detection with Top-Hat Transform	241,275	248,502
Hand-Gesture Recognition	6,338	7,080
Total	533,278	545,198

the others, corresponding to a reddish hue. For all users, the maximum threshold value was 64 that indicates yellow hue. From this analysis, the proposed method used the hue range shown in Figure 6.9(a), which was shown to perform well for users of Asian ethnicity. So this Hue circle has related to the H threshold range between 0 to 50 is support to all skin detection but the more yellow skin, H should be 0 to 60 in [11].

### 6.3 Real-Time Implementation

To implement the proposed hand-gesture recognition algorithm on an embedded system for human-machine interfacing, it is important to consider real-time responsiveness. According to the software design presented in Chapter 3, there are three main components: image processor, control interface, and mp3 player. Experimental results indicate that the image processor requires the most computation time due to the complexity of image processing algorithms. Therefore, this section will analyze the computation cost and verify real-time implementation.

#### 6.3.1 Execution Time

Execution time is the amount of time that the program requires per one processor cycle. Thus, a low execution time implies that the processor is faster. The proposed hand-gesture recognition algorithm will be implemented on Raspberry Pi2 Module B using C/C++, which contains a library to measure execution time in milli- or microseconds (time.h). Alternatively, processing time can be evaluated by using an oscilloscope to monitor high and low voltage signals from the general port

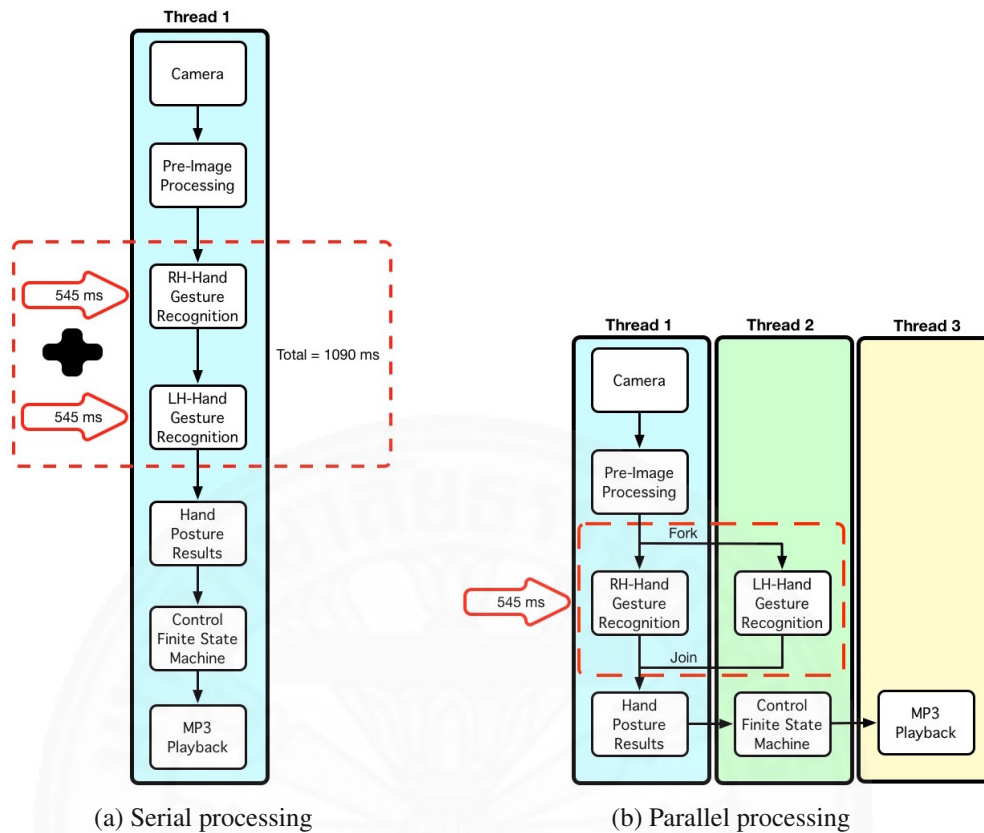


Figure 6.10 The proposed method implemented using serial and parallel processing.

input-output (GPIO). The GPIO issues an high signal at the start of the algorithm and a low signal upon completion. The execution can be obtained by measuring the duration of the high signal. This research employs both methods to determine the execution time of the proposed algorithm.

The execution time results are presented in Table 6.5. It is noted that finger detection using top-hat transform requires the most execution time, followed by palm detection. Both processes present large computational load due to the large filter sizes. The total processing time of the proposed method is between 533.278 and 545.198 ms per frame.

### 6.3.2 Serial and Parallel Processing

Simple applications are usually implemented using serial processing, where each component of the software is executed in sequence. Figure 6.10(a) shows the proposed method if run using a single thread with a serial structure. Since both left and right



hand-gesture recognitions require approximately 545 ms each, there is a bottleneck of 1090 ms during the processor cycle.

Alternatively, multi-thread programming can be used to execute parallel processing of the algorithm, as presented in Figure 6.10(b). This allows both left and right hand-gesture recognitions to be performed simultaneously, reducing the execution time back to 545 ms. The resulting fps of 1.83 fps satisfies the soft real-time condition that was specified as 1 command per second.



## Chapter 7

### Conclusion

The purpose of this thesis was to develop a real-time, vision-based hand gesture recognition system on an embedded platform that can be used for accurate and efficient human-machine interaction under dynamic lighting conditions. A monocular camera was employed to capture a human user situated 50-80 cm from the system to control media playback of an MP3 player using six different hand gestures. The application design was divided into three main parts, including image processing, control interface, and MP3 playback. The image processing part includes camera interfacing, hand segmentation, and finger detection. The control interface consists of a finite state machine to connect the output of hand gesture recognition with MP3 playback commands.

The real-world problem of changes in light intensity was directly considered in the design of the algorithm. This work analyzed the use of different color segmentation and background subtraction methods to overcome this problem. It was determined that the use of HSV color segmentation and UGV background subtraction provided the best performance, sensitivity, and robustness in both dim and bright conditions. This was then combined with top-hat transform to produce a simple and precise algorithm for recognizing hand gestures within the limit scope distance for the scaling metering effect, which included open palm (OP), forefinger (FF), forefinger and thumb (FT), fist (FS), leftward thumb (TL), and rightward thumb (TR). Experimental results and comparison with conventional methods were shown, along with an examination of the effects of user skin color.

This thesis considered the execution time of the algorithm when implemented on an embedded system, chosen as the Raspberry Pi2 Module B. By using parallel processing and multi-thread programming to handle the recognition of both the left and right hand regions simultaneously, the average processing time was 545 ms or approximately 1.83 fps. This satisfied the soft real-time deadline of 1 command per second as specified by the scope of this work to ensure smooth operations.

## References

- [1] A. Choudhury, A. K. Talukdar and K. K. Sarma (2014). A novel hand segmentation method for multiple-hand gesture recognition system under complex background. *Proc. of International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 136–140.
- [2] A. Dix, J. Finlay, G. D. Abowd, R. Beale (2004). *Human Computer Interaction*. Pearson Education Limited, Edinburg Gate, Harlow, Essex CM20 2JE, England.
- [3] A. Li. A method of detecting and recognizing hand gestures using opencv. <https://www.andol.me/1661/a-method-of-detecting-and-recognising-hand-gestures-using-opencv/>. Accessed on 6-March-2017.
- [4] A. Shahbaz, J. Hariyono, K. Jo (2015). Evaluation of background subtraction algorithms for video surveillance. *Proc. of Frontiers of Computer Vision (FCV)*, pages 1–4.
- [5] American Society for Surgery of the Hand (1990). *The Hand Examination and Diagnosis 3rd edition*. Churchill Livingstone, New York.
- [6] B. Hong, Z. Xingui (2010). Study on hand gesture segmentation. *Proc. of International Conference on Multimedia Technology (ICMT)*, pages 1–4.
- [7] C. Aiping, P. Lian, T. Yaobin, N. Ning (2010). Face detection technology based on skin color segmentation and template matching. *Proc. of second International Workshop on Education Technology and Computer Science*, pages 708–711.
- [8] CMake.org. About cmake and origin. <https://cmake.org/overview/>. Accessed on 6-March-2017.
- [9] D G. Bailey (2011). *Design for Embedded Image Processing on FPGAs*. John Wiley & Sons (Asia) Pte Ltd., 1 Fusionopolis Walk, # 07-01 Solaris South Tower, Singapore 138628.

- [10] Dr Andrew Greensted. Switch debouncing. <http://www.labbookpages.co.uk/electronics/debounce.html>. Accessed on 17-July-2017.
- [11] E. Kondela, H. D. Jun (2011). A power law transformation predicting lightness conditions based on skin color space detection. *Proc. of International Conference on Trust, Security and Privacy in Computing and Communications(TrustCom)*, pages 1472–1476.
- [12] E. Upton. New 8-megapixel camera board on sale now at \$25. <https://www.raspberrypi.org/blog/new-8-megapixel-camera-board-sale-25/>. Accessed on 6-March-2017.
- [13] E. Upton. Raspberry pi 2 on sale now at \$35. <https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/>. Accessed on 6-March-2017.
- [14] elinux.org. Rpi sd cards. [http://elinux.org/RPi\\_SD\\_cards](http://elinux.org/RPi_SD_cards). Accessed on 16-July-2017.
- [15] freegreatpicture. Basic hand gesture. <http://www.freegreatpicture.com/gestures-album/hand-gesture-30690>. Accessed on 6-March-2017.
- [16] G. B. Narejo, S. P. Bharucha (2013). Real time finger counting and virtual drawing using color detection and shape recognition. *Proc. of 5th International Conference on Information & Communication Technologies (ICICT)*, pages 1–6.
- [17] G. Liu and Z. Shi (2011). Embedded implementation of real-time skin detection system. *Proc. of International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*, pages 2463–2466.
- [18] G. Simion, V. Gui, M. Otesteanu (2011). Finger detection based on hand contour and colour information. *Proc. International Symposium on Applied Computational intelligence and Informatics (SACI)*, pages 97–100.

- [19] Guest Post. The desktop pc might finally be making a comeback.  
<http://www.valuewalk.com/2017/03/desktop-pc-might-finally-making-comback/>. Accessed on 23-July-2017.
- [20] J. Farooq, M. B. Ali (2014). Real time hand gesture recognition for computer interaction. *International Conference on Robotics and Emerging Allied Technologies in Engineering(iCREATE)*, pages 73–77.
- [21] K. Grifantini. Intelligent machines open-source data glove: Aceleglove can be programmed for many application. <https://www.technologyreview.com/s/414021/open-source-data-glove/>. Accessed on 23-July-2017.
- [22] K. Jack (2005). *Video Demystified*. Elsevier, 200 Wheeler Road, Burlington, MA 01803, USA.
- [23] K. N. Plataniotis, A. N. Venetsanopoulos (2000). *Color Image Processing and Applications*. Springer-Verlag Berlin Heidelberg.
- [24] Kook-Yeol Yoo (2014). Robust hand segmentation and tracking to illumination variation. *Proc. of International Conference on Consumer Electronics (ICCE)*, pages 286–287.
- [25] L. Gu, X. Yuan, T. Ikenaga (2012). Hand gesture interface based on improved adaptive hand area detection. *Proc. of International Symposium on Intelligent Signal Processing and Communication Systems(ISPACS)*, pages 463–468.
- [26] M. Pawlak (2006). *Image Analysis by Moments: Reconstruction and Computation Aspects*. Oficyna Wydawnicza Politechniki Wroclawskiej, 50-370 Wroclaw, Wybrzeze Wyspainskiego 27.
- [27] M. Piccardi (2004). Background subtraction techniques: a review. *Proc. of international conference on Systems, Man and Cybernetics*, pages 3099–3104.
- [28] mingw.org. Mingw. <http://www.mingw.org>. Accessed on 6-March-2017.

- [29] OpenCV.org. Miscellaneous image transformations. [http://docs.opencv.org/3.0-beta/modules/imgproc/doc/miscellaneous\\_transformations.html](http://docs.opencv.org/3.0-beta/modules/imgproc/doc/miscellaneous_transformations.html). Accessed on 6-March-2017.
- [30] P. A. Laplante, S. J. Ovaska (2012). *Real-Time System Design and Analysis: Tools for the practitioner*. John Wiley & Sons, Inc, Hoboken, New Jersey.
- [31] P. Prasertsakul and T. Kondo (2014). A robust hand segmentation method base on color and background subtraction. *Proc. of International Conf. ICICTES*.
- [32] P. Prasertsakul, T. Kondo (2014). A fingertip detection method based on the top-hat transform. *Proc. of International Conference ECTI-CON*, pages 1–5.
- [33] P. Premaratne (2014). *Human Computer Interaction Using Hand Gestures*. Springer.
- [34] P. R. V. Chowdary, M. N. Babu, T. V. Subbareddy, B. M. Reddy (2014). Image processing algorithms for gesture recognition using matlab. *Proc. of International Conference on Advanced Communication Control and Computing Technology (ICACCCT)*, pages 1511–1514.
- [35] P. Shah. Introduction to human machine interface (hmi). <http://www.doyouknow.in/Articles/Technology/Introduction-To-Human-Machine-Interface-HMI-Human-Machine-Interface-Design-Human-Machine-Interface-Software.aspx>. Accessed on 6-March-2017.
- [36] Pierre. Opencv&pi cam. <https://thinkrpi.wordpress.com/2013/05/22/opencvpi-cam-step-2-compilation/1>. Accessed on 17-july-2017.
- [37] R. M. Jusoh, N. Hamzah, M. H. Marhaban, N. M. A. Alias (2010). Skin detection based on thresholding in rgb and hue component. *Proc. of Symposium on Industrial Electronics and Applications (ISIEA 2010)*, pages 515–517.
- [38] R. Wang, Z. Yu, M. Liu, Y. Wang, and Y. Change (2014). Real-time visual static hand gesture recognition system and its fpga-based hardware implementation.

*Proc. of 12th International Conference on Signal Processing (ICSP)*, pages 434–439.

- [39] Raspberry Pi Foundation. Raspberry pi 2 model b. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Accessed on 6-March-2017.
- [40] S. Furber (2000). *ARM System-On-Chip Architecture second edition*. Pearson Education Limited 2000.
- [41] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein (2009). *Introduction to Algorithms*. Massachusetts Institute of Technology, Cambridge, Massachusetts London, England.
- [42] T.Kondo (2011). An image sequence segmentation method using gradient orientation information. *SICE Annual Conference*, pages 34–36.
- [43] W. Wang J. Pan (2012). Hand segmentation using skin color and background information. *Proc. of the International Conference on Machine Learning and Cybernetics*, pages 1487–1492.
- [44] Wikipedia. Embedded system. [https://en.wikipedia.org/wiki/Embedded\\_system](https://en.wikipedia.org/wiki/Embedded_system). Accessed on 6-March-2017.
- [45] Wikipedia. Yuv. <https://en.wikipedia.org/wiki/YUV>. Accessed on 6-March-2017.
- [46] Y. Lei, W. Hongpeng, T. Dianxiong, and Wangjue (2014). A real-time hand gesture recognition algorithm for an embedded system. *Proc. of International Conference on Mechatronics and Automation.(ICMA)*, pages 901–905.
- [47] Y. Shi, R. Taib, S. Lichman (2006). Gesturecam: A smart camera for gesture recognition and gesture-controlled web navigation. *Proc. of International Conference on Control, Automation, Robotics and Vision (ICARCV 06)*, pages 1–6.

- [48] Y. V. Parkale (2012). Gesture based operating system control. *Proc. of International Conference on Advanced Computing & Communication Technologies*, pages 318–323.

