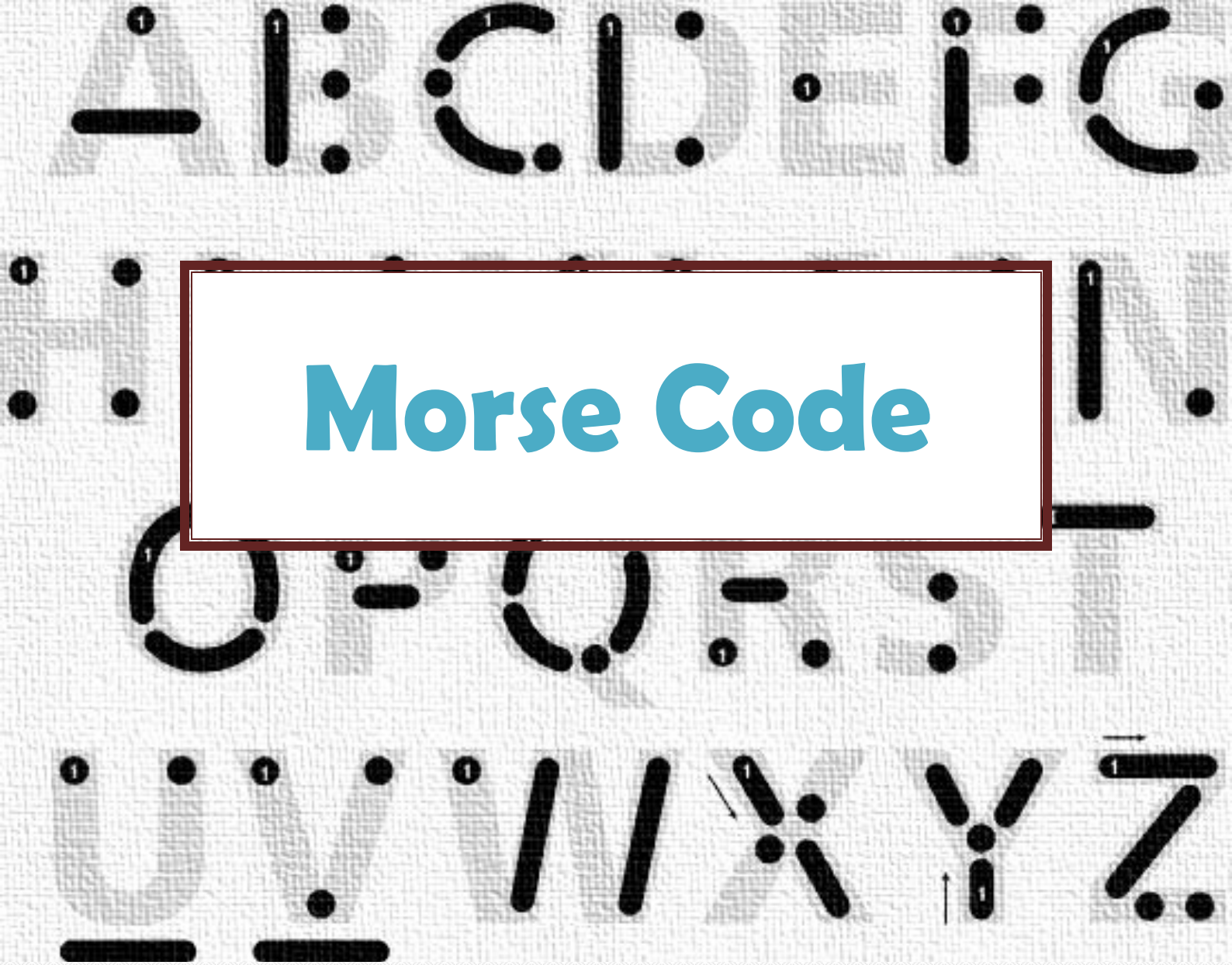


Morse Code

Final Year Project 2013 (1st Term)

LYU 1305

Real-Time Morse Code Communication App

Supervisor: Prof. LYU Rung Tsong Michael

Students: LUO Xin (1155026046)

ZOU Lei (1155026057)

Department of Computer Science and Engineering

The Chinese University of Hong Kong

<This is a blank page>

Abstract

Real time Morse Code Communication App is an application focused on Morse Code emission and reception in light. Our goal of the project is to implement this application in Android devices.

In this report, we will go through the whole process related to our interesting project. A table of content will guide us to each individual part of this report. At the beginning, Introduction part gives a rough idea about why we do this project and what our objectives are. We did some research on the Morse code and the main technology OpenCV used in this project. Some key concepts are explained so that we can understand all the things in a reasonable way. Then we present our design and implementation of the application. Some special functions usage and realization process accompanied with some pictures helping explain in a detail way. Testing is always necessary in any project. In the following Experiment and Testing part, we present how we find problems, solve problems and improve our application's performance. Finally, in the conclusion part, we show how we implement our application step by step. In the implementation process, we found that there still exist some limitations in our application, so we give an overview to these limitations and express our expectations in future development in the end.

Table of Contents

Chapter 1: Introduction

1.1 Background

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. It is initially invented by Samuel Finley Breese Morse in 1937. And it has been in use for more than 160 years—longer than any other electrical coding system. What is called Morse code today is actually somewhat different from what was originally developed by Vail and Morse. After some changes, International Morse Code was standardized at the International Telegraphy Congress in 1865 in Paris, and was later made the standard by the International Telecommunication Union (ITU). International Morse code today is most popular among amateur radio operators [1].

Morse code is useful in many fields, such as radio navigation, amateur radio, warship, the signal lamp included in a submarine periscope and so on. Besides, Morse code has been employed as an assistive technology, helping people with different native languages or people with a variety of disabilities to communicate. For the general public, an important application is signaling for help through SOS, "···— — — ···". This can be sent by many ways: keying a radio on and off, flashing a mirror, toggling a flashlight and similar methods.

This will be very useful especially when you are in wild and your phone is out of power or no signal.

1.2 Motivation

Nowadays, there are many kinds of Android Apps of Morse Code in the android market. Here are some examples:

Morse Code Trainer [2]**:**

This is an app helping users to learn Morse Code. One can choose either transmitting or receiving mode to practice corresponding skill and receive the performing feedback immediately. It includes the functions of letter training (both transmitting and receiving), word training (only transmitting), free mode, speed adjusting (WPM), sound effects adjusting and electronic handbook.

Morse Code Translator [3]:

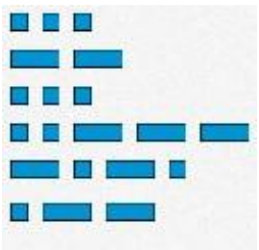
This is an app allowing users to send short flashlight text messages using the International Morse Code.

The apps includes the features that the flashlight can transmits short messages of lights (Morse Code); there are some template stored in the database for emergencies. For example, SOS; allowing users to save new messages; changing frequency of the transmitted signal.

Simple Morse Code Translator [4]:

This app allows users to input any text by keyboard or voice or select a commonly transmitted word or phrase. The app translates the received message and broadcast the translated text via camera flash.

SMS2CW - Convert to Morse Code [5]:



This is an app to convert incoming SMS or TXT messages into audible Morse Code. Once the user enable it and set the options, it will intercept incoming text messages and beep them out in Morse Code.

In summary, present Morse code apps in Android market mainly includes the following features:

- Helping users to learn Morse code;
- Allowing users to type Morse code;
- Encoding and Decoding between text message and Morse code (dot and dash);
- Decoding audible Morse code (beeps) to text message;
- Play Morse code with audible messages (beeps);
- Play Morse code with flashlight.

However, we didn't find an app that can receive a light message and decode it to a text message. Therefore, in addition to encoding Morse code by light sequences, we consider developing an android app to decode the Morse code generated by light. This will be very useful when you are in a disaster, in wild with a power-off cellphone or in other similar situation and you want to ask for help (sending "SOS" signal). Furthermore, in some movies, there will sometimes appear some Morse code message generated by light. For entertainment, if we can decode those Morse code ourselves, that would be very interesting.

1.3 Objectives

In our final year project, we are going to study the Morse Code, android programming, image processing, computer vision, real-time processing and develop a Real-time Morse Code Communication App for Android mobile device. The video camera of the Android mobile device is used to capture the flash light of another Android mobile device and decode the Morse code light pattern through the recognition of the image sequences.

Therefore, in our final year project, we want to achieve following objectives:

- Encoding Morse code and displaying it by flashlight;
- Decoding Morse code of light pattern;
- Allowing users to change transmission rates;
- The decoding part can decode messages with any transmission rate in some range.
- Storing some template in a database for convenience and in case of emergency, for example, SOS.
- Allowing users to save their words or sentences frequently used to the template database.
- Implement the bi-directional communication in the standard way.

1.4 Development Environment

Development platform: Windows 7

Application platform: Android OS

Development tool: Eclipse

Programming language: Java

Open source library: OpenCV

Encoding testing device: SAMSUNG GALAXY S4

Decoding testing device: SAMSUNG GALAXY S3

Chapter 2: Morse Code [6]

2.1 Overview

At the beginning of this report, we've already had a rough idea about what Morse code is and what Morse code can do. Here we introduce the mechanism of Morse code in detail, like what Morse code is composed of and how it works. International Morse code will be introduced and used here.

2.2 Coding Rule

International Morse code is composed of five elements:

1) Short mark, dot or "dit" (●) which is one time unit long.
2) Longer mark, dash or "dah" (▬) which is three times units long.
3) Inter-element gap between the dots and dashes within a character which is one dot's duration (one unit long).
4) Short gap between letters which is three times units long.
5) Medium gap between words which is seven times units long.

2.3 Symbol Representation

2.3.1 Letters

Character	Code	Character	Code	Character	Code
A	● ▬	J	● ▬ ▬ ▬	S	●●●
B	▬ ●●●	K	▬ ● ▬	T	▬
C	▬ ● ▬ ●	L	● ▬ ●●	U	●● ▬
D	▬ ●●	M	▬ ▬	V	●●● ▬
E	●	N	▬ ●	W	● ▬ ▬
F	●● ▬ ●	O	▬ ▬ ▬	X	▬ ●● ▬
G	▬ ▬ ●	P	● ▬ ▬ ●	Y	▬ ● ▬ ▬
H	●●●●	Q	▬ ▬ ● ▬	Z	▬ ▬ ●●
I	●●	R	● ▬ ●		

2.3.2 Numbers

Character	Code	Character	Code
0	▬ ▬ ▬ ▬ ▬	5	●●●●●
1	● ▬ ▬ ▬ ▬	6	▬ ●●●●
2	●● ▬ ▬ ▬	7	▬ ▬ ●●●
3	●●● ▬ ▬	8	▬ ▬ ▬ ●●
4	●●●● ▬	9	▬ ▬ ▬ ▬ ●

2.3.3 Punctuation

Character	Code	Character	Code
Period [.]	● ▬ ● ▬ ● ▬	Colon [:]	▬ ▬ ▬ ●●●
Comma [,]	▬ ▬ ●● ▬ ▬	Semicolon [;]	▬ ● ▬ ● ▬ ●
Question mark [?]	●● ▬ ▬ ●●	Double dash [=]	▬ ●●● ▬
Apostrophe [‘]	● ▬ ▬ ▬ ▬ ●	Plus [+]	● ▬ ● ▬ ●
Exclamation mark [!]	▬ ● ▬ ● ▬ ▬	Hyphen, Minus [-]	▬ ●●●● ▬
Slash [/]	▬ ●● ▬ ●	Underscore [_]	●● ▬ ▬ ● ▬
Parenthesis open [(]	▬ ● ▬ ▬ ●	Quotation mark [“]	● ▬ ●● ▬ ●
Parenthesis close [)]	▬ ● ▬ ▬ ● ▬	Dollar sign [$]	●●● ▬ ●● ▬
Ampersand [&]	● ▬ ●●●	At sign [@]	● ▬ ▬ ● ▬ ●

2.4 Speeds

An operator must choose two speeds when sending a message in Morse code. One is the character speed, or how fast each individual letter is sent. The other is text speed, or how fast the entire message is sent. An operator could generate the characters at a high rate, but by increasing the space between the letters, send the message more slowly.

Therefore, duration of a dot plays an important role in speed deciding.

For example, if dot = 0.5 seconds (one time unit), we will have:

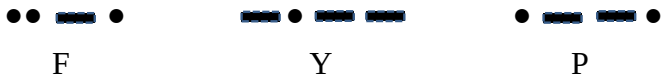
dash = 3 * dot = 1.5 seconds

space = 7 * dot = 3.5 seconds.

The lower the dot duration, the higher the speed of the message is sending. Generally more experienced operators can send and receive at faster speeds.

2.5 Instance

Here is an example of phrase “F Y P” in Morse Code format, each letter is separated by a space:



F Y P

Morse Code is often spoken or written with “dah” for dashes, “dit” for dots located at the end of a character, and “di” for dots located at the beginning or internally within the character. Thus, “F Y P” is orally:

Di-di-dah-dit Dah-di-dah-dah Di-dah-dah-dit.

Chapter 3: OpenCV

3.1 Overview

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It functions similar to other common libraries in programming languages. With its advantage in real time processing, OpenCV plays a fundamental role in computer vision applications and acceleration of the use of machine perception in the commercial products. As a BSD-licensed product, OpenCV also makes it easy for businesses to utilize and modify the code [7].

3.2 Algorithms and Usage

More than 2500 optimized algorithms were included into OpenCV now, like a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, stich images together to produce a high resolution image of an entire scene [8].

3.3 Development Platform

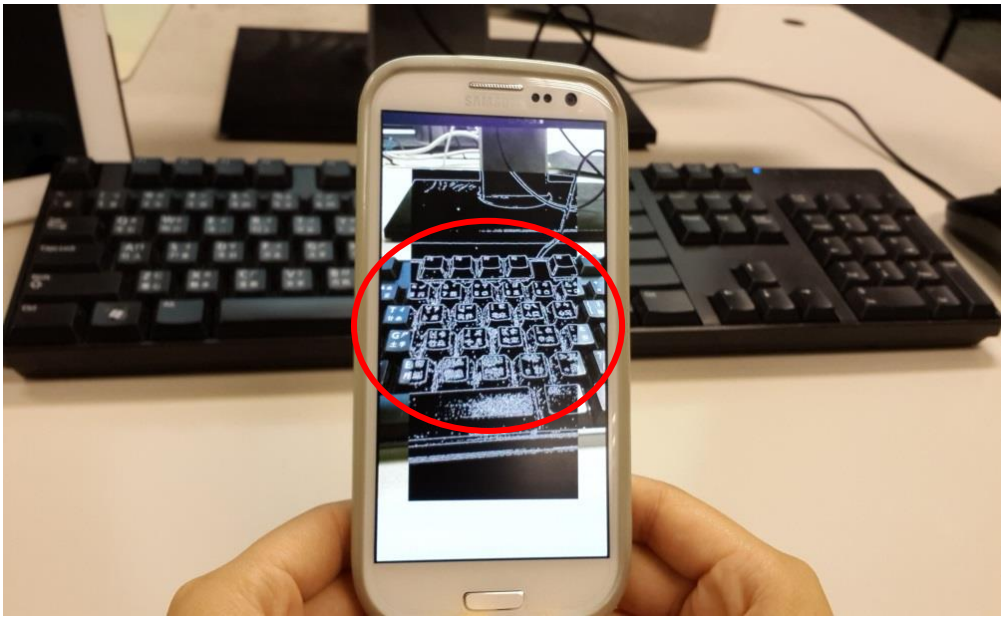
OpenCV was designed to be cross-platform. The library was written in C which makes OpenCV portable to almost any commercial system, from PowerPC Macs to robotic dogs. OpenCV also includes its traditional C interface and the new C++ one; Python and Java were included as well which encourage a wider audience's using. OpenCV's running on both desktop (Windows, Linux, Android, MacOS, FreeBSD, OpenBSD) and mobile (Android, Maemo, iOS) encouraged a variety of developers at the same time [9].

3.4 OpenCV and Android [10]

Since 2010 OpenCV was ported to the Android environment, it allows using the full power of the library in the development of mobile applications. Several fundamental applications in Android are introduced below:

3.4.1 Canny edge detection

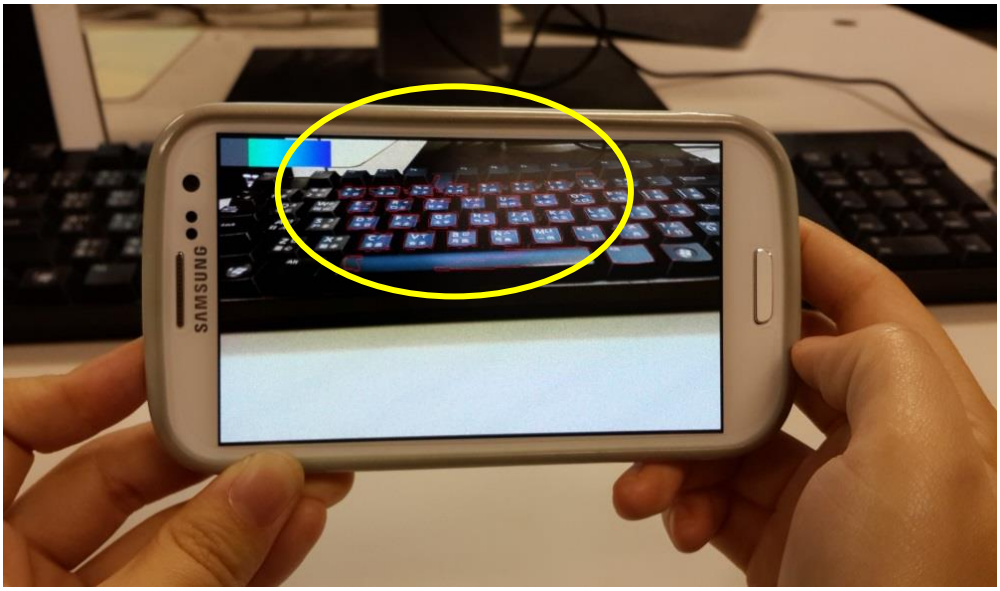
It was implemented by a canny edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. Implementation in Android with the help of OpenCV is shown below:



We can see that the edges of the keyboard were detected and displayed in a different way.

3.4.2 Color blob detection

A trivial color blob tracker was implemented in Android through the use of OpenCV. When user points to some region, the algorithm tries to select the whole blob of a similar color. Work with touch interface and contours are demonstrated in this way:



After we point the camera to the keyboard, parts with similar color (were draw in the red line part) were selected successfully.

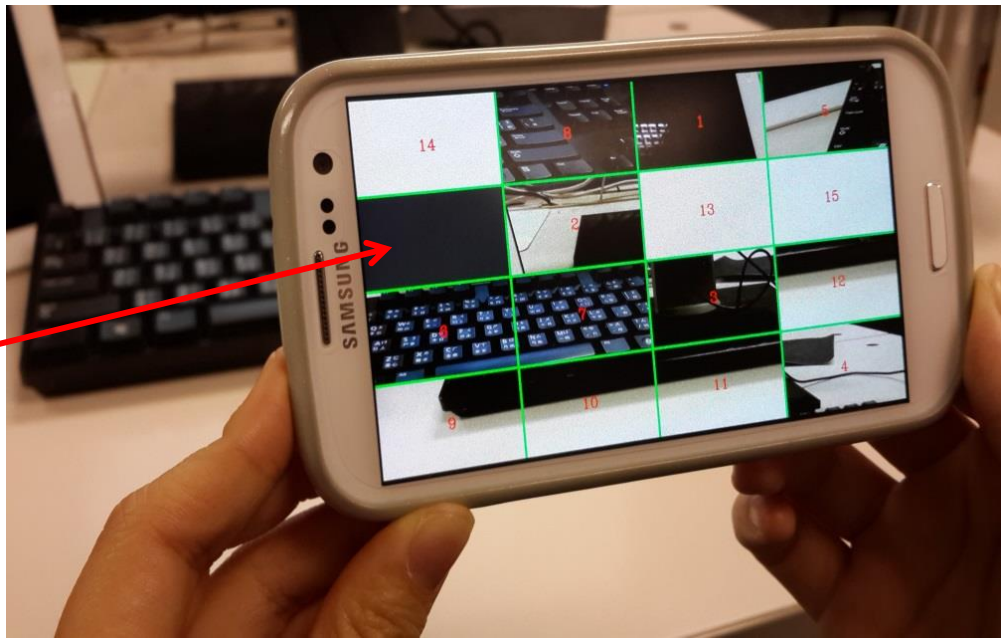
3.4.3 Face Detection

It is the simplest implementation of the face detection functionality on Android. It supports 2 modes of execution: available by default Java wrapper for the cascade classifier, and manually crafted JNI call to a native class which supports tracking. Even Java version is able to show close to the real-time performance on a Google Nexus One device [11].

We used a picture searched from the Internet to do the test. We can see that the face of the person was marked in the phone.

3.4.4 Puzzle

A simple puzzle game was implemented by dividing the real time image into 15 parts and disrupting the order of those small images. It can be developed with just a few calls to OpenCV.



The picture captured by the phone kept changing all the time. We can move each part to the empty slot in order to make it locate in right position. In this way, we can recover the origin picture.

3.5 OpenCV and Our App

Our App's Morse Code detection part mainly depends on the use of OpenCV. OpenCV provides a platform for us to process the image in real time. Its high efficiency in image processing makes the detection of Morse Code Possible. After obtaining frames of images in every second, those frames will then be analyzed to detect the ON/OFF condition of the flash light, the duration of each flash light and the flashing rule of the light. All these information contribute a lot to the decoding of the Morse Code.

Chapter 4: Problems in Environment Setup

4.1 Overview

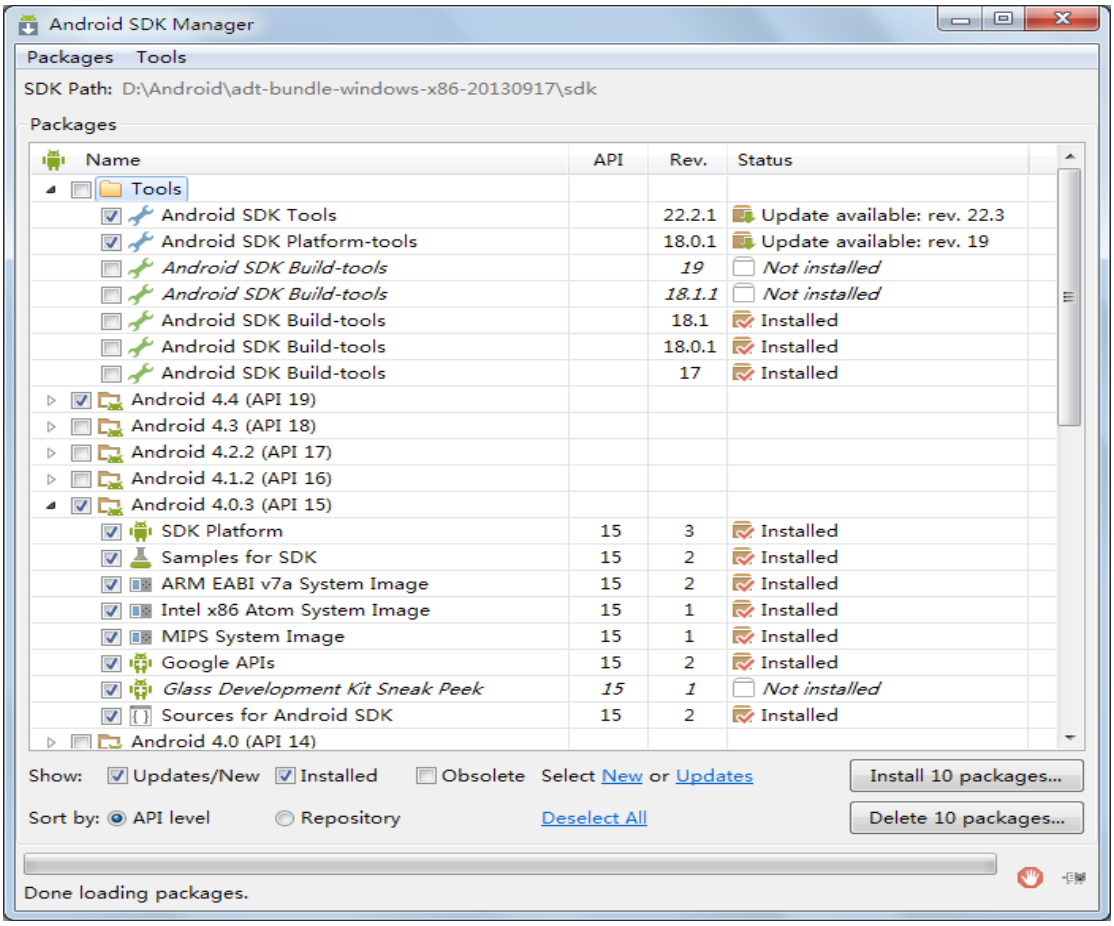
Our application is built in Android Operating System (OS) and developed in Eclipse combined with Android SDK in Windows. OpenCV is used as the main library in our project. The thing we need to do is to install Eclipse into our computer and set up all the parameters for the project development. However, things didn't go that smoothly. We met several problems in the setting process.

4.2 Problems in installing Eclipse and Android SDK

Eclipse provides an integrated development platform for us to develop application in Java basically. Android SDK is a Software Development Kit which provides a comprehensive set of development tools for developers.

Follow the normal steps of installing software to install Eclipse and SDK, we couldn't run the project correctly. Then we found that if they are not installed in the same folder, several path parameters need to be settled to ensure that the project can be built successfully which may bring a lot of trouble.

After all the installation, open eclipse and set configuration to SDK in this way:



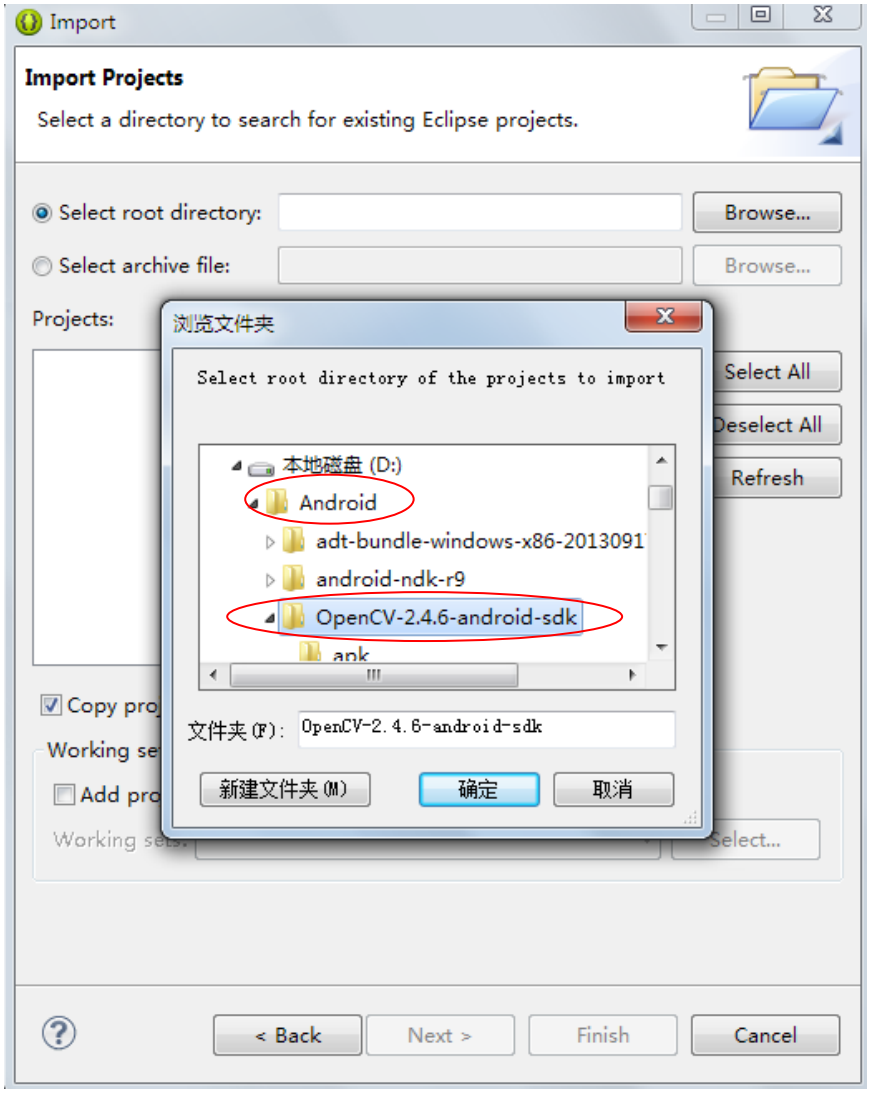
In the above interface, corresponding

- **Tools** and
- **Versions** of Android OS

need to be selected to make it possible for our application to run in a range of Android mobile phone. At first we didn't install necessary tools, the project couldn't run successfully as well. After doing things above, the lowest running environment for our application can be ensured.

4.3 Problems in importing OpenCV into the workspace

OpenCV plays the most important role in the whole development process. Necessary libraries, efficient algorithms, useful image processing tools are all provided by it. When we tried to run the samples in the folder, errors like “library can’t be found” appeared. So we found that we need to import like this:

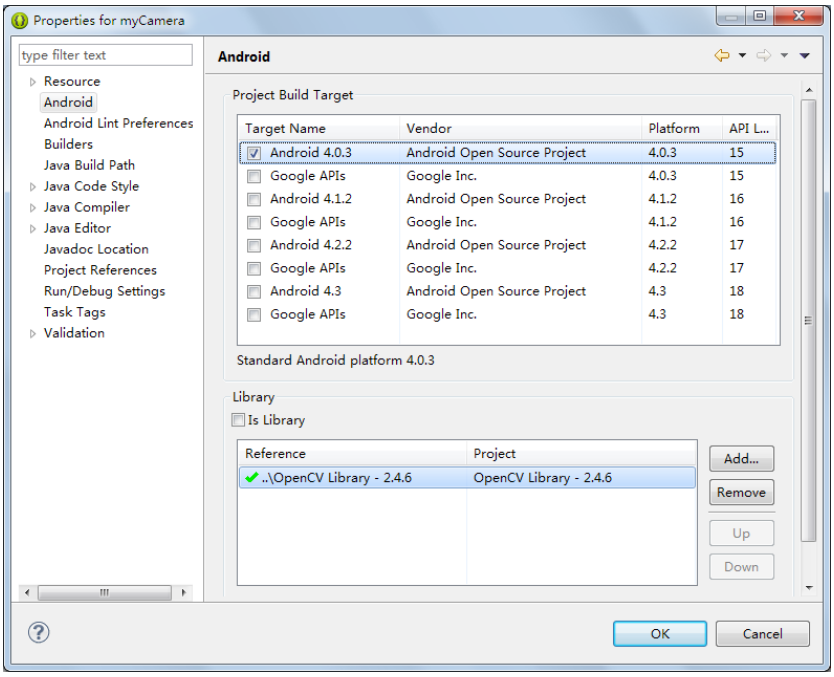


Only when the OpenCV folder was located in the same root directory can the later project run successfully. Otherwise, many other cumbersome parameters need to be changed achieve the same goal.

4.4 Problems in building a new project

Building a new Android Application project in Eclipse is an easy thing. However, when we tried to run it, several problems appeared, like library doesn't exist in some path or no appropriate Android device can be found. Thus we found that several parameters need to be changed to build the project successfully:

- Versions of Android OS
- OpenCV library

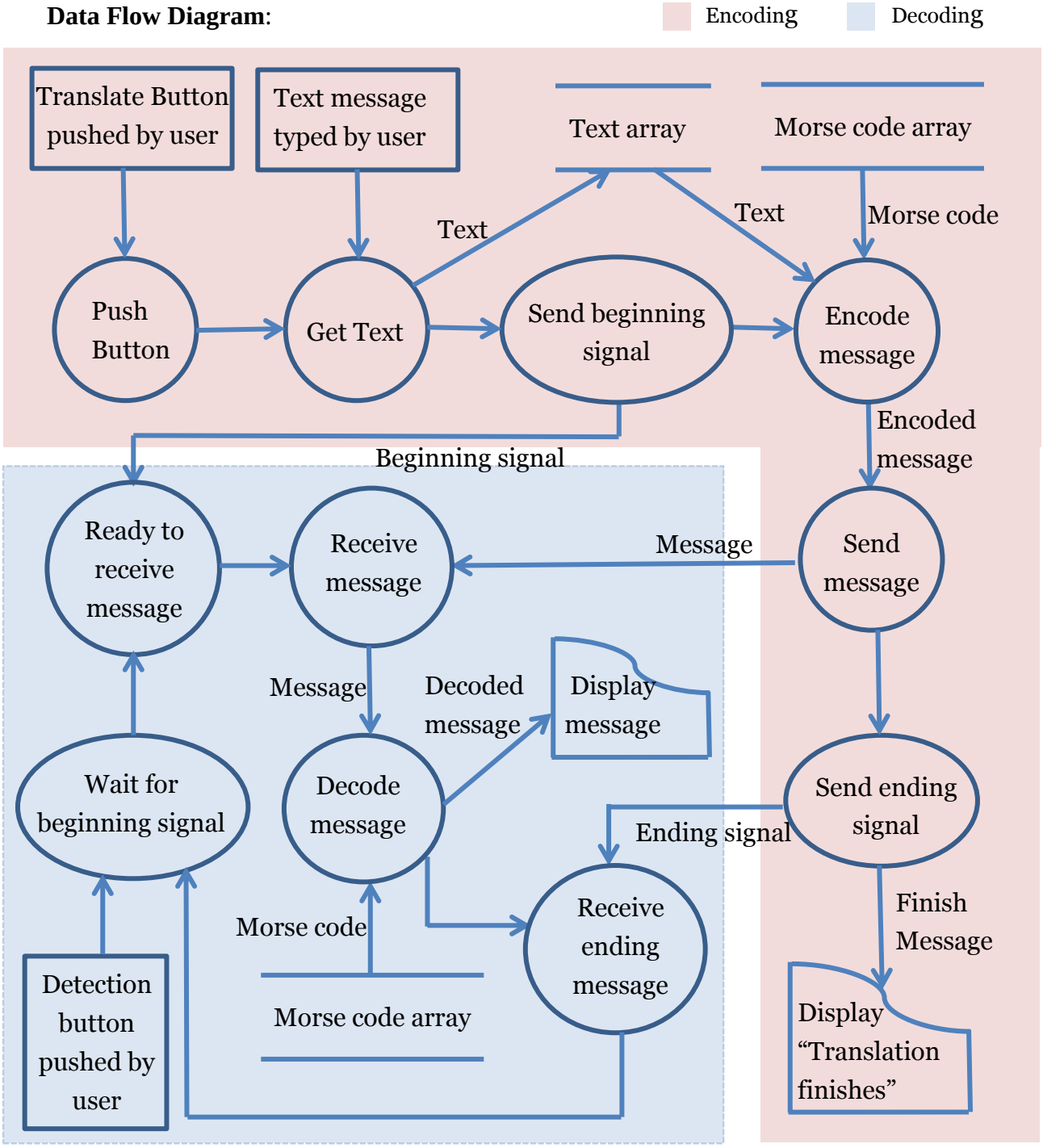


Just click on the lowest version of platform we need and add OpenCV library into the Library part, the sample project could run successfully.

Chapter 5: Design and Implementation

5.1 Design Overview

Data Flow Diagram:



5.2 Encoding

5.2.1 UI Design

- Input message. See figure 5.1.1-1.
- When translation begins, the button turns blue. See figure 5.1.1-2.
- When the encoding is finished, the note “Translation finishes” will appear and the button returns to gray. See figure 5.1.1-3.
- Click twice to exit the app. The first click will appear note “click again to exit” at the bottom of the screen. The second click will totally exit. See figure 5.1.1-4.

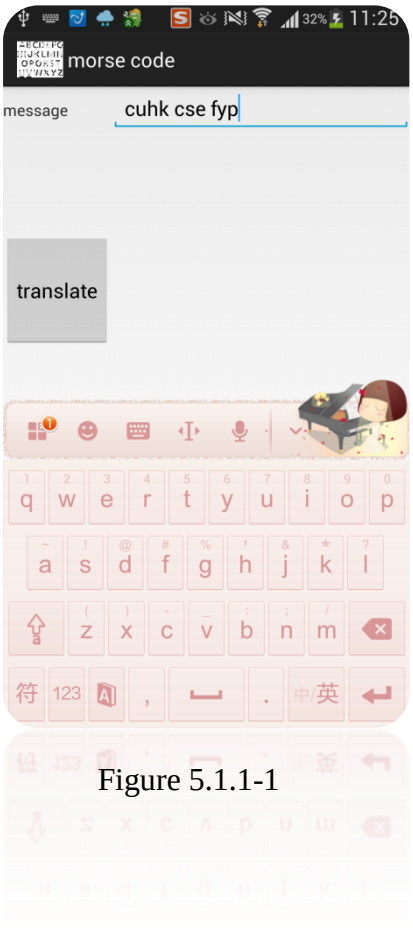


Figure 5.1.1-1

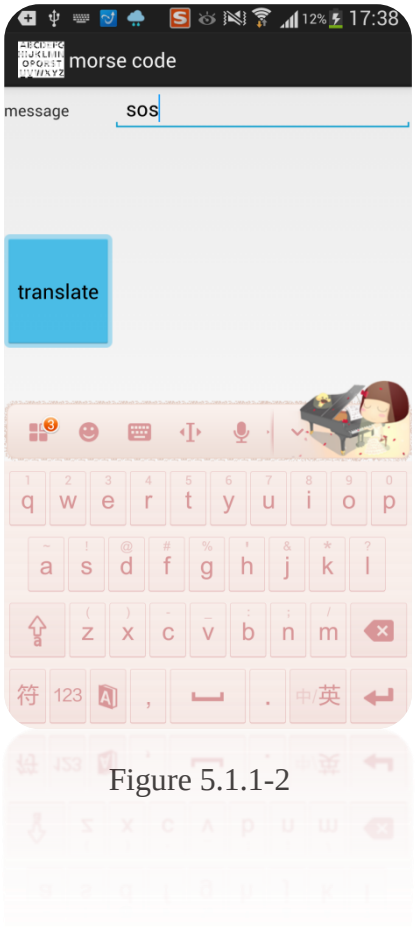


Figure 5.1.1-2

Figure 5.1.1-3

Figure 5.1.1-4

5.2.2 Flashlight control [12]

1) Declaring permissions :

To open the flashlight, we have to use the android API *android.hardware* and the class *camera*.

To access the device camera, we must declare the CAMERA permissions in the Android Manifest. Because we use the camera and flashlight features, the Manifest includes the following:

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-feature android:name="android.hardware.camera" />

<uses-feature android:name="android.hardware.FLASHLIGHT" />
```

2) Open camera:

Obtain an instance of Camera from using open(int).

```
private Camera camera = null;

if ( null == camera )

    camera = Camera.open();
```

3) Call startPreview() to start updating the preview surface.

```
camera.startPreview();
```

4) Get existing (defaut) settings with getParameters(). If necessary (open and close flashlight), modify the returned Camera.Parameters object and call setParameters(Camera.Parameters).

```
Camera.Parameters parameters = camera.getParameters();
```

5) Open Flashlight:

To open flashlight, modify the parameter to *FLASH_MODE_TORCH* mode and call setParameters(Camera.Parameters).

```
//open flashlight

private Parameters parameters = null;
```

```
parameters.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
camera.setParameters(parameters);
```

6) Close Flashlight:

To close flashlight, modify the parameter to *FLASH_MODE_OFF* mode and call setParameters(Camera.Parameters).

```
//close flashlight
parameters.setFlashMode(Parameters.FLASH_MODE_OFF);
camera.setParameters(parameters);
```

7) Release Camera:

Call stopPreview() to stop updating the preview surface.

Call release() to release the camera for use by other applications.

```
if ( camera != null )
{
    camera.stopPreview();
    camera.release();
    camera = null;
}
```

5.2.3 Time control

1) Open duration control

lastTime records the beginning time when the flashlight opens.

curTime records what time it is now.

openTime records how long the flashlight has opened, that is, openTime = curTime – lastTime.

codeTime is the duration that the flashlight should open.

At the beginning, lastTime = curTime, flashlight opens and the program enters an while loop. The program will not jump out of the loop until openTime = codeTime, and then the flashlight closes.

2) Stop duration between dit and dah:

unit denotes the duration of one dit.

Similar to open time control, at the beginning, lastTime = curTime, flashlight closes and the program enters an while loop. The program will not jump out of the loop until curTime - lastTime = unit, and then the flashlight is on.

3) stop time between symbols or words

stopTime = **unit** * 0.3 for stop duration between symbols;

stopTime = **unit** * 0.7 for stop duration between words.

Similar to the above, at the beginning, lastTime = curTime, flashlight closes and the program enters an while loop. The program will not jump out of the loop until curTime - lastTime = stopTime, and then the flashlight is on.

5.2.4 Encoding Morse code

1) Save the Morse code in a two-dimensional array **int**[][] code. Each row of the array save the corresponding Morse code of a symbol (a letter or a punctuation). Instead of using dot and dash, we use digits 1 and 3 to represent dit and dah, respectively.

2) Write a function code_index(char symbol) to return an integer that points to the index where the symbol is in the array code[][]. For example, code_index（A）= 0, then we have the modified Morse code for A is {1, 3} (See table 5.2.4-1), which means the Morse code for A is “dit dah”. Since we have codeTime = array[i][j] * unit, for A, the flashlight will be on for 1 unit time, off for 1 unit time, and on for 3 units time, and finally off. For words, for example, “A A”, the on-off sequence is “on(1 unit), off(1 unit), on(3 units), off(7 units), on(1 unit), off(1 unit), on(3 units), off(until the next decoding process)”.

The entire encoding process is indicated by the Process Flow Chart 5.2.4-2.

Table 5.2.4-1: Modified Morse code Table

Index	Code[index]	Correspon-ding symbol	Index	Code[index]	Correspon-ding symbol
0	{1, 3}	A/a	27	{1, 3, 3, 3, 3}	1
1	{3, 1, 1, 1}	B/b	28	{1, 1, 3, 3, 3}	2
2	{3, 1, 3, 1}	C/c	29	{1, 1, 1, 3, 3}	3
3	{3, 1, 1}	D/d	30	{1, 1, 1, 1, 3}	4
4	{1}	E/e	31	{1, 1, 1, 1, 1}	5
5	{1, 1, 3, 1}	F/f	32	{3, 1, 1, 1, 1}	6
6	{3, 3, 1}	G/g	33	{3, 3, 1, 1, 1}	7
7	{1, 1, 1, 1}	H/h	34	{3, 3, 3, 1, 1}	8
8	{1, 1}	I/i	35	{3, 3, 3, 3, 1}	9
9	{1, 3, 3, 3}	J/j	36	{1, 3, 1, 3, 1, 3}	.
10	{3, 1, 3}	K/k	37	{3, 3, 1, 1, 3, 3}	,
11	{1, 3, 1, 1}	L/l	38	{1, 1, 3, 3, 1, 1}	?
12	{3, 3}	M/m	39	{1, 3, 3, 3, 3, 1}	’
13	{3, 1}	N/n	40	{3, 1, 3, 1, 3, 3}	!
14	{3, 3, 3}	O/o	41	{3, 1, 1, 3, 1}	/
15	{1, 3, 3, 1}	P/p	42	{3, 1, 3, 3, 1}	(
16	{3, 3, 1, 3}	Q/q	43	{3, 1, 3, 3, 1, 3}	)
17	{1, 3, 1}	R/r	44	{1, 3, 1, 1, 1}	&
18	{1, 1, 1}	S/s	45	{3, 3, 3, 1, 1, 1}	:
19	{3}	T/t	46	{3, 1, 3, 1, 3, 1}	;
20	{1, 1, 3}	U/u	47	{3, 1, 1, 1, 3}	=
21	{1, 1, 1, 3}	V/v	48	{1, 3, 1, 3, 1}	+
22	{1, 3, 3}	W/w	49	{3, 1, 1, 1, 1, 3}	-
23	{3, 1, 1, 3}	X/x	50	{1, 1, 3, 3, 1, 3}	_
24	{3, 1, 3, 3}	Y/y	51	{1, 3, 1, 1, 3, 1}	”
25	{3, 3, 1, 1}	Z/z	52	{1, 1, 1, 3, 1, 1, 3}	$
26	{3, 3, 3, 3, 3}	0	53	{1, 3, 3, 1, 3, 1}	@

cm[]: the array of message inputted. **len:** length of cm[]. **‘ ‘:** space

index: the index of Morse code array. **Code[][]:** The Morse code array.

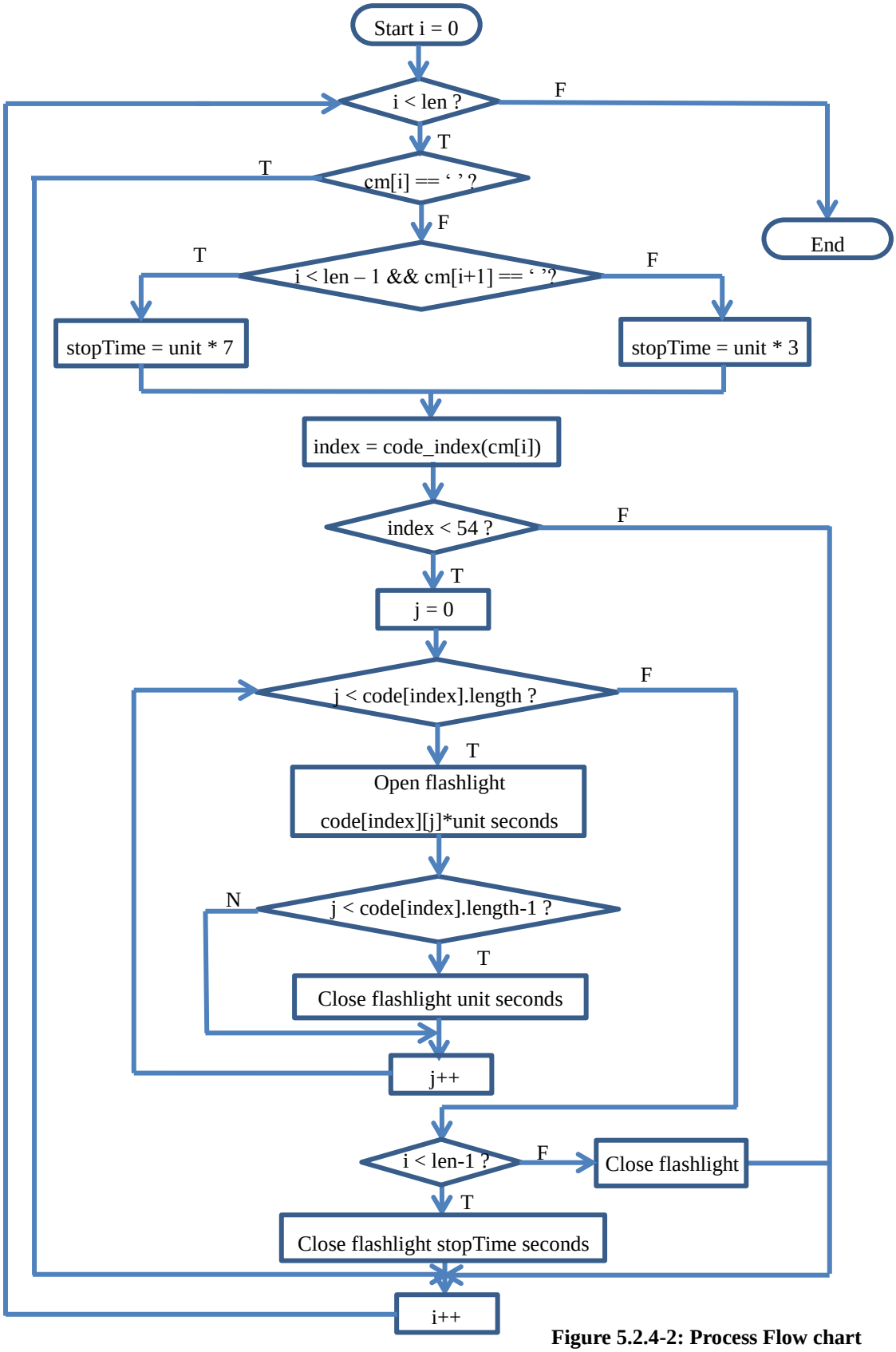


Figure 5.2.4-2: Process Flow chart

Explanation of Process Flow Chart:

Set the message "A A" as an example again. cm[] = {'A', ' ', 'A'}.

The variables changes as the following table.

And the descriptions of each step are introduced later.

Step	i	len = cm[].length	cm[i]	stopTime (s)	Index = code_index(cm[i])	j	code[index].length	Code[index][j]
1	0							
2	0	3						
3	0	3	'A'					
4	0	3	'A'	7 units				
5	0	3	'A'	7 units	0			
6	0	3	'A'	7 units	0	0		
7	0	3	'A'	7 units	0	0	2	
8-9	0	3	'A'	7 units	0	0	2	1
10	0	3	'A'	7units	0	1	2	
11	0	3	'A'	7 units	0	1	2	3
12-13	0	3	'A'	7 units	0	2	2	
14	1	3						
15	1	3	' '					
16	2	3						
17	2	3	'A'					
18	2	3	'A'	3 units				
19	2	3	'A'	3 units	0			
20	2	3	'A'	3 units	0	0		
21	2	3	'A'	3 units	0	0	2	
22-23	2	3	'A'	3 units	0	0	2	1
24	2	3	'A'	3 units	0	1	2	
25	2	3	'A'	3 units	0	1	2	3
26	2	3	'A'	3 units	0	2	2	
27	2	3	'A'	3 units	0	2	2	
28	3							
29	3							

1. At the beginning, i = 0. The program jumps in the first for loop.
2. Since the length of "A A" len = 3 and i < 3, the program continues and goes to the next step.
3. Since cm[0] is 'A', not space, the program goes to the next conditional statement.
4. Since i < len – 1 && cm[i+1] == ' ', stopTime = unit * 7.
5. Since code_index('A') = 0, index = code_index(cm[i]) = 0.
6. Since index = 0 < 54, the program goes to the second for loop and j = 0.
7. Since code[0] = {1, 3}, code[index].length = 2.
8. Since j < 2, the flashlight is on and lasts for 1 unit second (code[index][j] = 1).
9. Since code[index].length – 1 = 1 and j < 1, the flashlight in off for 1 unit second.
10. j = j + 1 = 1.
11. Since j = 1 < 2, the second for loop continues and the flashlight is on for 3 unit seconds (code[index][j] = 3).
12. Since code[index].length – 1 = 1 and j = 1, j = j + 1 = 2.
13. Since j = 2, the program jumps out of the second for loop and the flashlight closes for 7 unit seconds (stopTime = unit * 7) since i = 0 < len -1.
14. i = i + 1 = 1. The program continues the first for loop since i < 2.
15. Since cm[1] is a space, i = i + 1 = 2.

16. The first for loop continues since i < len.

17. Similar to step 3.

18. Since i = len – 1, stopTime = unit * 3.

19-26. Similar to steps 5-12.

27. Since j = 2, the program jumps out of the second for loop and the flashlight closes finally since i = 2 = len – 1.

28. i = i + 1 = 3.

29. Since i = 3 = len, the program jumps out of the first for loop and ends the encoding process.

5.3 Decoding

5.3.1 Overview

The decode part is basically is basically an image processing thing. What we need to do can be simply explained as gets the image and analyzes the image. Details can be realized in this way:

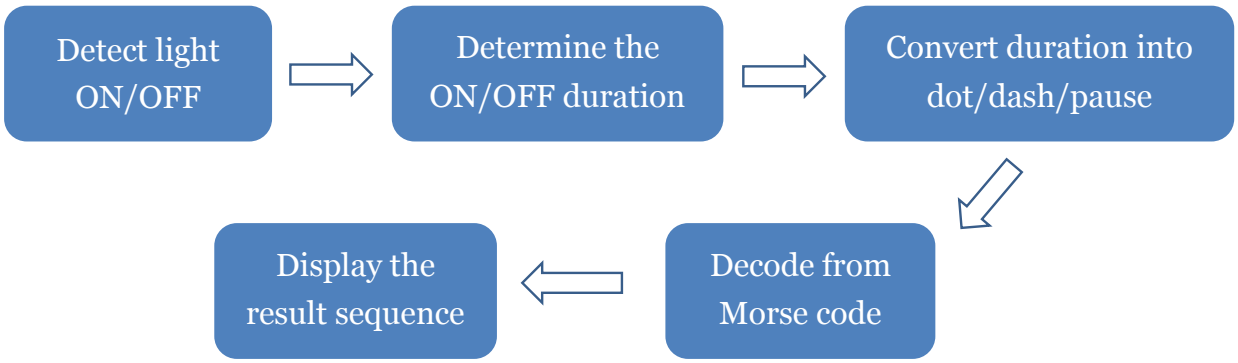


Figure 5.3.1-1

The problems here are how we can address image efficiently (when the image changes all the time) and how we can decide the light ON/OFF time precisely. Different devices' cameras have different fps (frames per second) as well. The emission frequency is seriously limited by the fps value. If the emission frequency was not controlled appropriately, the reception part may not able to receive the flash light frequency correctly.

5.3.2 Open the camera

Firstly, override the **LoaderCallbackInterface** which add OpenCV library initialization to our activity. The camera was activated here.

We want to make use of some efficient algorithms in OpenCV, so the class **JavaCameraView** in OpenCV was used to get each frame of the real time image in an efficient manner. Have a look on how it works:

JavaCameraView:

It's an implementation of the Bridge View between OpenCV and Java Camera. Through the **connectCamera** which opens Java camera and sets the **PreviwCallback** to be delivered, preview frame can be obtained through this callback.

When frame is delivered via the above callback, it processed via OpenCV to be converted to RGBA32 and then passed to the external callback for later use.

Our subsequent work is mainly focused on the frame obtained here.

5.3.3 Set parameters for camera

When the activity is first created, the parameters value need to be settled.

FLAG_KEEP_SCREEN_ON: Keep the screen on all the time.

setVisibility: Set the enable state of the view (which is the view created in step 1) of the activity.

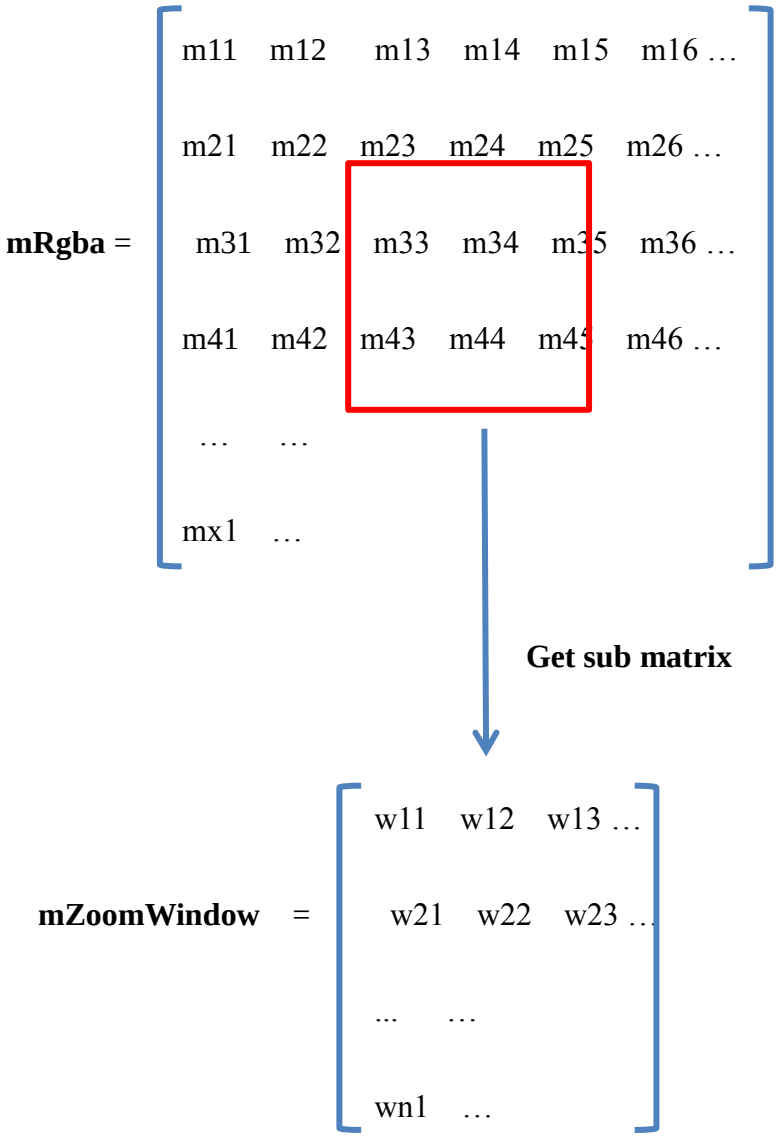
5.3.4 Process frame values

onCameraFrame is the main function here. It is invoked when the delivery of the frame needs to be done which means, as soon as the frame changed, it will be invoked. It provides the current camera frame for us to work on.

1) Draw rectangle on the image

Rectangle is the area we used to limit the range that light can be detected. After we got the RGBA Matrix with frame, we need to calculate the position for the rectangle.

RGBA of the input frame:



The size of **mZoomWindow** depends on the rectangle size we want. After that, we can decide the position the corner of the rectangle. Use **Core.rectangle()** combined with **mZoomWindow** and its position to draw the rectangle on the image.

2) *Determine the threshold value for light on and off*

We've already got the RGBA of rectangle area: mZoomWindow. Then we need to check the value of each point in the rectangle area under different light condition.

At first, we use **Core.sumElems** to compute the sum of all the elements in the above matrix. And then we got the average value of the sum to get a rough idea about RGBA value when the light is ON. In this way, we need to ensure that the light fully fills the rectangle. Although the implementation is limited to that condition, we got the rough threshold value for RGBA which can be used later.

Finally, we found a function that can extract every element value from the matrix. Then we did some test on the pixel value and compare to the rough threshold value we find in last method, and got this:

After analyzing the data, we found that pixels with obvious light ON state are inside the blue circle part and pixels with light OFF state are inside the red circle part.

Thus, we choose the threshold value for each channel to be:

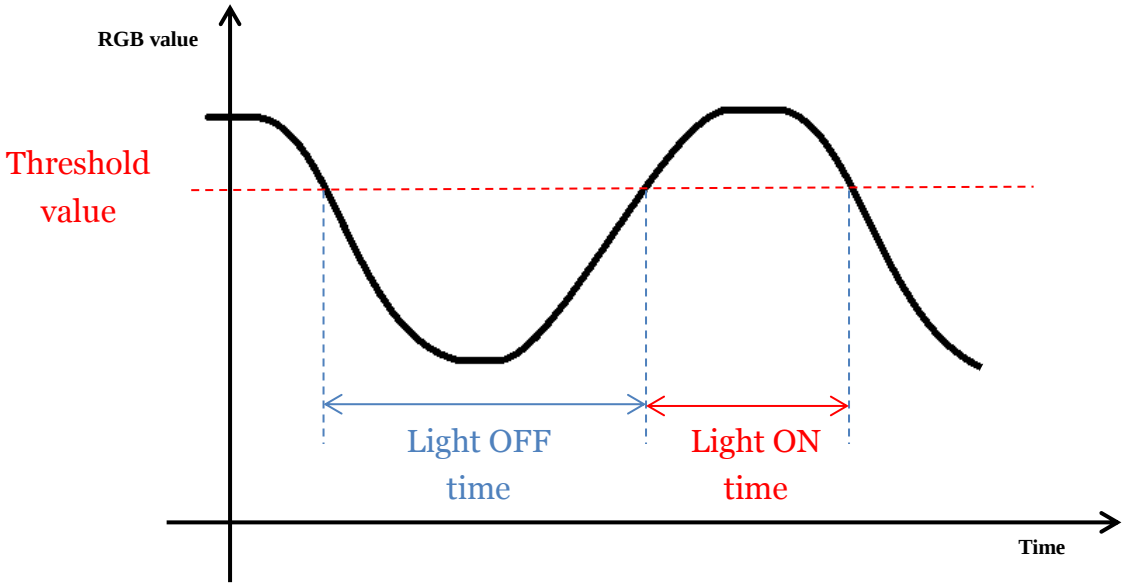
V(R) = 210 V(G) = 210 V(B) = 210

Which means only when V(R) and V(G) and V(B) all exceed 210 can this pixel be defined as light on.

There are so many pixels in this rectangle area, so we need to decide how much percent pixels are above the threshold value can the area be defined as light on. Here we choose P = 10%.

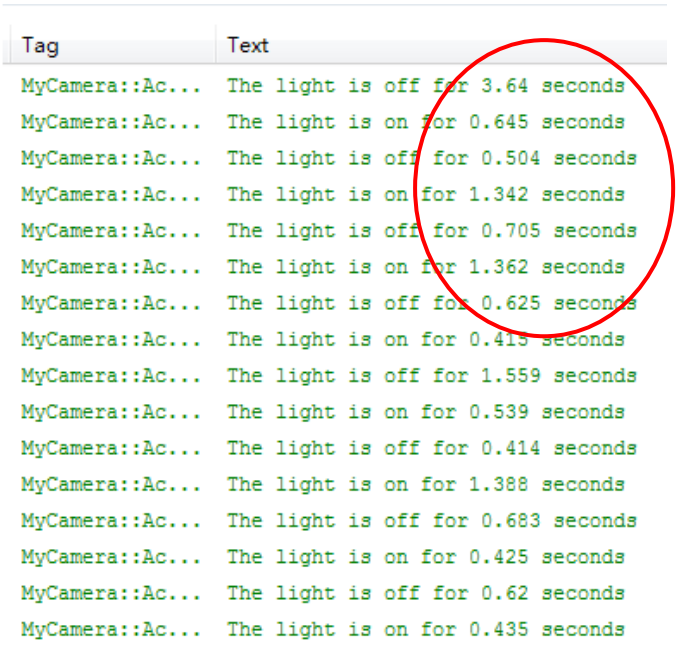
3) *Calculate the duration of light ON/OFF*

According to the light ON/OFF state of the rectangle area, we record the time of the state change to calculate the duration.



However, until now, the threshold value was not that precise which may cause error happen when deciding the duration time. This will be discussed in the experiment part.

4) Analyze the raw data



Tag	Text
MyCamera::Ac...	The light is off for 3.64 seconds
MyCamera::Ac...	The light is on for 0.645 seconds
MyCamera::Ac...	The light is off for 0.504 seconds
MyCamera::Ac...	The light is on for 1.342 seconds
MyCamera::Ac...	The light is off for 0.705 seconds
MyCamera::Ac...	The light is on for 1.362 seconds
MyCamera::Ac...	The light is off for 0.625 seconds
MyCamera::Ac...	The light is on for 0.415 seconds
MyCamera::Ac...	The light is off for 1.559 seconds
MyCamera::Ac...	The light is on for 0.539 seconds
MyCamera::Ac...	The light is off for 0.414 seconds
MyCamera::Ac...	The light is on for 1.388 seconds
MyCamera::Ac...	The light is off for 0.683 seconds
MyCamera::Ac...	The light is on for 0.425 seconds
MyCamera::Ac...	The light is off for 0.62 seconds
MyCamera::Ac...	The light is on for 0.435 seconds

Raw time data

We can see that the duration time is varied in some range, so we need to classify them into dot and dash. If we specify the value of the "dot" time, after several tests, we decide:

$$\text{newTime} = \begin{cases} \text{dot} & 0.7 * \text{dot} < \text{realTime} < 1.3 * \text{dot} \\ 3 * \text{dot} & 2.5 * \text{dot} < \text{realTime} < 3.5 * \text{dot} \\ 7 * \text{dot} & 6.5 * \text{dot} < \text{realTime} < 7.5 * \text{dot} \end{cases}$$

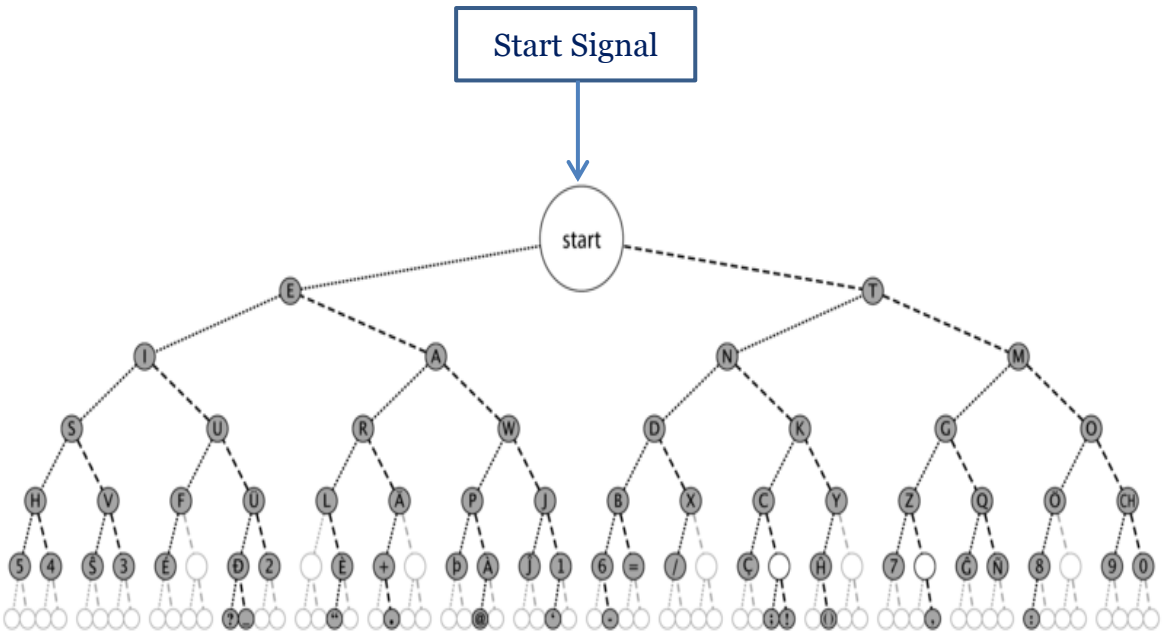
(dash = 3 * dot)

Under this range limiting, the state of light can be determined in a much more flexible manner.

5.3.5 Decode

According to this tree, using if-else-if to decide whether the code exists or not and display correct code.

When come across dot, enter the left sub tree, when come across dash, enter the right sub tree.



Besides the tree above, we also add an end signal. In any node inside the tree, when come across the end signal, just stop decoding.

Chapter 6: Experiments and Testing

To ensure that the final interaction part (encode-decode) goes well, we do testing step by step.

Once one basic part passed through the test, we go to another complex one next.

6.1 Light ON/OFF testing

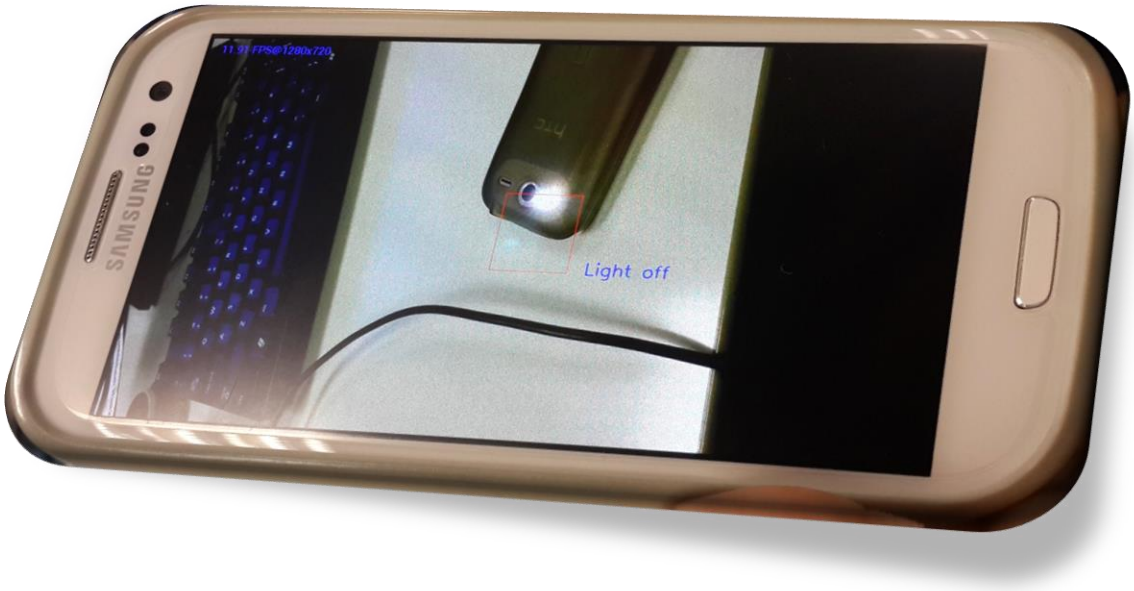
Whether a Morse code can be received directly depends on whether the Light ON/OFF state can be detected correctly. In the program we decided that, when the percent of Light-ON state pixels in the rectangle area is bigger than 10%, the rectangle area should be considered as Light-ON state.

6.1.1 Light ON condition

Denote the percent of Light-ON state pixels in the rectangle area as P.

We can see that, the Light-ON state actually depends on the P:

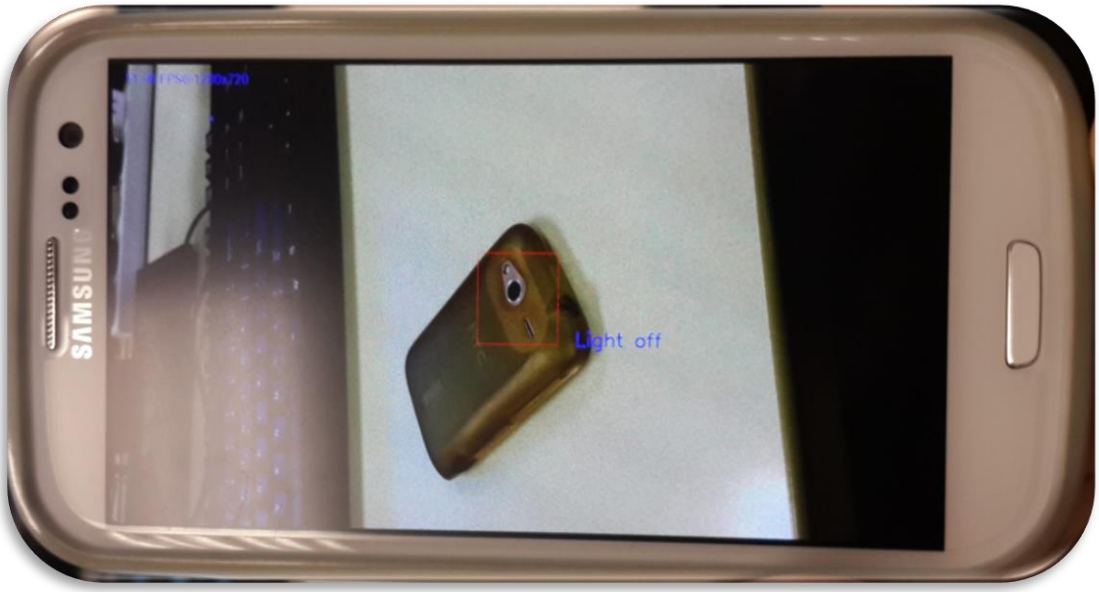
Comparing the two test cases:



The value of P in the second pictures is smaller than the value of P in the firstly picture and the second one is in Light OFF state which means, the threshold value of P actually works.

6.1.2 Light OFF condition

When there is no bright area in the rectangle, the result is correct as well:



Note: Some errors appeared in the test. When we shake the phone, the camera may deflect the environment light and the rectangle area was mistakenly considered as Light-ON state.

6.2 Light ON duration testing

We knew that DOT and DASH in the Morse code have different flash time, so after the light can be detected by our app, the duration of the Light ON/OFF state should be detected as well.

Show the actual light ON time (raw time data) and comparing it with the actual time range:

The time showed in the screen totally matches to the actual time. However, it's not exactly equal to the actual Light ON time. We don't have a precise rule to determine whether the Light is ON or not, because when the light is turning off, its brightness will experience a decay process.

6.3 Distance and transmission rate testing

When the Light Signal sequence is being transmitted, the ON/OFF duration will be influenced by the brightness of the Light Signal, while the brightness of Light Signal will be influenced by the distance between the emission terminal and reception terminal.

In this part, we controlled the variables distance, minimum RGB values and OFF duration ranges respectively to analyze their influence.

6.3.1 Fixed parameters:

Min(R) = 210, Min(G) = 210, Min(B) = 210

Denote Dot duration as Dd.

OFF duration ranges: 2.5 * Dd < OFF duration (between **letters**) < 3.5 * Dd

6.5 * Dd < OFF duration (between **words**) < 7.5 * Dd

Distance Dd(s)(m)	0.25	0.5	0.75	1.0	1.25	1.5
1.0	√	√	√	√	√	√
0.5	X	-	-	X	X	X
0.4	X	X	X	X	X	X

(Note:

"√": the possibility that the message is decoded correctly >= 75%.

"X": the possibility that the message is decoded correctly =< 25%.

"-": the message decoded is partially correct.)

6.3.2 Fixed parameters:

Min(R) = 220, Min(G) = 220, Min(B) = 220

Denote Dot duration as Dd.

OFF duration ranges: 2.0 * Dd < OFF duration (between **letters**) < 4.0 * Dd

6.0 * Dd < OFF duration (between **words**) < 8.0 * Dd

Distance Dd(s) (m)	0.25	0.5	0.75	1.0	1.25	1.5
1.0	√	√	√	√	√	√
0.5	√	√	√	√	√	-
0.4	X	X	X	X	X	X

(Note:

"√": the possibility that the message is decoded correctly >= 75%.

"X": the possibility that the message is decoded correctly =< 25%.

"-": the message decoded is partially correct.)

6.3.3 Fixed parameters:

Min(R) = 210, Min(G) = 210, Min(B) = 210

Denote Dot duration as Dd.

OFF duration ranges: 2.0 * Dd < OFF duration (between **letters**) < 4.0 * Dd

6.0 * Dd < OFF duration (between **words**) < 8.0 * Dd

Distance Dd(s) (m)	0.25	0.5	0.75	1.0	1.25	1.5
1.0	√	√	√	√	√	√
0.5	√	√	√	√	√	√
0.4	X	X	X	X	X	X

6.4 Results Analysis

1) Given a Dot duration, minimum RGBA values and the OFF duration ranges, the accuracy is high in the middle range of time duration. Going to the two ends of that range, the accuracy gradually decreases. There is a brightness increasing process when the flash light was opened and there is a brightness decreasing process when the flash light was turned off. Therefore, the RGBA value was influenced by these two processes that the accuracy was influenced as well.

2) When the Dot duration decreases, the distance between the emission terminal and the reception terminal should be longer enough so that the Light state can be determined correctly in some degree.

3) The longer the distance between the emission terminal and the reception terminal is, the weaker the brightness of the light is. In this way, if the threshold value of RGB is too high, the number of Light-ON-state pixels that can be detected will decrease, which means the Light ON time becomes shorter and the Light OFF time becomes longer. Error happens easily.

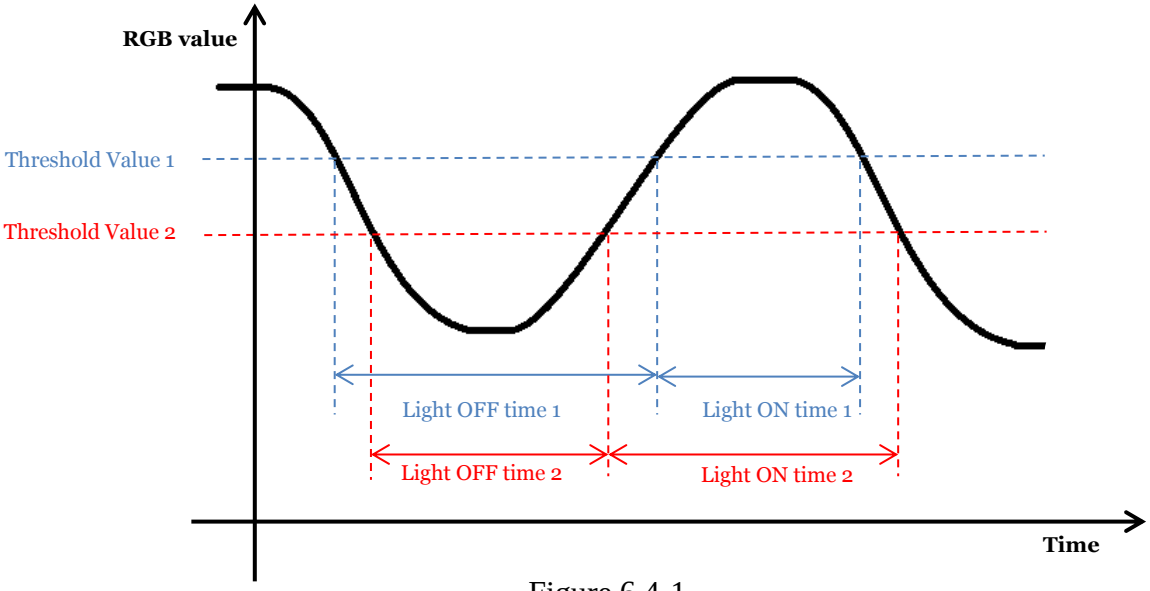


Figure 6.4-1

At the same time, the lower the threshold value is, the shorter the Light OFF time is and the longer the Light ON time is. When the threshold value is too low, sometimes the flashlight state will be falsely considered to be ON, while it should have been detected to be OFF, which may cause error as well.

Therefore, we need to find a threshold value and the rough corresponding Light OFF duration and Light ON duration to ensure that most messages can be decoded correctly.

4) The relationship of emitting frequency and receiving frequency

For signal emission and reception, we need to ensure that the receiving frequency is 2 times the emitting frequency at least. We've already known that the approximate value of FPS of the receiving terminal, according to the above rule, the emitting period, which is "1 / (Dot duration)", should be 0.5 times the FPS at least.

In our project, we found that the camera's receiving frequency is approximate 8 fps which means the camera can receive 8 frames per second at most. Then, the 1 / (Dot duration) should be at least 0.5 * 8 = 4 Hz, which means

Min (Dot duration) = 1 / 4 second = 0.25 second.

However, the limitation is now we can only receive correctly when Dot duration >= 0.5 second.

Here is the reason why Min (Dot duration) = 2 / FPS:

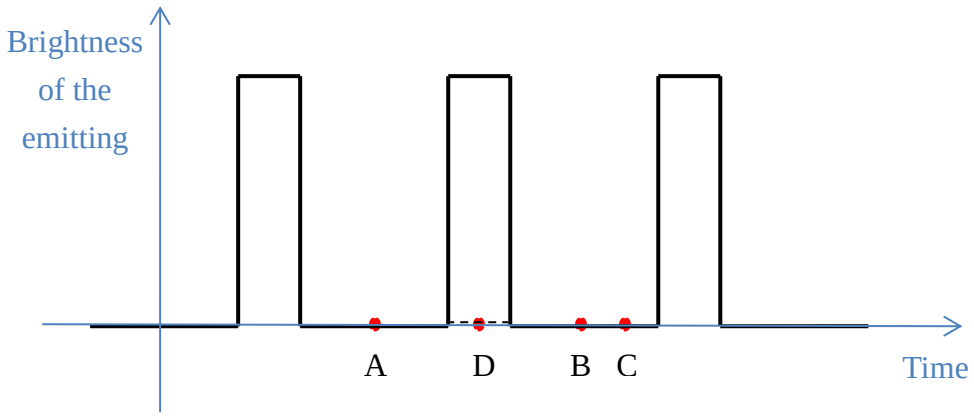
For the emission part, suppose the emitting frequency

F1 = n Hz, Dot duration = 1 / n seconds.

Denote the frequency of the receiving part as F2 which means FPS = F2.

And receiving period T = 1 / F2 second.

The timing diagram of the emitting Light shown below:



If F2 = F1 = n, suppose it receives the first emitting message at time point A. Then it will receive the next message after 1 / n second, which is in point B. We can see that it misses the peak part in interval [A, B] which means the peak message can't be received. Then, error happens.

If F2 < F1, we also suppose it receives the first emitting message at time point A. We calculate that T2 = 1 / F2 > 1 / F1 = 1 / n, which means the next receiving point is in the right of point B, for example in point C. It misses the peak again.

Therefore, we need to ensure that the receiving terminal can receive the "peak message" in interval [A, B] at least. It means that the next receiving point must between point A and point D. When it's in D, we got the maximum value for T2, which implies

T2 <= (Dot duration / 2)

F2 >= 2 / (Dot duration)

Thus, Dot duration >= (2 / F2) which means

Min (Dot duration) = 2 / F2 = 2 / FPS.

6.5 Symbol testing

According the above testing, we choose parameter as follows, which makes the testing more stable, to test all the letters/numbers/punctuations in the Morse code and the results are correct. (See figure 6.5-1 and 6.5-2.)

Distance: 0.5m

RGB values: Min(R) = 210, Min(G) = 210, Min(B) = 210

OFF duration ranges: 2.0 * Dd < OFF duration (between **letters**) < 4.0 * Dd

6.0 * Dd < OFF duration (between **words**) < 8.0 * Dd

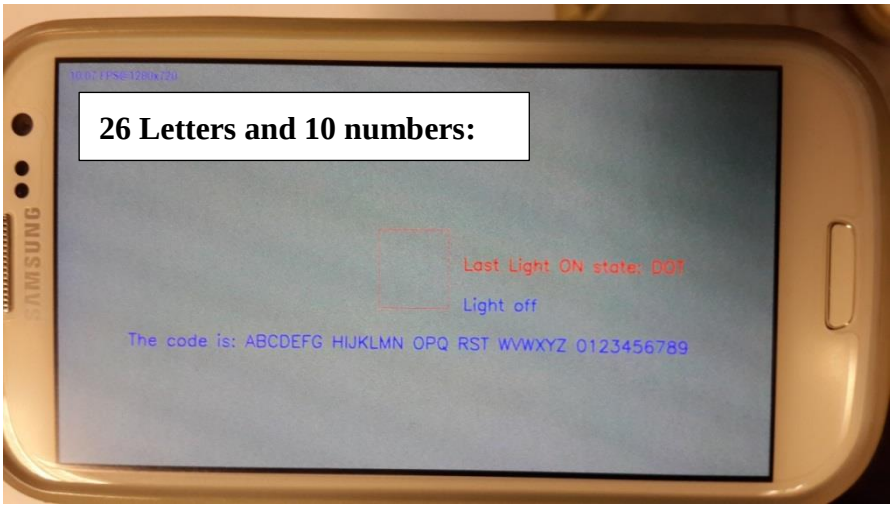


Figure 6.5-1

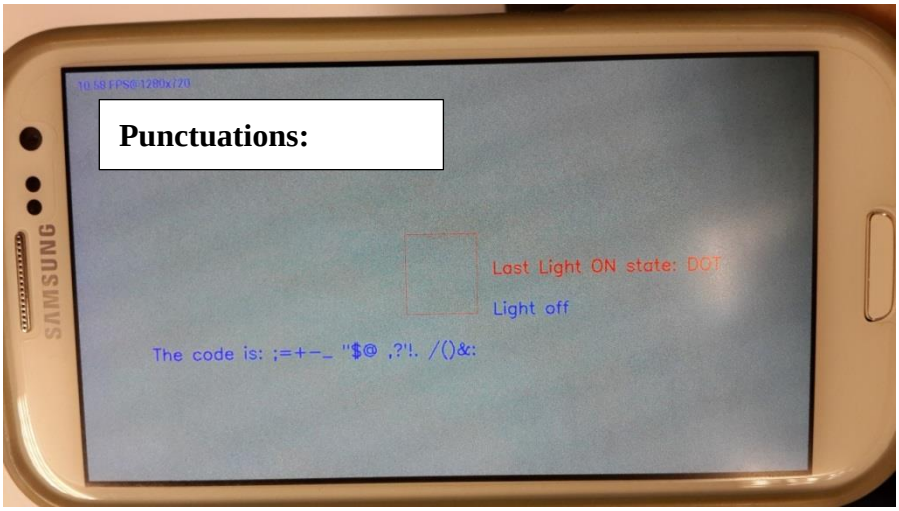


Figure 6.5-2

Chapter 7: Conclusion

7.1 Progress

The encoding part and the decoding part are implemented simultaneously.

7.1.1 Encoding

1) Study on Android programming;
2) Try to write a simple android program to open and close the flash light of android device;
3) Try to control the on and off duration of the flash light;
4) Add the function that a user can type some text to a textbox and click the "translate" button, and then the flash light will continue opening with different durations of different letters;
5) Study the Morse code and try to implement the encoding;
6) Testing.

7.1.2 Decoding

1) Study on Android programming and OpenCV;
2) Try to open the camera preview and know the working mechanism;
3) Obtain the average RGBA value of a limit area and obtain the RGBA value of each pixel in the rectangle;
4) Optimize the detection area and get the threshold value of the state of the light;
5) Analyze the threshold value and determine the Light ON/OFF state;
6) Calculate the duration of the light ON state and the duration of light OFF state;
7) Analyze the raw data and classify each flashlight into “dot” and “dash, then letter and word;
8) Decode the flashlight and display the result;
9) Testing.

7.2 Difficulties

7.2.1 Encoding

1) At first, we didn't know how to use android programming to control the flashlight. We seek the solutions on the internet, but some methods were not compatible with out device. Finally, we found a method, modified it and controlled the flashlight successfully.

2) At the very beginning, the first symbol of one message is always decoded falsely. We thought that the decoding part is responsible for computing the on and off durations of flashlight, so the problem must be in the decoding part, but we were wrong. Somehow we finally found that it was the problem about encoding. The problem is about time variables declarations and the process procedure. At first, we declared

"**double** lastTime = System.*currentTimeMillis*()/1000;",

but the return value of System.*currentTimeMillis*() should be **long**.

Then we modified the codes to as follows:

```
long lastTime = System.currentTimeMillis();
long curTime = System.currentTimeMillis();
double openTime = (double)(curTime - lastTime) / 1000.0;
```

3) Since when we meet a space while encoding, the flashlight should be off 7 unit

seconds, the process between two words is a little complex. At the beginning, we meet some problem. For example, consider the input message “cuhk cse”, the flashlight will be off 3 unit seconds after very letter. However, there is a space after the last letter “k” and the flashlight will be off 7 unit seconds for that space. Overall, after “k”, the flashlight will be off 10 unit seconds, which is not correct. And we added some conditional statement to fix it out.

7.2.2 Decoding

1) No idea about android programming and OpenCV, all we can do is start from scratch. We went through every example in the OpenCV tutorial and tried to figure out what those functions do. After that, we got functions we need and applied it to the Light detection part.

2) At first, our program can only detect the light that totally fills the rectangle. We got the sub matrix we need from a matrix and used functions to calculate the sum of all the elements in that sub matrix. Then we used the result to obtain the average value of the sum, in this way we got the average RGBA value of the rectangle area.

   Finally we find a way to extract the value of each element in the matrix, so that we can decide the RGBA value of each pixel in the rectangle area. When there are above

10% pixels in the rectangle area satisfy the Light-ON condition, the rectangle area then can be considered as Light ON.

3) The emission Light ON/OFF time is not exactly equals to the Light ON/OFF that was received. Then we need to determine the Light ON duration that was corresponding to a DOT/DASH, while it is a little hard. Because in the ideal state, the inter-element gap between dot and dash is one time unit, while the actual time being detected fluctuated up and down with respect to the ideal one. We experimented on it for many times and tried different time range and finally we found an appropriate one.

4) Since the flashing frequency was determined by ourselves, at first we could only detect the Morse code whose time unit was at least 1 second. After several experiments on the threshold of RGBA value, Light ON/OFF duration decision and distance testing, we can detect Morse code in higher frequency whose time unit was 0.5 second.

7.3 Current limitations

- The encoding part and the decoding part were separated as two applications.
- The emission frequency cannot be changed by user.
- The detection area was limited to a specific area.
- Detection of light was easily disturbed by those reflecting light.
- When emission frequency is too high, the app couldn't decode precisely or even couldn't decode.
- The program can't change the parameter of the environment light, like exposure value.
- The app couldn't decode the Morse code automatically, which means it can't decode the Morse code if it doesn't know the emission frequency.

Chapter 8: Future development

8.1 For the whole project:

- Combine the encoding part and the decoding part to one app.
- Try to implement the bi-directional communication in the standard way. For example, when the emission terminal send a signal which tells it's ready to send the signal, after receiving the signal, the reception terminal can reply a signal to tell it's ready to receive the signal as well. The emission terminal will not send message until the reception terminal is ready.

8.2 For the encoding part:

- Optimize the User Interface and make it user friendly.
- Increase the accuracy of sending the Morse code.
- Implement the function that a user can change the emission frequency before encoding.

8.3 For the decoding part:

- Increase the receiving accuracy and decoding accuracy.
- Try to receive and decode the Morse code whose frequency is higher than present.
- Realize longer distance Morse code emission and reception.
- Try to implement auto-detection of Morse code, which means the Morse code can be decoded without knowing its emission frequency.
- Implement the decoding part in a different way in which we can change the parameters of the environment, like exposure, brightness and so on.

Chapter 9: Acknowledgement

We would like to give our sincere thanks to Prof. Michael R. Lyu who met us every week to follow our progress in the project. Prof. Lyu gave many useful comments and suggestions on our project's modifications and improvements.

Besides, we would also like to thank VIEW Lab's researcher, Mr. Edward Yau, without his inspiring idea and practical instructions, our application wouldn't be developed so smoothly.

Chapter 10: Reference

[1] Wikipedia. "*Morse code*," Wikipedia.org. [Online]. Available:

http://en.wikipedia.org/wiki/Morse_code [Last Modified: 23 November 2013, 04:22].

[2] Google Play. "*Morse Code Trainer*," Google.com. [Online]. Available:

https://play.google.com/store/apps/details?id=hunt.morseDit [Accessed: November. 24, 2013]

[3] AppsZoom. "*Morse Code Translator*," AppsZoom.com. [Online]. Available:

http://cn.appszoom.com/android_applications/tools/morse-code-translator_gnfkp.html [Accessed:

November. 24, 2013]

[4] AppsZoom. "*Simple Morse Code Translator*," AppsZoom.com. [Online]. Available:

http://cn.appszoom.com/android_applications/tools/simple-morse-code-translator_grsdw.html

[Accessed: November. 24, 2013]

[5] AppsZoom. "*SMS2CW – Convert to Morse Code*," AppsZoom.com. [Online]. Available:

http://cn.appszoom.com/android_applications/communication/sms2cw-convert-to-morse-code_ceze.ht

ml [Accessed: November. 24, 2013]

[6] Wikipedia. "*Morse code*," Wikipedia.org. [Online]. Available:

http://en.wikipedia.org/wiki/Morse_code [Last Modified: 23 November 2013, 04:22].

[7] OpenCV. "*OpenCV ABOUT*," opencv.org. [Online]. Available:

http://opencv.org/about.html [Accessed: November. 24, 2013]

[8] OpenCV. "*OpenCV ABOUT*," opencv.org. [Online]. Available:

http://opencv.org/about.html [Accessed: November. 24, 2013]

[9] OpenCV. "*OpenCV PLATFORM*," opencv.org. [Online]. Available:

http://opencv.org/platforms.html [Accessed: November. 24, 2013]

[10] OpenCV. "*OpenCV PLATFORM*," opencv.org. [Online]. Available:

http://opencv.org/platforms.html [Accessed: November. 24, 2013]

[11] OpenCV. "*OpenC4Android Samples*," opencv.org. [Online]. Available:

http://opencv.org/platforms/android/opencv4android-samples.html [Accessed: November. 24, 2013]

[12] Android. "*Developers*," android.com. [Online]. Available:

http://developer.android.com/reference/android/hardware/Camera.html [Accessed: November. 24, 2013]