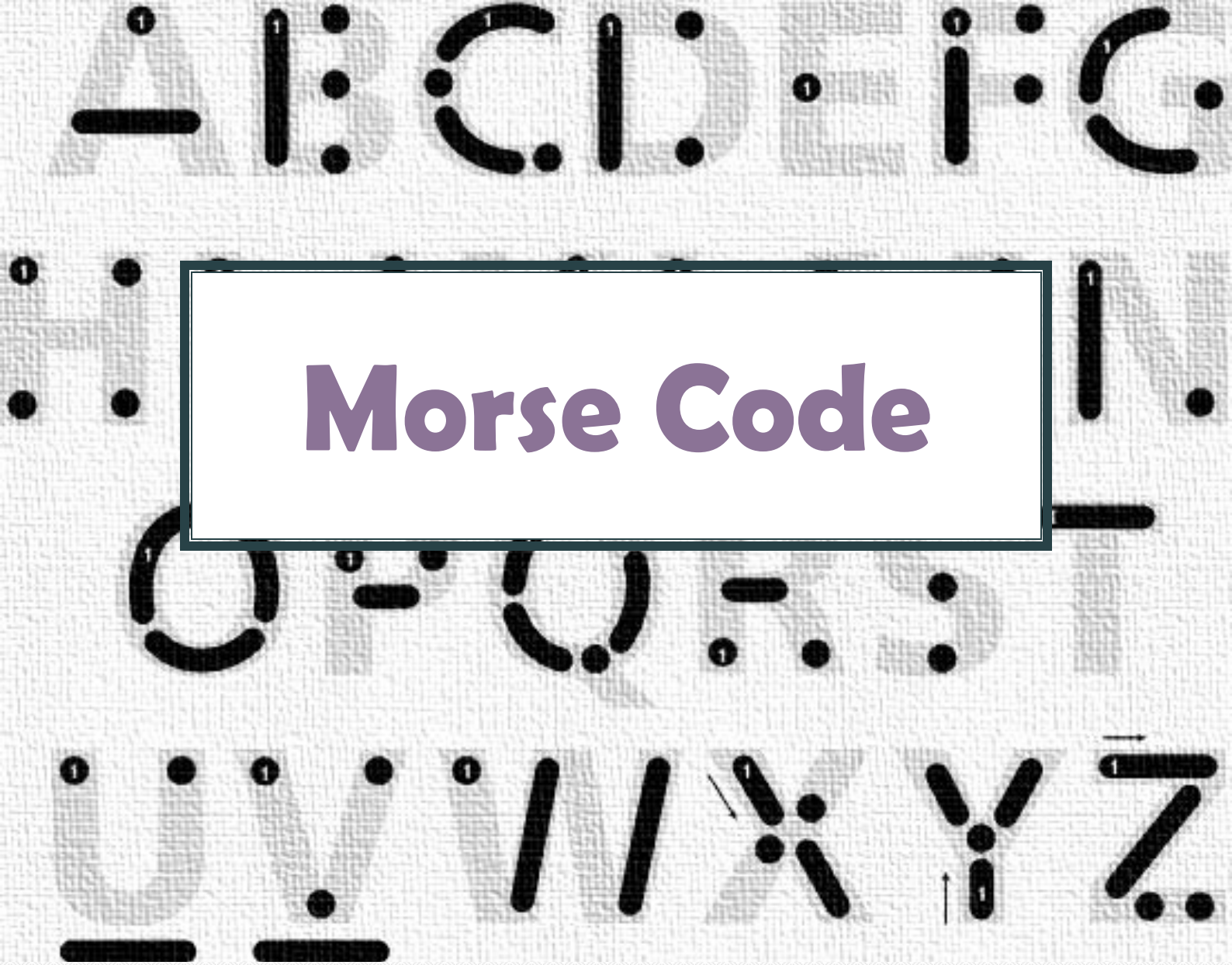


Final Year Project 2013 (2nd Term)

LYU 1305

Real-Time Morse Code Communication App

Supervisor: Prof. LYU Rung Tsong Michael

Prepared by 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 in this semester. 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. In this semester, we changed the implementation of the decoding part completely because of the different platform. We did some research on the usage of android camera and use it to get the real time images. 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. In the conclusion part, we show how we implement our application step by step. Finally, contributions to the project of each of us were described in detail.

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]:

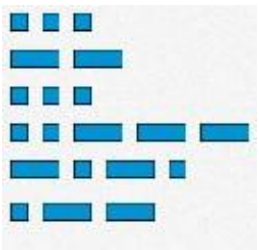
This is an app allowing users to send short flashlight text messages using the International Morse Code.

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 Summary in Fall 2013 (Term 1)

- Morse code message encoding and decoding part separately;
- Morse code was limited to English words base on alphabets;
- Base time of Morse code is fixed to 0.5 second;
- Detection area is a fixed rectangle.

1.4 Highlight in Spring 2014 (Term 2)

- A complete application including Morse code message sending and receiving part;
- A message box showing the sending and receiving messages was added;
- Message sending and receiving can be stopped manually;
- Base time of Morse code can be changed by users;
- Auto-search the light at the very beginning;
- Enable Chinese sending and receiving;
- Light tracking when the light is ON.

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:

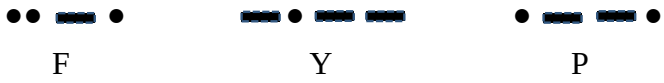
dash = 3 * dot = 1.5 seconds

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: Environment Setup

3.1 Overview

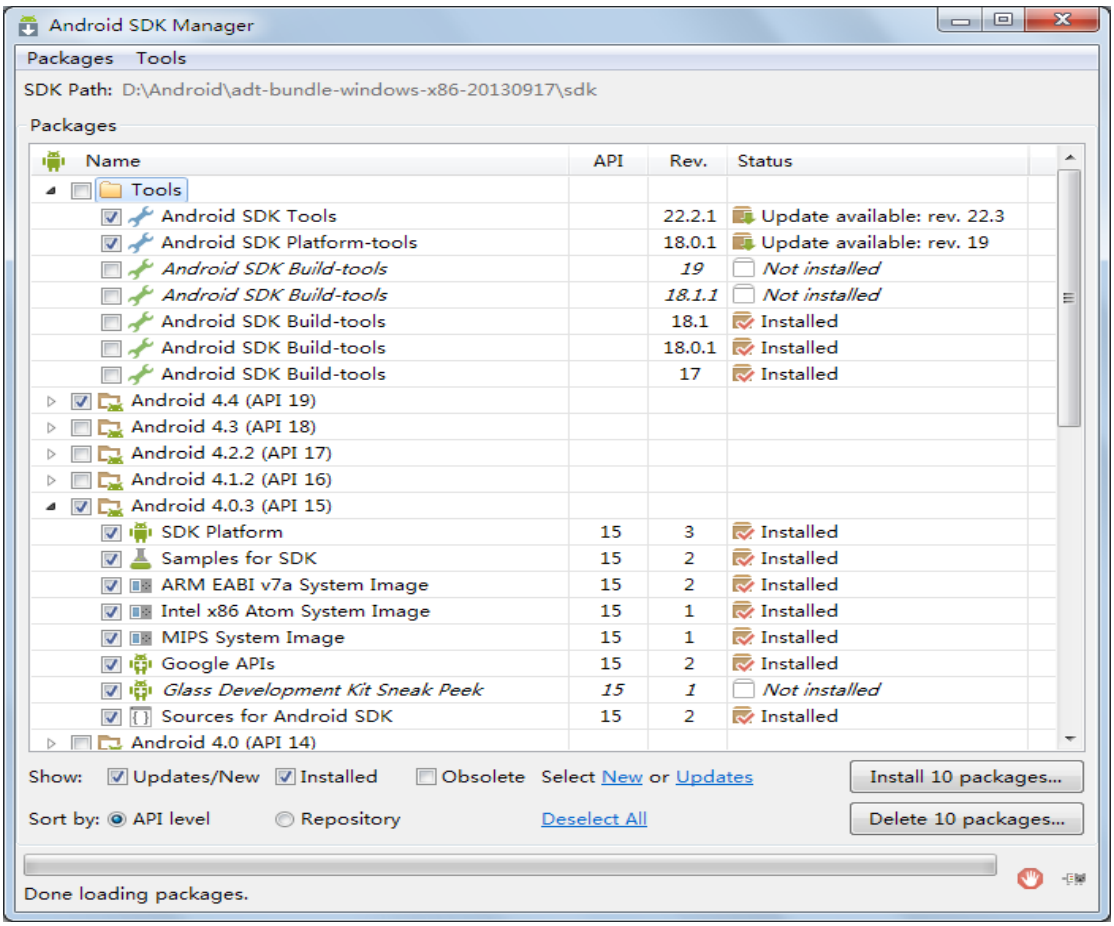
Our application is built in Android Operating System (OS) and developed in Eclipse combined with Android SDK in Windows. OpenCV is used as the main library in our project. The thing we need to do is to install Eclipse into our computer and set up all the parameters for the project development. However,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

3.2 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:



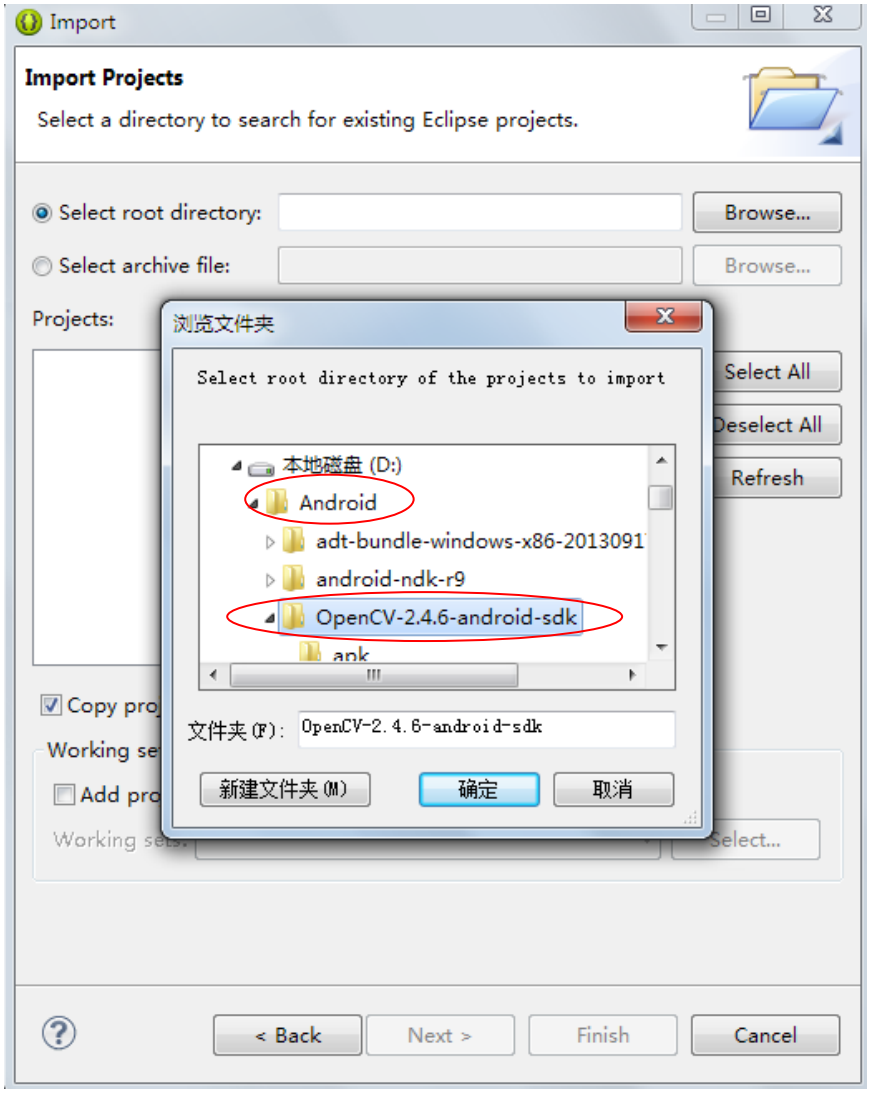
In the above interface, corresponding

- **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.

3.3 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:

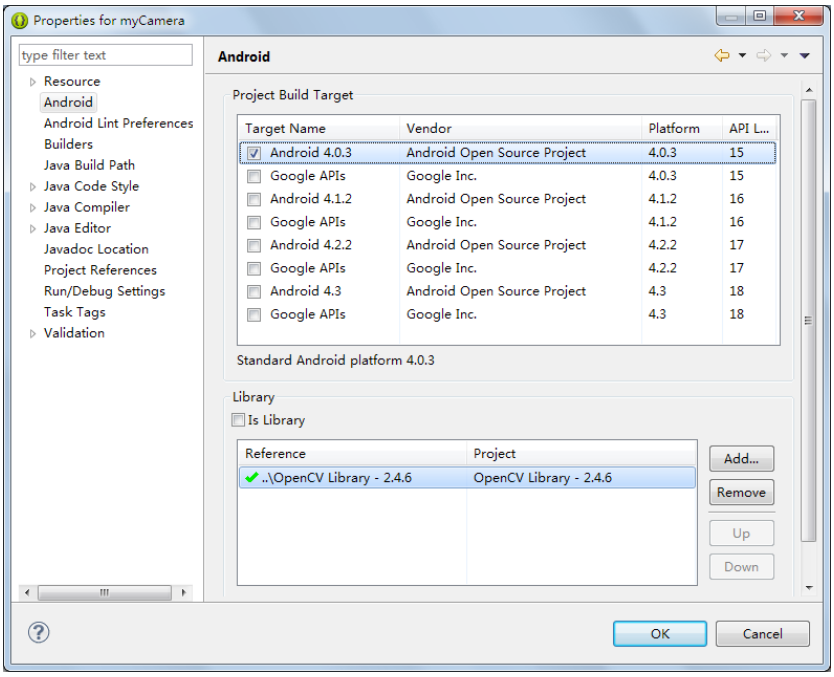


Only when the OpenCV folder was located in the same root directory can the later project run successfully. Otherwise, many other cumbersome parameters need to be changed achieve the same goal.

3.4 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

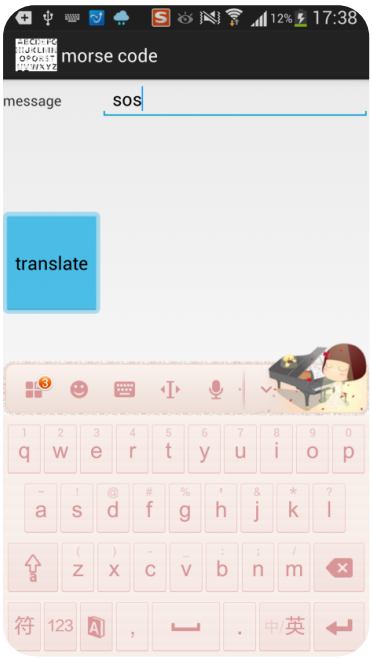


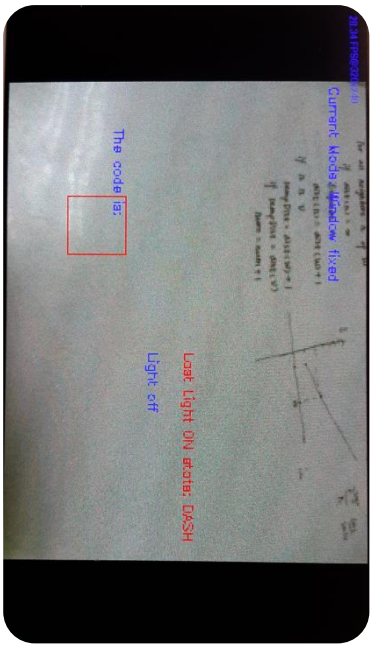
Just click on the lowest version of platform we need and add OpenCV library into the Library part, the sample project could run successfully.

Chapter 4: UI and Functionality Design

4.1 Initial Design

In last semester, we divided the app into encoding part and decoding part and therefore two user interfaces. They are just simple initial designs as followings:





In the encoding part, the UI was just a simple linear layout, including a text box for user to type message and a button to start transmission. The limitation was that you could not stop the transmission until it was totally finished.

In the decoding part, we used the OpenCV Library and the layout was just the camera preview and a focus rectangle. The limitation is that the light source had to appear in the area of the focus rectangle. Otherwise you might get incorrect message or even no message. It was not user-friendly because the two users had to hold their devices still.

4.2 Final Design

In the final design, we combine the encoding and decoding parts into one user interface (See Figure 4.2-1 and Figure. 4.2-2).

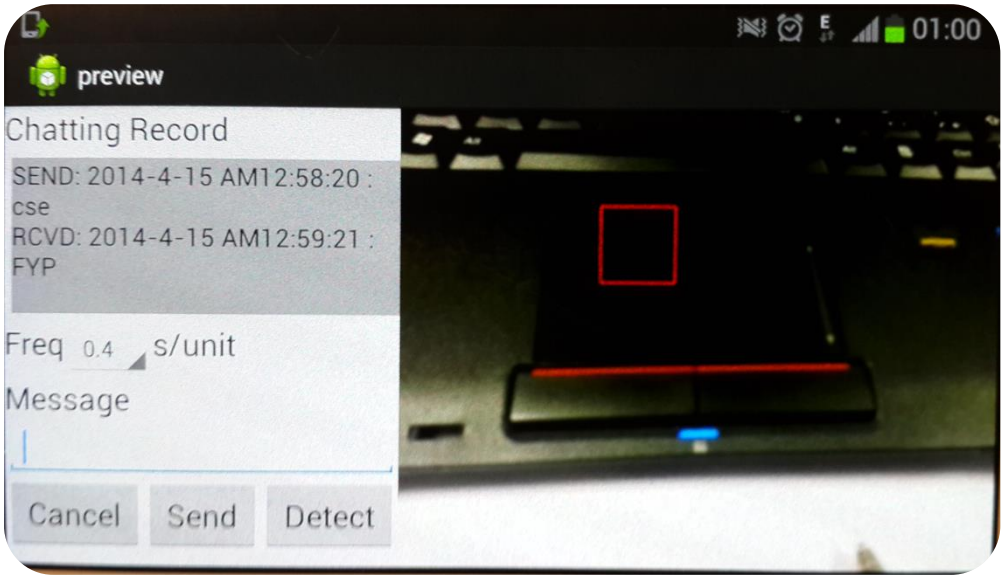


Figure 4.2-1

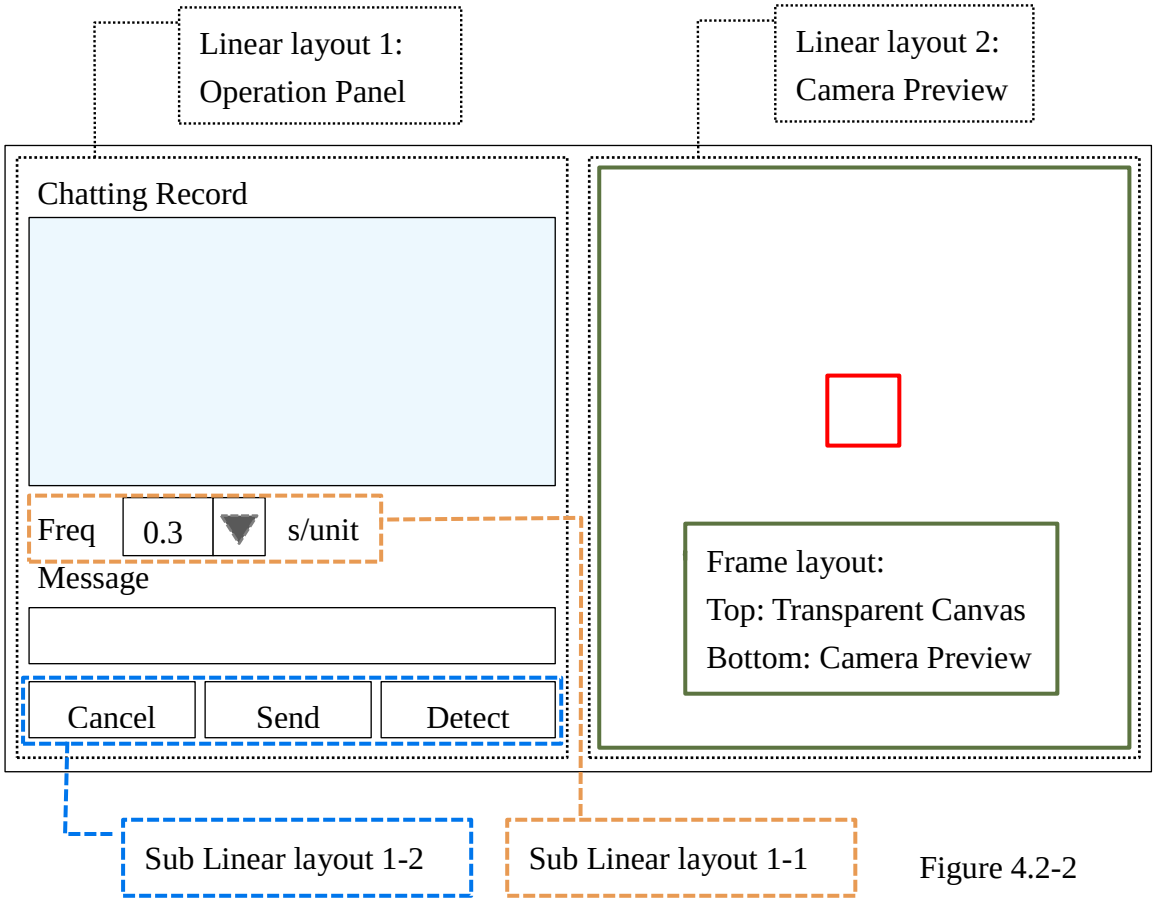


Figure 4.2-2

We divide the UI into two linear layouts. The left one is used to do operations and includes 2 sub linear layouts. The right one is used to display the camera preview and receive the message. It includes a frame layout which contains two layers. The bottom one is just the camera preview. The top one is a transparent canvas used to draw the focus rectangle.

The pseudo codes of the UI structure are as follows:

```
<LinearLayout: Overall UI>
    <LinearLayout 1: Operation Panel  orientation="vertical"  weight="1">
        <Text: “Chatting Record”>
        <EditText: Chatting Record Box>

        <Sub-LinearLayout 1-1: Frequency selection orientation="horizontal">
            <Text: “Freq”>
            <Spinner:Frequency selection>
            <Text: “s/unit”>
        </End of Sub-LinearLayout 1-1>

        <Text: “Message”>
        <EditText: Message Box>

        <Sub-LinearLayout 1-2: Buttons orientation="horizontal">
            <Button1: Cancel  weight="1">
            <Button2: Send  weight="1">
            <Button3: Detect  weight="1">
         </End of Sub-LinearLayout 1-2>
    </End of LinearLayout 1>

    <LinearLayout 2: Camera Preview  orientation="vertical"  weight="1">
        <FrameLayout
            <preview surface : the bottom surface/>
            <Canvas surface : the top surface  opacity=100%/>
        </End of FrameLayout>
    </End of LinearLayout 2>
</End of Overall UI LinearLayout>
```

4.3 Functionality

“Chatting Record” Edit Box

The sent messages and received messages will appear in this box.

The message formats are:

SENT: DATE TIME \n message

RCVD: DATE TIME \n message

“Freq” Selection List

User can choose a frequency (0.2/0.3/0.4/0.5 unit/s) to send message. The frequency means the time needed to send a dot.

“Message” Edit Box

User can type the message he/she wants to send in this box.

“Cancel” Button

When the message is sending and you want to stop it, click the “cancel” button.

“Send” Button

Click “Send” button to send message. If there is no content in “message” box, it will alert that “Cannot send empty message” and send nothing.

“Detect” Button

After click this button, the camera will start detecting light source and valid message. If a valid “starting signal” is received, the program will start decoding.

If the “Detect” button is not clicked, the decoding part will not work even if a “starting signal” is received.

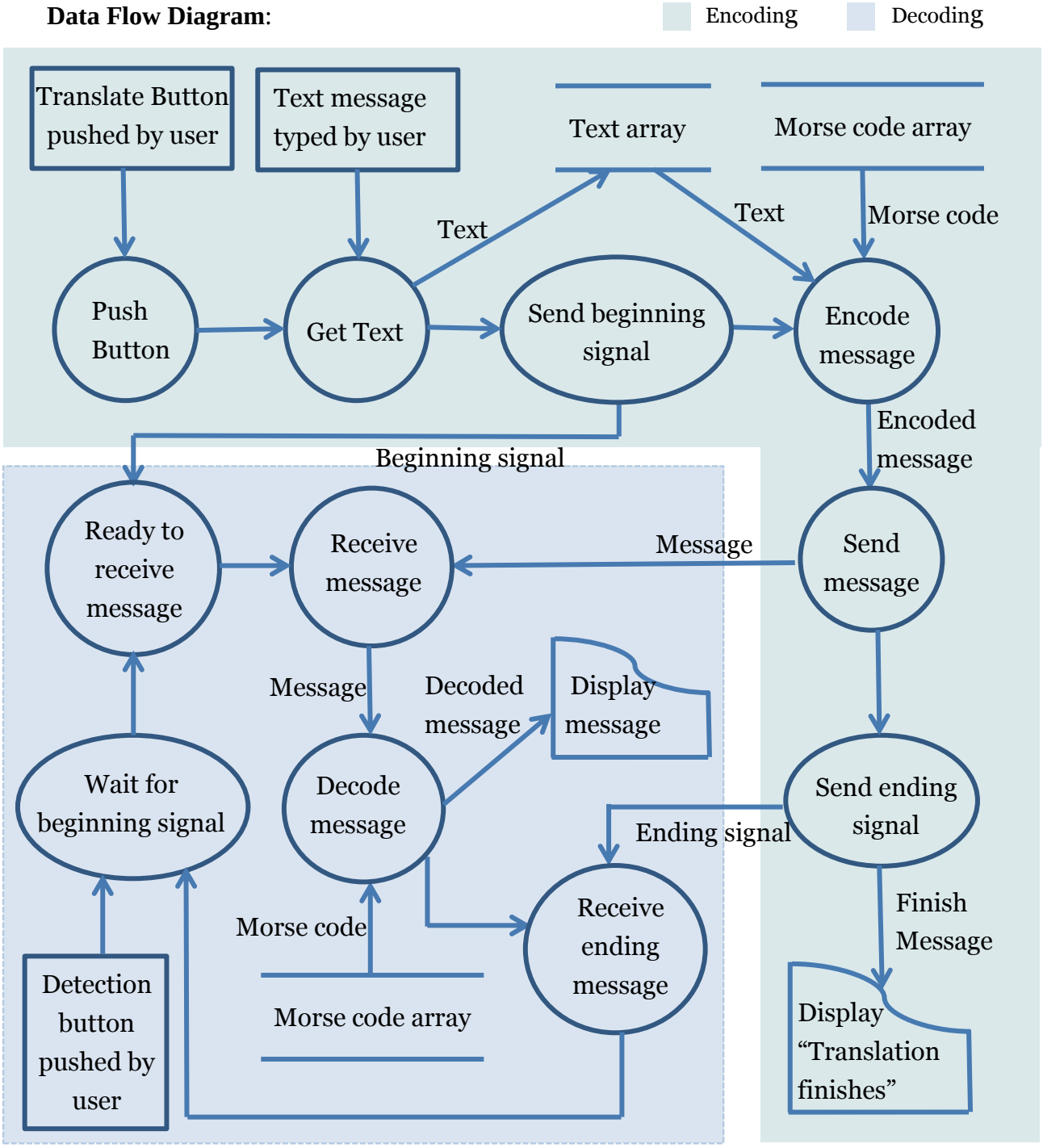
Camera Preview

If the “Detect” button is clicked and there is a light source, the focus rectangle will appear, focus on and track the light source. If it detects some valid “starting signal”, the decoding part will begin to work.

Chapter 5: Implementation Details

5.1 Overview

Data Flow Diagram:



5.2 Stage 1

5.2.1 Encoding

1. Flashlight control [7]

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;
}
```

2. 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.

3. 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.

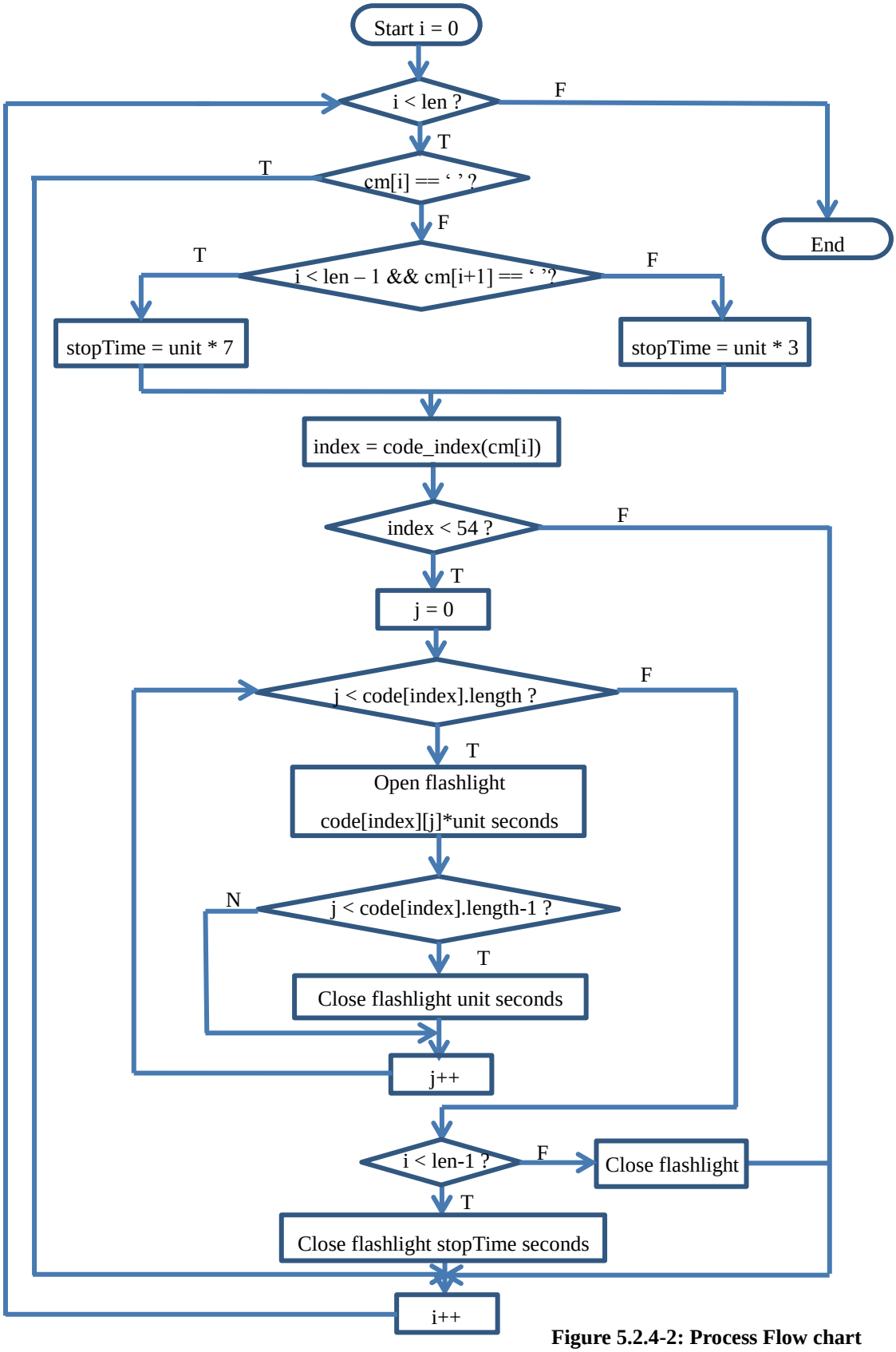


Figure 5.2.4-2: Process Flow chart

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.

5.2.2 Decoding: using OpenCV

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:

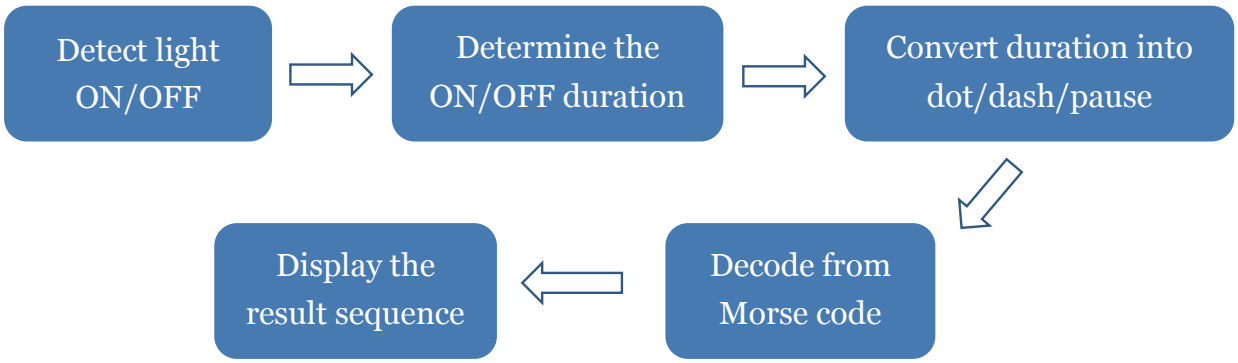


Figure 5.3.1-1

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.

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.

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.

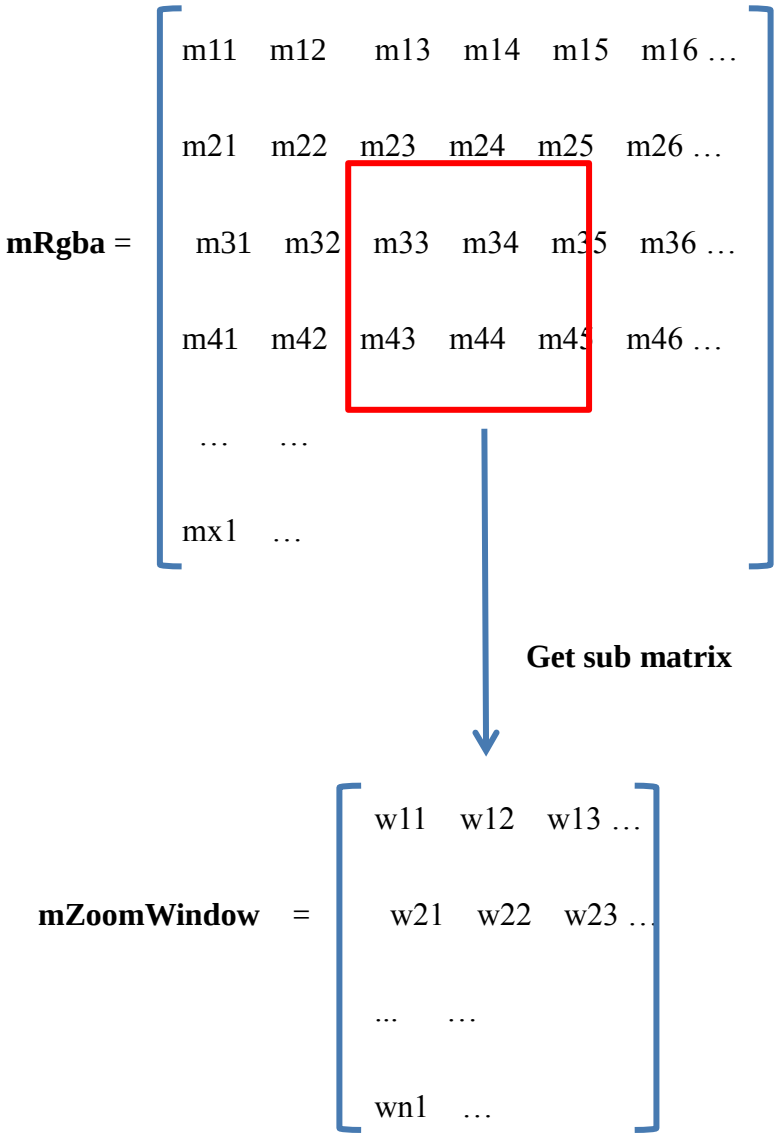
4. Process frame values

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:

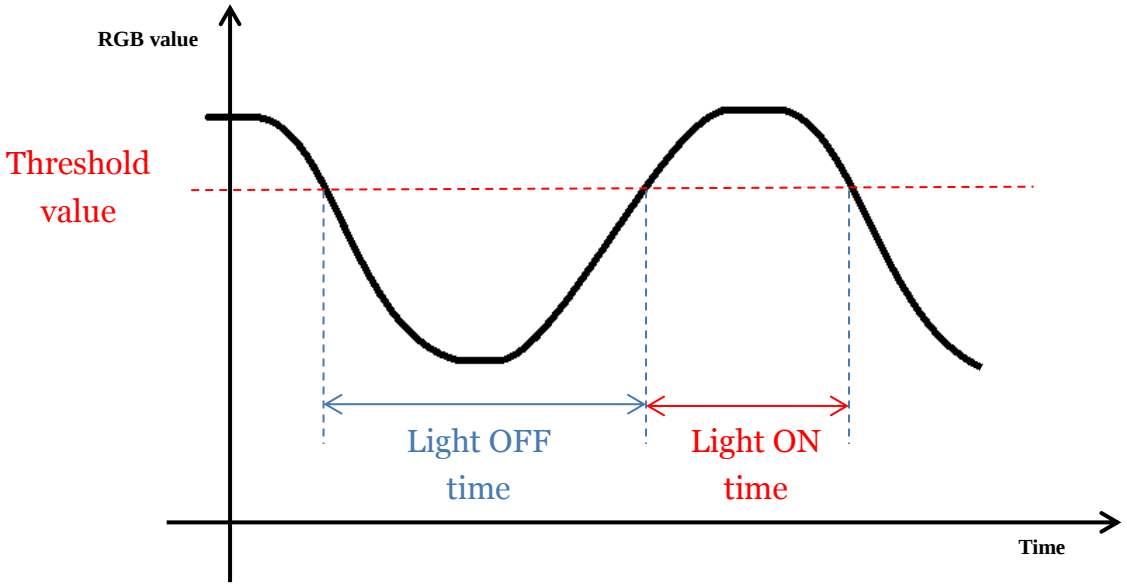
V(R) = 210 V(G) = 210 V(B) = 210

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%.

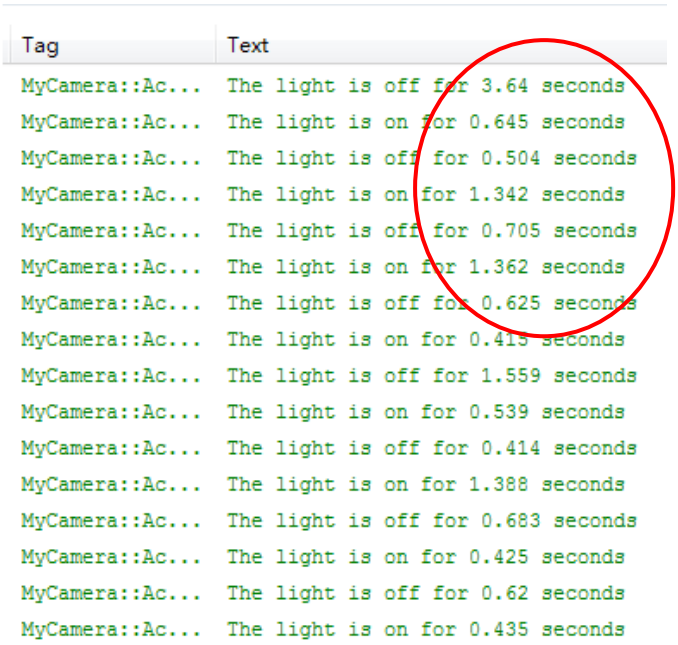
3) *Calculate the duration of light ON/OFF*

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:

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

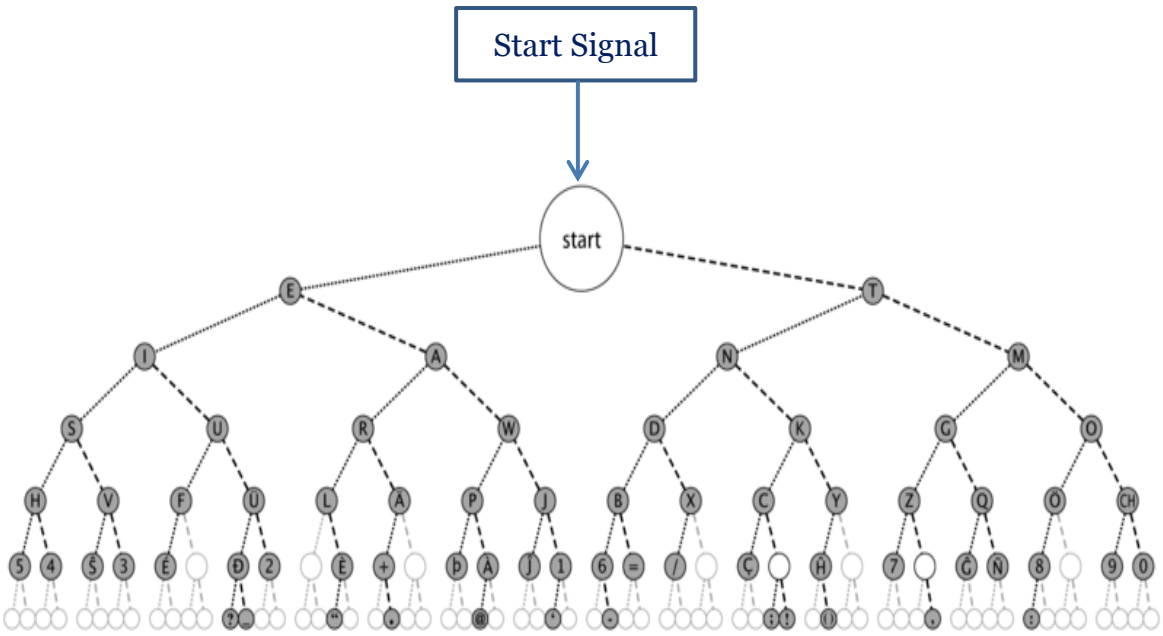
(dash = 3 * dot)

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

5. 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.

5.3 Stage 2

5.3.1 Camera Preview

Generally speaking, the steps to get the camera preview are as follows[8]:

1. Call the function *open()* in ***Camera*** to open the camera.
2. Call the function *getParameters()* in ***Camera*** to get parameters of camera, like resolution, exposure and so on.
3. Set *Camera.Parameters* and call *setParameters(Camera.Parameters)* to control the camera.
4. Call *startPreviewDisplay(SufaceHolder holder)* to select the surface to display the pictures.
5. Call *startPreview()* to get the camera preview.
6. Call *stopPreview()* to stop camera preview.

Actually, we implement the SurfaceHolder.Callback by overriding several functions:[9]

```
public void onCreate(Bundle savedInstanceState);
public void surfaceCreated(SurfaceHolder arg0);
public void surfaceChanged(SurfaceHolder arg0,int arg1,int arg2,int arg3);
public void surfaceDestroyed(SurfaceHolder arg0).
```

In function onCreate(), we use

`previewSurfaceView.getHolder()` to get the SurfaceHold of SurfaceView;

`addCallback(this)` to add a callback listener for the SurfaceHolder;

`setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS)` to set the type of PUSH buffer, declaring that the surface data is provided by other source (here is the camera), not the Canvas.

`setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE)` to fix the screen orientation.

In function surfaceCreated(), we use

`Camera.open()` to open camera when the preview surface is created;

`setPreviewSize(320, 240)` to set the resolution of camera;

`setPreviewFpsRange(30000, 30000)` to set the frame frequency;

`setExposureCompensation(params.getMinExposureCompensation())` to set exposure value.

The surfaceChanged() function is used to process the preview information when the data of surface view changes.

In function surfaceChanged(), we use

`setPreviewDisplay(arg0)` to set the surface view;

`setOneShotPreviewCallback(this)` to trigger the `onPreviewFrame(byte[], Camera)` function of Class `PreviewCallback`;

`startPreview()` to start the camera preview;

`stopPreview()` to stop the camera preview.

In function surfaceDestroyed(), we use

`setPreviewCallback(null)` to stop the preview callback, which is used to get the frame buffer data;

`stopPreview()` to stop camera preview;

`release()` to release the camera source.

5.3.2 Camera Buffer Frame

Last semester, OpenCV provides us an interface for processing each frames in real-time. The frame data it gives to us is already in RGB format. However, the frame it provides in every second is a little low. So we decide to use the camera that provided by android itself.

The API only gives us the data of first frame, while what we need are those real time frames continuously.

Trial1: Using thread to get frame-by-frame.

We define a class FRAME inherits from the class Thread:[10]

```
public class FRAME extends Thread {
      @Override
      public void run() {
          while(!Thread.currentThread().isInterrupted()) {
               try {
                  long lastTime = System.currentTimeMillis();

                   if(null != myCamera)
                   {
                       /*Processing the frame data*/

              myCamera.setOneShotPreviewCallback(MainActivity.this);
                   }
                   Thread.sleep(0);
               } catch (InterruptedException e) {
                   // TODO Auto-generated catch block
                   Thread.currentThread().interrupt();
               }
          }
      }
 }
```

Continuous frame can be accessed by calling:

At first, we called this function inside the thread:

setOneShotPreviewCallback(MainActivity.this)

It means that we build another thread to call the callback function iteratively.

Thread.sleep(0) was used to control the time between each thread. 0 was proved to be wrong finally. It called the preview callback function without stopping, while the camera can provide 30 frames in 1 second only. It caused that there are not enough frame for the thread to process. The image is stuck now and then. We tried to change 0 into other values that fit in the frame rate; however it was not that accurate. Therefore, we gave up this method.

Final solution:

Call the setOneShotPreviewCallback(MainActivity.this) at the end of the callback function onPreviewFrame(**byte**[] data, Camera camera). It will call itself every time it finishes the process of the frame data.

Frame provided by the camera: Byte[] data: in YUV420sp format.

YUV420sp[11]:

Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8
Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16
Y17	Y18	Y19	Y20	Y21	Y22	Y23	Y24
Y25	Y26	Y27	Y28	Y29	Y30	Y31	Y32
U1	V1	U2	V2	U3	V3	U4	V4
U5	V5	U6	V6	U7	V7	U8	V8

From the format table, we can see that every 4 Y data corresponds to 1 U data and 1 V data. Only when we deal with one 4x4 block Y data do we need to dead with one U data and one V data.

YUV data is not convenient for us to use. Therefore, after we get the Y, U, V data we

need, we turn them into R, G and B format data in this way[12]:

```
y1192 = 1192 * y;

r = (y1192 + 1634 * v);

g = (y1192 - 833 * v - 400 * u);

b = (y1192 + 2066 * u);
```

Then we stored these RGB values in a 3-dimensional matrix in integer format. There are 3 elements in each pixel, R, G and B:

```
mRgb[i][j][0] = (int)(r >> 10);

mRgb[i][j][1] = (int)(g >> 10);

mRgb[i][j][2] = (int)(b >> 10);
```

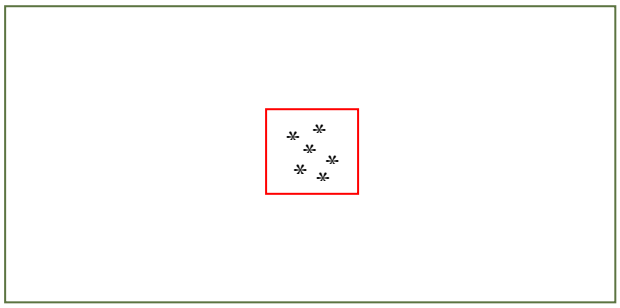
In this way, we can use these RGB values in the same way as last semester.

5.4 Stage 3

5.4.1 Light Detection

1. Algorithms for Light detection

1) Original version (asterisk represents pixel in Light ON state)



Last semester, our application's detection area is fixed to the red rectangle in the above picture. Only when light appears in this area can they be detected.

The way we determine that "Light ON" is:

Pixel Value > threshold value ==> Pixel in Light ON state

Percentage of "Light ON" pixel >= 10% of the pixels in red rectangle

That's how we determine the "Light ON" state last semester.

The problem is when the light attenuates, some pixels' value will affect the decision on "Light ON" state which made the application misunderstood that the Light is still ON. It caused that we can't decode those light sequence in high frequencies.

Therefore, we tried several ways to improve our light detection part.

2) 2nd Version

We don't want to fix the detection area anymore, so we need to search the light in the whole preview window.

Firstly, find the light center for the whole screen:

(Light center is similar to the mass center):

Suppose we have:

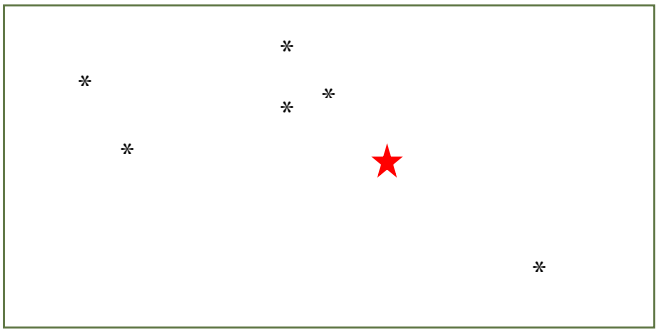
RGB(0,0)	RGB(0,1)	RGB(0,2)	RGB(0,3)
RGB(1,0)	RGB(1,1)	RGB(1,2)	RGB(1,3)
RGB(2,0)	RGB(2,1)	RGB(2,2)	RGB(2,3)

totalRGB = $\sum_{r=0}^{totalRow} \sum_{c=0}^{totalCol} RGB(r, c)$

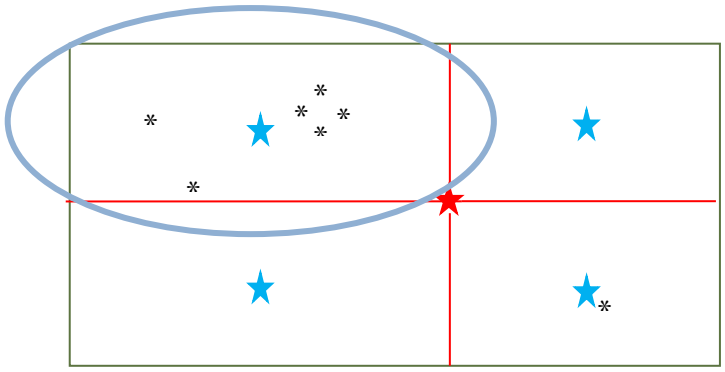
centerRow = $(\sum_{r=0}^{totalRow} \sum_{c=0}^{totalCol} r * RGB(r, c))$ / totalRGB

centerCol = $(\sum_{r=0}^{totalRow} \sum_{c=0}^{totalCol} c * RGB(r, c))$ / totalRGB

Using centerRow and centerCol to locate the light center: (Star represents the Light Center)



Cut the screen into several parts based on the light center:



Find the light center for each grid and let the grid with the largest light center values to be next search area.

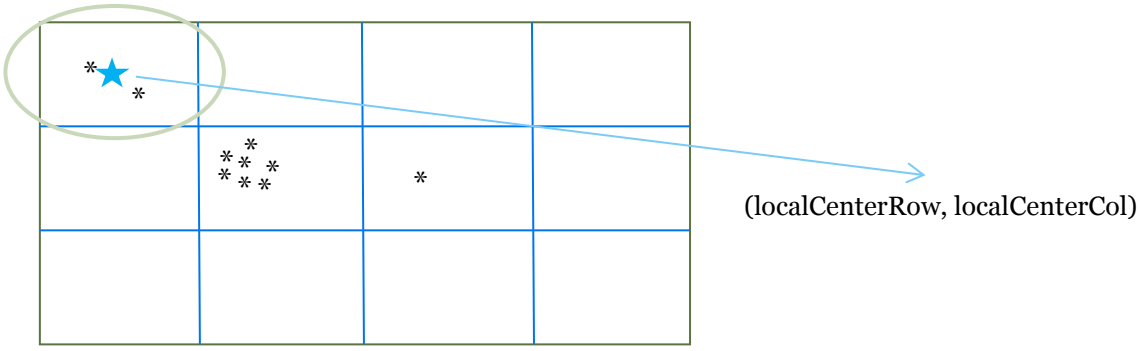
Repeat the above procedure until we find the cluster of the Light ON area.

Trial failed:

Searching in the first step already took too much time, let alone cut the screen and repeats the procedure in the selected area. Frame showed by the camera totally stuck. Therefore, we gave up this method.

3) 3rd Version: By comparing grid light center's RGB value

Divide the screen into 3x4 grids at the beginning.



Set maxLightCenterRGB = 0; maxRow = 0; maxCol = 0;

Start from the first grid[0, 0].

While there is still grid not processed

Find Light center's RGB value tempLightCenterRGB of the grid and its coordinate tempRow and tempCol, compare the tempLightCenterRGB with maxLightCenterRGB.

If tempLightCenterRGB > maxLightCenterRGB:

maxRow = tempRow;

maxCol = tempCol;

maxLightCenterRGB = tempLightCenterRGB

At last, we got the grid with the maximum Light Center RGB. Then we can follow on this grid later.

Comparing to 2nd Version, we can see that we only need to traverse all the pixels in one frame one time, which saved us a lot of time.

However, taking all the pixels into consideration made the light center offset from the light source too much. Thus we add another restriction, that is only taking pixels whose (R + G + B) / 3.0 is bigger than 210 into consideration. The result is better than before.

4) **4th Version:** Find local maximum RGB pixels and count numbers

During the test of 3rd Version, we came across some problem.

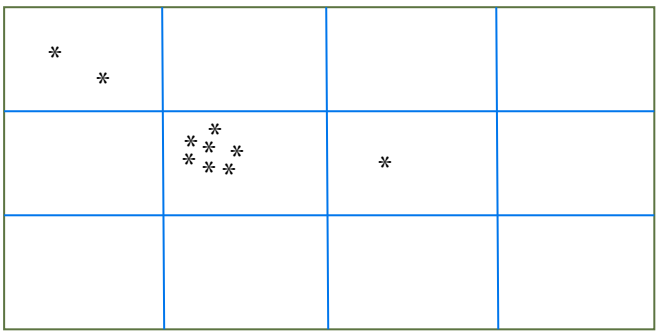
If we have two grids like this:

We can see that in the second grid, the Light Center still offset from the light source which means the RGB value of the Light Center may not approximate to the value of the light source. However in the first grid, the Light Center is inside the white paper because of the uniform distribution of the pixels' color.

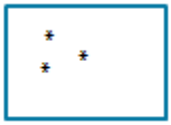
In this way, the RGB value of the Light Center in first grid will be larger than the RGB value of the Light Center in second grid. It's obviously not we want, because the second grid is the actual grid that contain the light source. Our algorithm misunderstood the "white paper" in the middle to be the light source.

Therefore, the RGB value of "Light Center" doesn't work all the time, and then we have our 4th algorithm.

Still divide the screen into 3x4 grids:



Intra-grid:



Define local maximum RGB value of pixel: localMaxRGB = 0

local number of pixels with maximum RGB: localMaxNum = 0.

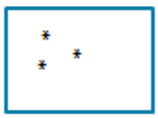
Start from the first pixel in this grid.

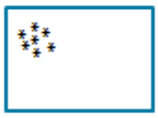
If pixelRGB > localMaxRGB, it means that the original local maximum RGB value is not the maximum anymore. We need to assign pixelRGB to localMaxRGB and reassign value 1 to localMaxNum.

If pixelRGB = localMaxRGB, it means that we have another pixel with maximum local RGB value, then let localMaxNum = localMaxNum + 1.

That's how we get the local maximum RGB value of pixel and number of them.

Inter-grid:





Suppose we have localMaxRGB1 and localMaxNum1 for the first grid,

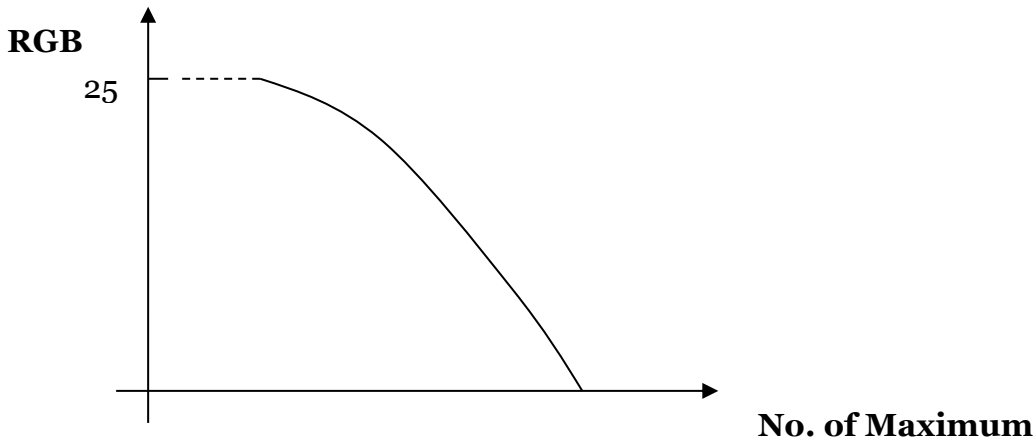
localMaxRGB2 and localMaxNum2 for the second grid.

How to determine which one is the one with light source?

Comparing localMaxRGB1 to localMaxRGB2 and comparing localMaxNum1 to localMaxNum2.

Previous idea:

At first, we thought the relationship between maximum RGB value and number of pixels with maximum RGB value is: (Under the condition that the light exitsts)



We thought that even if the RGB value of the pixel is not 255, if it approximates 255 and the number of these pixels is very large, we can consider it to be Light ON. However, after testing on this condition, we found that only if there is light inside a grid, some pixels' values must be 255.

Present solution:

Select the larger one from localMaxRGB1 and localMaxRGB2 and keep checking localMaxRGB value for other grids. After we get the largest localMaxRGB, assign it to globalMaxRGB, and assign its localMaxNum to globalMaxNum.

This is how we implement the comparison part:

```
for (int i = pRow[m]; i <= (pRow[m] + recSide - 1); i++)
{
            for (int j = pCol[m1]; j <= (pCol[m1] + recSide - 1); j++)
            {
                  data1 = mRgb[i][j];
                  tmpRgb = data1[0] + data1[1] + data1[2];

                  if (tmpRgb > tmpMax)
                  {
                       sumI = j * tmpRgb; //Initial sum of max RGB pixel
                       sumJ = j * tmpRgb;
                       sumRgb = tmpRgb;

                       tmpMax = tmpRgb; //Local Maximum RGB
                       numMax = 1; //number of maximum pixels
                  }
                  else if (tmpRgb == tmpMax)
                  {
                       sumI = sumI + j * tmpRgb;
                       sumJ = sumJ + j * tmpRgb;
                       sumRgb = sumRgb + tmpRgb;

                       numMax++; //number of maximum pixels
                  }
            }
}
```

If globalMaxRGB = 255 and globalMaxNum > = 10

We can determine that there is light in this area.

After we know that there is light, how to draw the detection window?

Use the method in 4th Version:

Get the light center (centerRow, centerCol) of the grid with the globalMaxRGB and globalMaxNum.

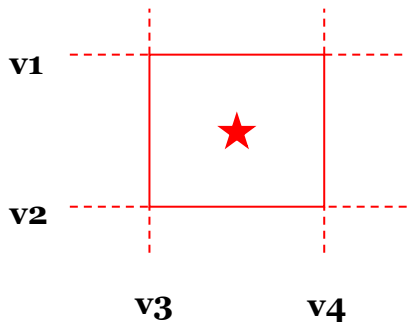
Suppose the window size is RecSize, draw the detection window around the center: (Limit v1, v2, v3, v4 inside the screen)

v1 = centerRow – 0.5 * RecSize

v2 = centerRow + 0.5 * RecSize

v3 = centerCol – 0.5 * RecSize

v4 = centerCol + 0.5 * RecSize

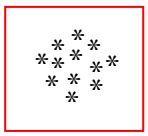


Therefore, we got the initial detection window.

Due to the precision of the locating, we change the division into 6x8. Then the locating area can be much smaller.

5.4.2 Auto Tracking

After locating light at the beginning, we have a fixed window for it:



Fixed window means that, if the camera shook accidently, then the light may offset from the fixed window. Decoding can't go on anymore and tracking is needed.

Tracking procedure:

Calculating the light center of the initial detection window by:

centerRow = $\sum_{r=v1}^{v2} \sum_{c=v3}^{v4} r * RGB(r, c)$

centerCol = $\sum_{r=v1}^{v2} \sum_{c=v3}^{v4} c * RGB(r, c)$

Updating v1, v2, v3, and v4 based on the centerRow and centerCol we just got.

Repeating the above procedure only if light appears in this window.

Problems:

Tracking should have meant that the application can capture the light anytime it appears. We tried to keep searching the whole screen unless we detect the light. However, every time it searches, the whole screen will stick and light data may lose. Therefore we restrict the condition of tracking as follows:

- The application only will locate the light at the very beginning.
- Tracking works only when the light is inside the detection window.
- Detection window is fixed when the light is OFF.
- Light should appear in the detection window if we have the detection window.

5.5Stage 4

5.5.1 Combine Encoding and Decoding Parts

Up to this stage, we have used the Android Class “Camera” to display the camera preview, get the frame buffer from the preview and also convert the frame data from YUV format to RGB format successfully. With the RGB data from the frames we can do the calculation and decoding. And we also have implemented the User Interface combined with the operation panel and the camera preview. It is time to combine the encoding part and decoding part.

We use multiple threads to achieve the bi-directional communication.

Main thread:

Get the User Interface and Camera Preview. When “Detect” button is pressed, it will get the frame buffer, search and track the light source and decode the light sequences.

Child thread “ON”:

When “Send” button is pressed, this thread is created. This thread is used to open the flashlight, convert the message to Morse Code and send the light sequences.

In the thread "ON", we need to update the User Interface in following situations:

1. When the "Cancel" button is pressed, the User Interface should display the string "Transmission canceled" to the screen.
2. When the transmission is finished, the User Interface should put the input message to the "Chatting Record" box and display the string "Transmission finished" to the screen.

The above operations are actually to update the User Interface. But in Android programming, only the original thread that creates the UI view (here is the main thread) can update the UI. Other thread cannot directly update the UI. So we use the function runOnUiThread(Runnable) to create the UI-update codes in a Runnable object. When the UI needs to be update, pass this Runnable object to the Activity.runOnUiThread(Runnable). Then the Runnable object can be called in the UI thread.[13]

5.5.2 Unicode Encoding and Decoding

The International Morse Code can only represent 54 characters, so it cannot communicate with Chinese. It is not convenient for Chinese users. Therefore, we tried to use Unicode to represent Chinese.

Unicode Representation[14]:

Unicode defines a space of 1,114,112 code points in the range from 0hex to 10FFFFhex. Normally a Unicode code point is referred to by writing "U+" followed by its hexadecimal number. For code points in the Basic Multilingual Plane (BMP), four digits are used (e.g. U+0058 for the character LATIN CAPITAL LETTER X); for code points outside the BMP, five or six digits are used, as required (e.g. U+E0001 for the character LANGUAGE TAG and U+10FFFD for the character PRIVATE USE CHARACTER-10FFFD).

Figure 5.4.2-1

Chinese Unicode is from U+4E00 to U+9FA5 (19968 – 17194).

Some examples of Chinese Unicode are as Figure 5.4.2-2[15].

Actually, Java uses “\uhhhh” to represent a Unicode code point. But we cannot use it because no code for ‘\’ in Morse code set. Therefore, we still applied the

normal format “U+hhhh” to represent Unicode code points.

Encoding of Chinese Characters:

In the encoding part, we have a function convert Chinese characters to Unicode:

```
public String chineseToUnicode(String str).
```

This function works like this:

1. Checks every character chr1 in the input message string.
2. If (chr1>=19968&&chr1<=171941) is true, which means the character chr1 is a Chinese word, then append string "U+"+Integer.*toHexString*(chr1) to the result string. Else if the character chr1 is not a Chinese word, then just append it to the result string.
3. Return the result string.

Decoding of Chinese Characters:

In the encoding part, we have another function convert Unicode back to Chinese characters:

```
public String UnicodeToChinese(String str).
```

This function works like this:

1. Use the Class “Pattern” to define the pattern of Unicode “U+hhhh”:

```
Pattern pattern = Pattern.compile("(\\U+(\\p{XDigit}{4}))");
```

2. Use the Class “Matcher” to search the pattern “U+hhhh” in the received message string str:

```
Matcher matcher = pattern.matcher(str);
```

3. Whenever find the matched pattern, omit the sub-string “U+” and convert the 4 hexadecimal numbers back to Chinese character.

```
while (matcher.find()) {
        ch = (char) Integer.parseInt(matcher.group(2), 16);
        str = str.replace(matcher.group(1), ch + "");
}
```

4. Return the string str.

U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4e00	一	丁	丂	七	丄	丅	丆	万	丈	三	上	下	丌	不	与	丏
4e10	丐	丑	丒	专	且	丕	世	丗	丘	丙	业	丛	东	丝	丞	丟
4e20	丠	両	丢	丣	两	严	並	丧	丨	丩	个	丫	丬	中	丮	丯
4e30	丰	丱	串	丳	临	丵	丶	丷	丸	丹	为	主	丼	丽	举	丿
4e40	乀	乁	乂	乃	乄	久	乆	乇	么	义	乊	之	乌	乍	乎	乏
4e50	乐	乑	乒	乓	乔	乕	乖	乗	乘	乙	乚	乛	乜	九	乞	也
4e60	习	乡	乢	乣	乤	乥	书	乧	乨	乩	乪	乫	乬	乭	乮	乯
4e70	买	乱	乲	乳	乴	乵	乶	乷	乸	乹	乺	乻	乼	乽	乾	乿
4e80	亀	亁	亂	亃	亄	亅	了	亇	予	争	事	二	亍	于	亏	
4e90	亏	云	互	亓	五	井	亖	亗	亘	亙	亚	些	亜	亝	亞	亟
4ea0	亠	亡	亢	亣	交	亥	亦	产	亨	亩	亪	享	京	亭	亮	亯
4eb0	亰	亱	亲	亳	亴	亵	亶	亷	亸	亹	人	亻	亼	亽	亾	亿
4ec0	什	仁	仂	仃	仄	仅	仆	仇	仈	仉	今	介	仌	仍	从	仏
4ed0	仐	仑	仒	仓	仔	仕	他	仗	付	仙	仚	仛	仜	仝	仞	仟
4ee0	仠	仡	仢	代	令	以	仦	仧	仨	仩	仪	仫	们	仭	仮	仯
4ef0	仰	仱	仲	仳	仴	仵	件	价	仸	仹	仺	任	仼	份	仾	仿
4f00	伀	企	伂	伃	伄	伅	伆	伇	伈	伉	伊	伋	伌	伍	伎	伏

Figure 5.4.2-2[15]

5.6 Stage 5: Auto Detection

We set a start signal with different time length for auto detecting. The length of the start signal is based on the frequency of the Morse code we want to send. After we detected the start signal, we can calculate the base time for the Morse code according to our rule. Then, the subsequence Morse code pattern can be decoded.

Chapter 6: Experiments and Testing

6.1 Window size testing

At first, we set the detection window size to be 80x80 pixels.

Locating time (ms)	**276**	**357**
Tracking time (ms)	20 ~ 60	20 ~ 60

Then we tried the size to be 40x40 pixels:

Locating time (ms)	**276**	**298**
Tracking time (ms)	5 ~ 15	5 ~ 20

We can see that the locating time of these two are almost same. However, because of the window size of the second one is much smaller, the tracking time of it is less than the first one. Smaller window size also means more precise locating.

Therefore, we adopt the 40x40 pixels' window.

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.

7.1.3 Combination

1) Research on the tracking algorithm to realize that the focus rectangle can track the light source position.
2) Use the Android Camera Class instead of the library OpenCV to display the camera preview.
3) Design the User Interface that combines the operation panel and the camera preview display.
4) Research on how to get the frame buffer from the camera preview and how to convert YUV to RGB.
5) Combine the encoding part and the decoding part.
6) Implement the Chinese communication by utilizing Unicode.
7) Implement the transmission frequency auto detection.

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.

7.2.3 Combination

1) Locating

At first, we want to locate the light after it is turned ON. The way is to recursively search the whole screen and the grid we want. However it costs too much time and the preview image is stuck. We changed the way by dividing the whole screen and searching each grid, process the data at the same time.

2) Tracking

We tried to track the light even when it's OFF during the decoding process. However, keep locating when the light is OFF costs too much time, when the light is suddenly ON, Light ON time data may lose. Therefore, we do the tracking only when the light is ON. When the light is OFF during the process, it must appear in the detection window to be decoded.

3) Detection window drawing

A canvas was used to draw the detection window. It's on the top surface of the frame layout. When we got the detection window, we found that it offsets from our expected position totally. The detection window always appeared in the upper left corner of our preview image. Finally we found that, our preview size is 320x240,

while the canvas's original size is 1280x960, they didn't match with each other. That's why the detection didn't appear in the area we want. We resized the canvas and the problem was solved.

4) Frame Buffer

OpenCV gives us the real time frame buffer last semester, while the android only gives the frame buffer of the first frame. We need to call function by ourselves to get subsequent frames. At the beginning, we used thread to solve this problem, while the interval time between threads is hard to control. Then we decide to call the callback function immediately after we finished the processing of last frame.

5) Update UI in child thread

At first, we use the functions `Toast.makeText(MainActivity.this,String,0).show()` and `EditText.append(String)` in the child thread directly. But the program would break down. Finally we found that in Android programming only the original thread that creates the UI view (here is the main thread) can update the UI. Other thread cannot directly update the UI. The function runOnUiThread(Runnable) is one way to communicate with the UI thread (main thread) and then update the UI.

public final void runOnUiThread (Runnable action):[16]

If the current thread is the UI thread, then the action is executed immediately. If the current thread is not the UI thread, the action is posted to the event queue of the UI thread.

Parameters action: the action to run on the UI thread.

6) Thread declaration

When the “Send” button is clicked, the encoding thread “ON” should be on. At first, we declare the “ON” thread variable outside the click listener of “Send” button “`send.setOnClickListener(new OnClickListener(){});`”. But the problem is that after the “Send” button has been clicked once, the “ON” thread will not start again. And we found the solution is to declare the thread inside the onClickListener(){} as a local variable. Because every thread has a thread ID once it is created. If it is declared as an global variable, its ID will not be changed. And once it is finished, it will not be start again. But if it is declared as a local variable, every time when it is declared, it will be assigned another ID and then can be start the encoding program.

7.3 Limitations

- Detection of light was easily disturbed by the environmental light.
- When emission frequency is too high, the app couldn't decode precisely or even couldn't decode.

Chapter 8: Contribution

We started thinking and discussing about the project from September, 2013. After we got a rough idea about what we need to do on this project, I started working on it.

8.1 Fall 2013

Our project is built in android operating system, while Android programming was totally new to me. I started by setting up the project environment in Eclipse and learning the Android programming. Simple programs in guiding books and website helped me get to know about it. Then I built my first project in Android.

I was responsible for the decoding part and my partner is responsible for the encoding part of our project in first semester. Originally, it was also an OpenCV depending project. OpenCV was another new thing to me.

- I downloaded all the tutorials and sample program from the OpenCV's website. By searching the meaning of all the functions and type of variables, I almost knew how to adapt them to our project.
- After knowing how to get frame buffer from the function provided by OpenCV, I was able to working on the algorithm for decoding.

- By analyzing frame data and coordinating with other functions in my program, my program changed from successfully detecting the light to getting the duration of light ON time.
- Finally, a light patter Morse code decoder was finished at the end of the first semester.

My partner and I conducted many experiments on the encoder and decoder to make sure that our parts worked normally. During the experiments, we also tried to find and fix bugs, improving the performance of our application respectively.

8.2 Spring 2014

After getting the initial design of our application last semester, we tried to make more improvements in this semester.

I was responsible for improving the performance of light detecting algorithm and adapting the decoder part to the android camera; my partner was responsible for combining encoder and decoder into one application and multi-thread programming.

- Light locating and light tracking were implemented in this semester. It means the light doesn't need to fix to a specific detecting window at the beginning and the light source can move when it's sending Morse code.
- Different algorithms were tried to make the locating much more accurate. Light Center of the area was calculated in the same way like calculating the mass center of object.
- During the combining the two parts, OpenCV was dropped totally. We changed to use the frame buffer provided by Android camera and turned the format of the pixel values from YUV420sp to RGB for convenient use later.
- Drawing the detection window is also different from drawing it based on OpenCV. A canvas was obtained to do the drawing thing on the top surface of the frame layout.

After all the works were finished by my partner and me, experiments were conducted on them to fix bugs and improve performance.

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

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

[8] Baidu. "*Baidu Wenku*," baidu.com. [Online]. Available:

http://wenku.baidu.com/view/733d2268b84ae45c3b358c37.html [Accessed: April. 14, 2014]

[9] Baidu. "*Baidu Wenku*," baidu.com. [Online]. Available:

http://wenku.baidu.com/link?url=pMWqPAjzaT7Ak9PK7NP3Fz28tf6Y7eRsd9qiv3qerCxaIvOkxQzs_LKLm0hzgwcNZ1PmTTO-NJ_4kQDL9vkJJjz5Nwh4OuBt2ISgsG2Su3y [Accessed: April. 14, 2014]

[10] CSDN. "*C Blog*," csdn.net. [Online] Available:
http://blog.csdn.net/yanzi1225627/article/details/8605061 [Accessed: April. 14, 2014]

[11] CSDN. "*C Blog*," csdn.net. [Online]. Available:

http://blog.csdn.net/jefry_xdz/article/details/7931018 [Accessed: April. 14, 2014]

[12] CSDN. "*C Blog,*" csdn.net. [Online]. Available:

http://chenweihuacwh.iteye.com/blog/571223 [Accessed: April. 14, 2014]

[13] CSDN. "*C Blog,*" csdn.net. [Online]. Available:

http://blog.csdn.net/luckyjda/article/details/8601517 [Accessed: April. 14, 2014]

[14] Wikipedia. "*Unicode,*" Wikipedia.org. [Online]. Available:

http://en.wikipedia.org/wiki/Unicode [Last Modified: 24 March 2014, 19:40].

[15] Baidu. "*Baidu Baike,*" baidu.com. [Online]. Available:

http://baike.baidu.com/view/40801.htm#12 [Last Modified: 23 April 2014].

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

http://developer.android.com/reference/android/app/Activity.html#runOnUiThread(java.lang.Runnabl

e) [Accessed: April. 14, 2014]