Final Year Project

Interim Report

QR-Motion: QR code generator and decoder in continuous mode on Android

Haoxuan Wang
Supervised by Dr Ioannis Patras

3-12-2012

Table of Content

1 Introduction

QR code is a kind of 2 dimensional barcode which has a wide range of uses. Due to its high efficiency and excellent performance in error correction, it is now one of the most popular ways to store text or other information in barcodes.

There are existing QR decoding applications on Android like AT&T Scanner and Zebra Crossing barcode scanner. All these applications on mobile devices provide QR Scanning and online or offline translation of a single static QR code. Once the smartphone detects a QR code, it will stop detecting and decode the information in the received code.

1.1 Literature Review and Related Works

QR code is firstly invited by Denso Wave a Japanese company who aim to provide “a way of Improving Inventory control and tracking in in manufacturing units” [1]. In order to promote this new technology

which latter became an ISO standard [2], Denso Wave released the right of using QR code to the public for free in 1994. Now QR code is not only used in industrial field, but it also can be seen in peoples' daily life. Mobile devices with a camera and enough computing power like a smart phone can capture and decode a QR code and show the information which might be an URL or a piece of text.

1.1.1 Application of QR Code

In Logistics and Retailing

Since barcode is invented, it has been printed on commodities, envelopes, and books for quite a long time. As the successor of 1 dimensional barcode, QR code is now also widely used in the area mentioned above. Due to its larger capacity, QR codes can do more than the traditional 1-D barcode in these areas. For example, in the library of University of Bath, QR codes are put onto some bookshelves to help the users find the information of the books [3].

As an Identification

Besides, QR code can also be found on train tickets, boarding passes or even ID cards in some countries. As an identifier of individuals, QR code can carry all the important information of a person yet hiding them in its pattern which better protect everyone's privacy.

Multimedia Applications

Not only printed on papers, there're also applications which can generate QR code on mobile devices. One typical application is the News Flash [4] developed by a team of MIT. News Flash is an application on iPad which shows a frequently flashing QR code on the screen. The colour of the QR code changes so quickly that human eyes can only see the mixture of the colours. Therefore, QR code can be hidden in the background colour. Another smartphone application called Wechat [5] is an instant messaging software which uses QR code as the user's name card. By scanning other people' QR code name card, one can add the person to his or her contact.

1.1.2 Existing Systems

AT&T Code Scanner

AT&T Code Scanner is a commercial application which runs a barcode scanner on a mobile phone. In most cases, it is used to help users read the barcode on a product and find information of this product online such as price and manufacture. An important feature of the system is that it depends on a net-based service which means it cannot decode solely on the mobile device itself.

ZXing Barcode Scanner

Different from AT&T Scanner, ZXing Barcode, also known as Zebra Crossing Barcode Scanner provides an offline decoding function. A user does not need to connect to the internet to decode the information in a barcode instead all the works are finished on his or her mobile device.

1.2 Motivation

The current QR code can expand its capacity by simply increasing its size and putting more pixels into one QR code .Currently, there are 40 different versions of QR codes which support capacity ranging from 7 Bytes (Version 1) to 2953 Bytes (Version 40) [6]. However, there are always physical limitations for both the medium which showsa QR code and the device which runs a scanner. In order to make sure that the

information stored in a QR code can be decoded by other devices, each module (the minimum unit in a QR code) should contain at least 4 dots in both of its dimensions [7]. Thus, there would not be enough space to show a big QR code pattern on a screen since commonly these devices would not have a large screen. Also, the resolution of the camera of the mobile devices will not be capable to capture all the necessary information if there are too may pixels.

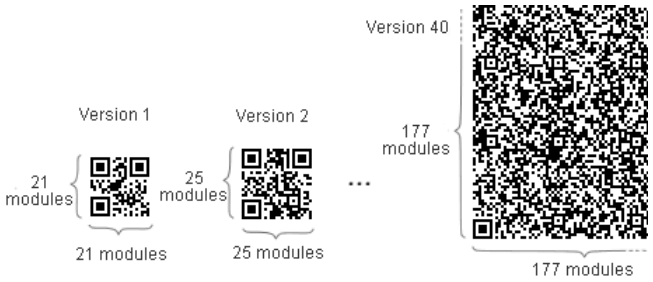


FIGURE 1 QR CODE OF DIFFERENT VERSIONS, FROM SMALLEST TO LARGEST [6]

Therefore, the idea of simply putting more information into a single QR code is not practical on mobile devices. Accordingly, we should take other methods to extend its capacity.

Instead of expanding it into space, we can extend its capacity by storing information into a series of QR code patterns, making each picture relatively smaller, but showing a series of QR code. Thus, the data can be transmitted in a more effective manner. Therefore, the physical limitation of the capacity of QR Code can be exceeded in this way.

1.3 Objectives

This project is aimed on developing an application which can generate and read QR codes in continuous mode transferring data on different Android smart phones. The software will have the following features:

- Automatically captures QR code.
- Automatically recognize and categorize the content of the data (characters or binaries).
- Automatically generate series of QR code and play them on the screen.
- Compatible to static QR codes.
- Error detection and error correction.
- Communicating control functions which guarantees the efficiency and accuracy of information flow.
- Data encryption function.

1.4 Methodology

1.4.1 DESIGN

The system design will be splited into different layers with each layer taking relative short period of time and less effort to be finished. This will also give convenience to carrying unit test and debugging.

1.4.2 IMPLEMENTATION

The software system will be developed in Java on Eclipse with Android SKD. Instead of using an Android virtual machine, the codes will be mainly tested on an Android smartphone.

The process of implementation will follow “bottom up” sequence. So the first steps will be concentrated on QR encoding and decoding. Functions of higher layers or of additional features shall be added later. During

the process of implementing, several experiments will take place giving essential information of the performance evaluation of the finished parts.

1.4.3 Use of Existing Open-source Projects

There are several existing open-source QR encoding and decoding projects. My project which is not concentrating on static QR code image decoding will be based on these existing sources codes with new features built upon them. These open source libraries are:

Open Source QR Code Library [8]on Sourceforge.com

This is an open sources QR-decoding library which is written in Java by some online developer. The current version of the library is v0.9 which can provide basic decoding function of QR code. The library now does not provide function for QR encoding although it aims to do that. It uses a GNU General Public license which is very convenient for people to spread it or make their own programme with it. This project stopped updating its code in 2008 for some unknown reason, yet it is still one of the most widely spreaded QR code libraries on the internet. In this project, the decoder will be built upon this library.

ZXing Library [9]

ZXing, also known as Zebra Crossing is an open source project on Google's Project Hosting. It is an open source QR code library under Apache Licence. Currently, it is still an on-going project with full functions in encoding and decoding of several kinds of barcodes. Originally written in C++, ZXing is transplanted to J2SE and Android. As a part of the project, it also provides an Android QR decoder application which can be downloaded from Google Play for free [10]. In this project, the encoder will be built upon this library.

1.4.4 Collecting User's feedback

Users' feedback can play a vitally important role in software development. In order to get more feedback from different users, 2 main collecting methods will be taken. The first method is to ask some users to go on a trial and fill in an evaluation form or go on an interview. The second method as a complementary is to make the application available on Google Play, Google's Android market which will have wider range of users yet the quality of evaluation might not be guaranteed.

1.4.5 Evaluation

The methods of evaluation will be focused on 2 key factors, the error rate and the speed. Experiments with controlled conditions will be applied to the system in order to carry out the evaluation. The evaluation will give important information on whether the system performance is acceptable or how to optimize the system in the future.

2 Design and Implementation

2.1 System Level Design

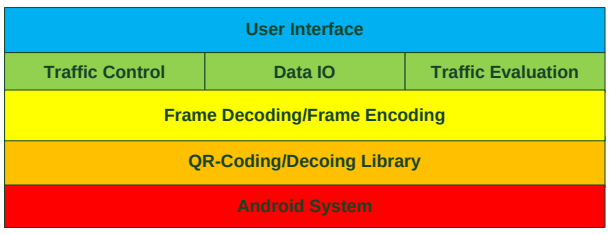


FIGURE 2 SYSTEM STRUCTURE

In order to have a clear and high efficient procedure of design and implantation, this project will follow a process in which the works are divided into different layers according to their functions. Upon Android system 4 layers of different functions will be built.

2.2 Experiments on the libraries

Before the libraries are used in this project, some important factors should be clear in the first place. These factors may include performance of the libraries, the standard format of using the libraries, and the effects on the system once these libraries are used. In order to collect all these information, several experiments have been designed and taken.

2.2.1 Experiments on Open Source QR Code Library

Error Detection Test

The library itself provides error detection which will give the system feedback with the type of error occurring if it can't decode the input image. In order to know the performance of the library when it encounters a typical error in the inputting code, several QR code with different kind of errors occurring on them are generated. These errors could be:

Part of the code is covered yet its required pattern of positioning is not violated and the input image is of high resolution. If the part that is covered or lost is small enough that the redundant information in QR code can tell what the original information is, the decoder will come to the right result. This depends on the error correction level of the QR generator which can provide different levels of error correction (from 7% to 30%) [11].

The second kind of error is that the positioning part is lost. No matter how good the quality of input image is, the decoder will always return an image not found error.

Finally, a situation with poor resolution of input image yet the input image contains the whole QR code. In this experiment, several QR code images are generated in different resolutions. The typical minimum resolution for a QR code containing 80 bytes is 100x100 pixels while for a QR code containing 160 bytes the values is 150x150 pixels. There is not a distinctive line between a successful or unsuccessful decoding in the real practice. Sometimes, when the resolution is not good enough, the system will return information of successful decoding yet with flaws in the decoded data.

Speed Test

In order to know how fast the system can run and calculate the throughput of the decoder, a speed test was made. The time that will take the decoder to decode an input QR code relies both on the resolution of the input image and on the number of bytes encoded in a single image. To get an accurate value of decoding time, each decoder was set to work 8 times and calculate the total time that has been spent. The result is shown as follows:

Storage in Bytes	Resolution	Time in MS
80	376x376	2566

Storage in Bytes	Resolution	Time in MS
80	223x223	1370
80	150x150	906
80	135x135	return error
160	504x504	5087
160	252x252	2051
160	200x200	1704
160	150x150	return error

FORM 1 RESULT OF SPEED TEST

According to the result shown above, the speed of the decoding library depends on both the input resolution and the amount of information that a QR code holds. The highest speed that the decoding library can work at (under an ideal condition) is approximately 700Bytes per second.

2.2.2 Experiments on ZXing Library

There're more experiments that will be taken place in the Future aiming at testing the performance of ZXing library. Although only half of these experiments are finished now, the code of the decoder system which is already implemented has an independent and exchangeable part that is linked to libraries which can be easily changed without interfering other parts of the system.

2.3 Frame Structure and Inter-frame Protocol Design

2.3.1 Frame Structure

Index	M	E	N	Data
Data				
…				

FORM 2 FRAME STRUCTURE

- Index: Field of 1 Byte, integer type value indicating the sequence of the frames. 0 is only for Control Frames.
- M (More): Field of 1 Bit, Boolean type value indicating whether it is the end of a session.
- E (Encrypted): Field of 1 Bit Boolean type value indicating whether the frame is encrypted.
- NS: 6 Bit field remains for further uses.

2.3.2 Control Frame Structure

Index	M	E		Type	Length
Version	Name				
…					

FORM 3 STRUCTURE OF CONTROL FRAME

Besides all the information contained in a normal frame, a Control Frame also has:

- Type: Field of 1 Byte integer type value indicating the form of information transmitting.
- Length: Field of 1 Byte integer type value indicating the number of frame in the current session.
- Version: Field of 1 Byte integer type value indicating the version of QR-Motion. Currently set to 0.
- Name: Field with unspecified length, always end with 0x03(End of Text in ASCII). This field is used to name the current session(or the filename of the file being transmitted)

2.3.3 Receiver State Machine

In this design, the transmitter will transmit all the frames in a loop. For each loop, there will be at least one control frame holding all the control information followed by several data frames. The receiver of the system should be able to capture and decode the QR code coming from its camera. Then the receiver system decodes the information in the frame, categorize it and find control information. Also it should stop at the right stage when it has already collected all the data being transmitted. The receiver system has different jobs to do when it is in one of the following state:

- **Idle:** The system is waiting to start.
- **Wait:** The system waits for the clock thread to call it to start to work on the current incoming frame.
- **Decode Incoming Frame:** The system is triggered by the clock thread and start to decode the image it captured. And then, the system judge whether the frame is the same frame as the previous one. If it is an old frame then states **does not change**, otherwise it move to the state of **Decode Frame**.
- **Decode Frame:** The system decode the current frame and judge whether it is a control frame or a data frame. If the current frame is a control frame it then moves on to the state of **Renew Control Info**, otherwise it moves on to **Store Data**.
- **Store Data:** The system save the data by pushing the current frame into an array-list. Then it will look at all the received data to see whether all the information has been received. If so, it will turn to the state of **Assemble Frames**, otherwise it will move on to the state of **Wait** to start a new loop.
- **Renew Control Info:** When the system receives a control frame, the system will then look at its control information. If there is no control information received then it stores all the information. If the control information is already in place, then the system simply drops the new frame. Then it will check whether all the frames has been received, if so the system will turn to **Assemble Frames**, otherwise it will turn to **Wait**.
- **Assemble Frames:** The system will traverse all the element in the data array then put then in the right order according to their index, then save the file with the filename given in the control frame. Then the system will be **Terminated**
- **Terminated**

The whole procedure can be translated into the following state machine diagram:

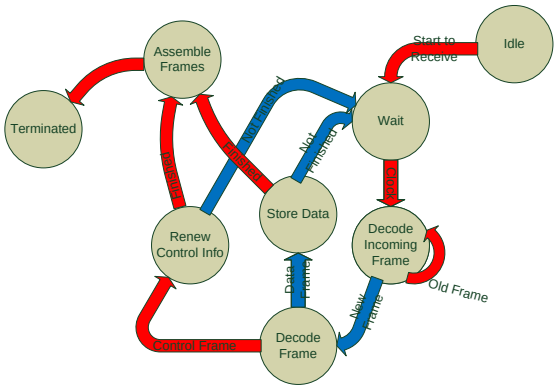


FIGURE 3 STATE MACHINE OF RECEIVER (FOR A LARGER VIEW PLEASE REFER TO **APPENDIX 1**)

The figure above shows how the receiver operates when it starts to receive QR code. There are three main parts of this state machine:

- A thread triggering the camera to capture and decode an image.
- A module which looks for control frames and decodes control information once a control frame is found.
- A module monitoring all the received frames and to judge whether the transmission is finished.
- A module decoding all the incoming frames
- A module of doing file operations

2.4 Implementing the receiver

2.4.1 General Procedure of Android Development

The procedure of developing an Android application can be simply divided into 4 phases including “Setup”, “Development”, ” Debugging and Testing”, ”and Publishing” [12].

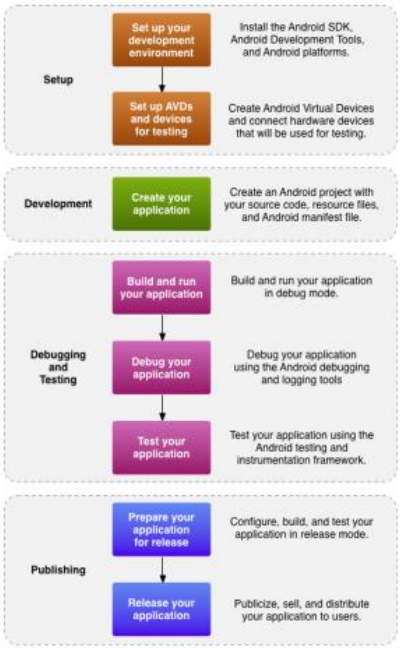


FIGURE 4 GENERAL STEPS OF ANDROID APPLICATION DEVELOPMENT [12] (FOR A LARGER VIEW PLEASE REFER TO **APPENDIX 2**)

Setup is to setup the develop environment on a computer. First to download Eclipse, link all the necessary android SDKs, and prepare other tool kits for Android development. After that a devices running Android should be linked to the computer and tested by some simple software to verify whether the device works well with the development environment.

Development is to design and write codes for the application. There're 2 kinds of files that will be needed when developing an Android application. Dot java files describe the most important contents of a programme in which the objects and behaviors are described. Dot XML files hold the Graphic User Interfaces and properties of the application. These files are grouped and organized in different packages according to their different functions.

Debugging and Testing always take place along with developing. One important feature of Android development platform for debugging is a build-in system log monitor which shows all the important messages and report of errors. This information monitor is extremely helpful when an error occurs since the cause of the error can be identified and tracked in the source code.

Publishing is release the application to users, a popular to publish Android application is through Google Play, smart phone manufacture like Samsung and Motorola also have app-shop for their customers.

2.4.2 Preparatory works

Before started programming several important preparatory works should be finished in the first place. This include both hardware preparation and software preparation.

Hardware Preparation

The smart phone for testing this system is my Motorola XT890 with both front camera and rear camera (A later version of bilateral communication would need 2 cameras). This smart phone runs Android 4.0.4 as its operating system. The phone is set into debugging mode and can download a temporary .apk file from a computer when connected with a Micro-USB cable.

Software Preparation

An Eclipse was downloaded and installed to my computer with a latest Android SDK. Other works include creating an Android virtual machine for debugging and downloading the relevant Android API packs which might be useful when developing this application for different versions of Android. The development environment is created and can be now used to develop a real application.

2.4.3 Programming

According to the different functions of each module, the codes are further divided into several classes under the path "cam.home". The"CamActivitity.java" takes charge of capturing images and controlling the whole systems. The "QRFrame.java" and "QontrolFrame.java" are used to decode a byte stream of information in to a standard frame. "FileService.java" does all the file operations.

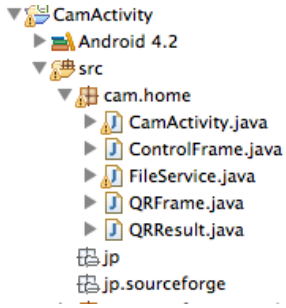


FIGURE 5 PACKAGE AND CLASSES

The "CamActivity.java" holds the most important functions of the whole receiving system whose function can be described by the following flowchart shown in Figure 6. This class can be seen as the implementation of the state machine described in 2.3.3.

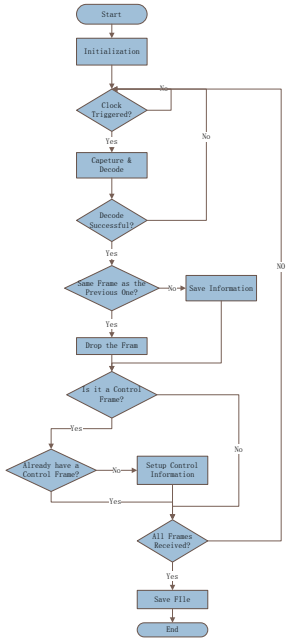


FIGURE 6 RECEIVER FLOW CHART (FOR A LARGER VIEW PLEASE REFER TO **APPENDIX 3**)

By Monday 26th November 2012, the code of the receiving part is finished. The system can now capture QR codes in series from a video clip modeling the sender, decode the information, and put them into the right order. The Figure 7 shows how this application looks when running.

FIGURE 7 THE USER INTERFACE OF THE APPLICATION

2.4.4 User Interface

The user interface can be created by using the build-in UI tool in Android SDK. The information needed to create a user interface can be pre-stored in a file called main.xml, or it can be dynamically created by java codes when the programme is running. Figure 8 shows the GUI that the application uses.

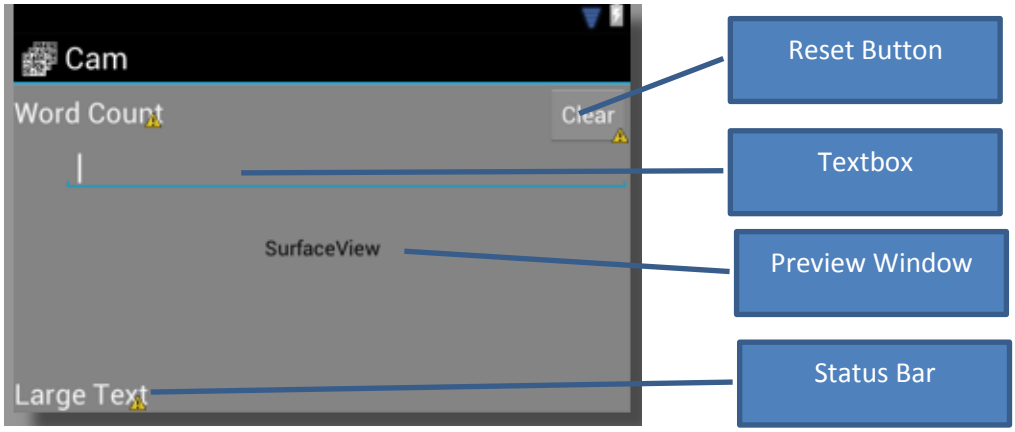


FIGURE 8 GUI OF QRMOTION

The GUI is built up by the following parts:

Status Bar shows the status of the current transmission. The status may include the total number of frames of the current sessions, the quality of the incoming frame, or the process of the whole session.

Reset Button can reset the current transmission, it will clear the data buffer and restart receiving information. This can be used anytime during debugging or an error occures.

Textbox shows the information in the QR codes being transmitted if the QR codes are holding text information, or it will shows the file name of the file being transmitted if the QR codes are in byte mode.

Preview Window shows the image taken by camera from which an user can know whether the camera is facing to a QR code in a proper direction and angle.

3 Plan for Semester 2

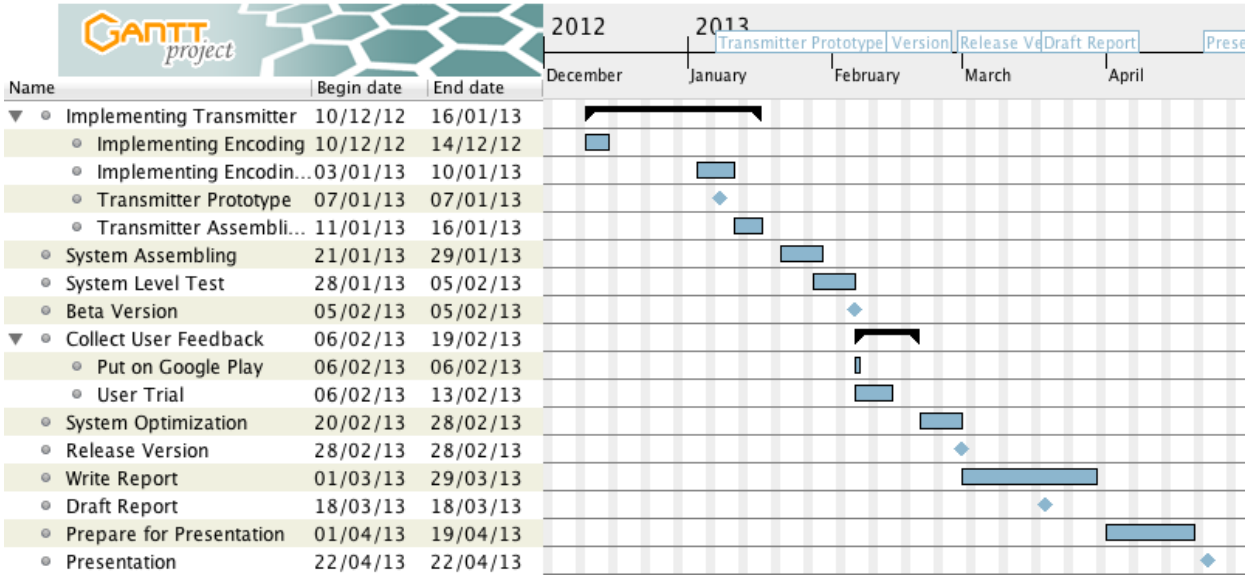


FIGURE 9 GANT CHART OF SEMESTER 2

Based on the works that have been finished in semester 1, several minor changes in plan for semester 2 are made. In this revision more time will be allocated to optimizing the performance and collecting users' feedback.

Another job to do in semester 2 is to design and build a simple bilateral communication system. The bilateral communication system will allow two devices talk together in QR code face to face simultaneously. Considering the time and work burden of the whole project, this task will be put in a secondary places which will be started after finishing most of the works in semester 2.

4 Risk Assessment

Description of risk	Description of impact	Likelihood rating	Impact rating	Preventative actions
Some function is not supported by Android system.	This part of function will not be implemented or would be rather difficult to find another solution.	Low	Very High	Making functional experiments before implementation. Change the design quickly once a problem is found
Poor performance of the software like low speed or high error rate.	Bad user experience.	Medium	Medium	Now the software is using real-time decoding, if the performance is very annoying, change the system to do post-decoding.
Not enough feedback	Will have influence on	Medium	Low	Post the application to

from users.	system optimizing.			Google's Android Market.
Poor Compatibility	Will not run on different phones or have poor performance on other phones	High	Medium	Test on different phones
System crash on the developing phone	Cost hours to restore the operating system	High	Low	Test on the virtual machine in the first place every time a big change is made
Important files damaged or lost	Can be fatal, if no backup is made	Medium	High	Use online backup and restore.
Debugging takes too long time	The project may be lagged behind.	Medium	High	Reschedule the project, do the easier parts first.
Potential Requirements Changes	Make it extremely difficult to proceed and may abandon previous works	Medium	Medium	Communicate with supervisor as early as possible.

FORM 4 RISK ASSESSMENT

Reference

[1] B. E. Massis, “QR codes in the library,” *New Library World,* vol. 112, p. 466–469, 2011.

[2] ISO/IEC, "ISO/IEC 18004:2006,Information technolog, Automatic identification and data capture techniques, QR Code 2005 bar code symbology specification," 2006. [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=43655. [Accessed 10th November 2012].

[3] Library of University of Bath, “Using QR Code,” [Online]. Available: http://www.bath.ac.uk/library/services/qrcode.html. [Accessed 1st December 2012].

[4] Z. Honig, “News Flash uses high-frequency light to transmit data from iPad tosmartphone,we go hands-on Mobile,” 24th April 2012. [Online]. Available: http://www.engadget.com/2012/04/24/mit-media-lab-newsflash-hands-on. [Accessed 1st December 2012].

[5] Tencent, “Wechat,” [Online]. Available: http://www.wechatapp.com/en/. [Accessed 2nd December 2012].

[6] Denso Wave Inc, “Symbol Version,” [Online]. Available: http://www.qrcode.com/en/qrgene2.html. [Accessed 20th Novemver 2012].

[7] Denso Wave Inc., “Printer Head Density and Module Size,” [Online]. Available: http://www.qrcode.com/en/qrgene3.html. [Accessed 2nd December 2012].

[8] Open Source Team, “Open Source QR Code Library,” [Online]. Available: http://qrcode.sourceforge.jp/index.html.en. [Accessed 2nd November 2012].

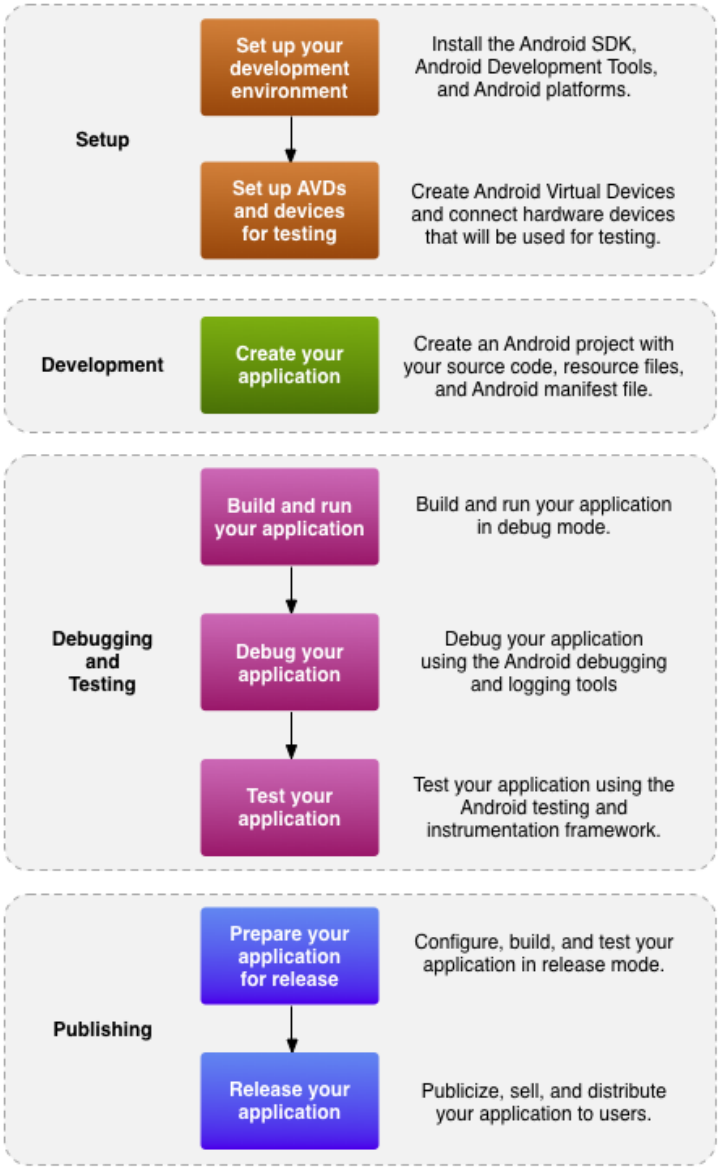
[9] ZXing Team, “ZXing,” [Online]. Available: http://code.google.com/p/zxing/. [Accessed 21st November 2012].

[10] ZXing Team, “Barcode Scanner,” [Online]. Available: https://play.google.com/store/apps/details?id=com.google.zxing.client.android. [Accessed 21th November 2012].

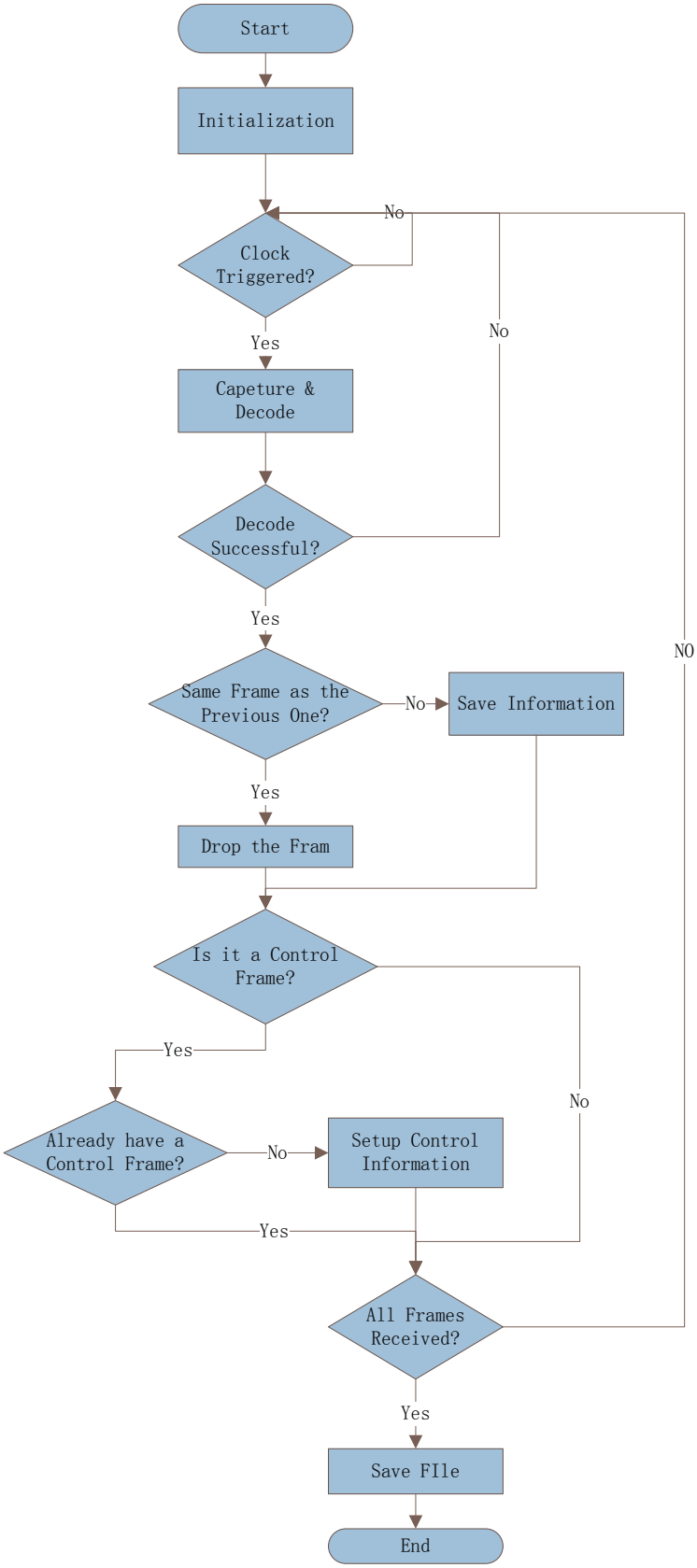
[11] Denso Wave Inc., “QR Code Outline Specification,” [Online]. Available: http://www.qrcode.com/en/qrstandard.html. [Accessed 25th November 2012].

[12] Google, “Android Developer Workfolw,” [Online]. Available: http://developer.android.com/tools/workflow/index.html. [Accessed 2nd December 2012].

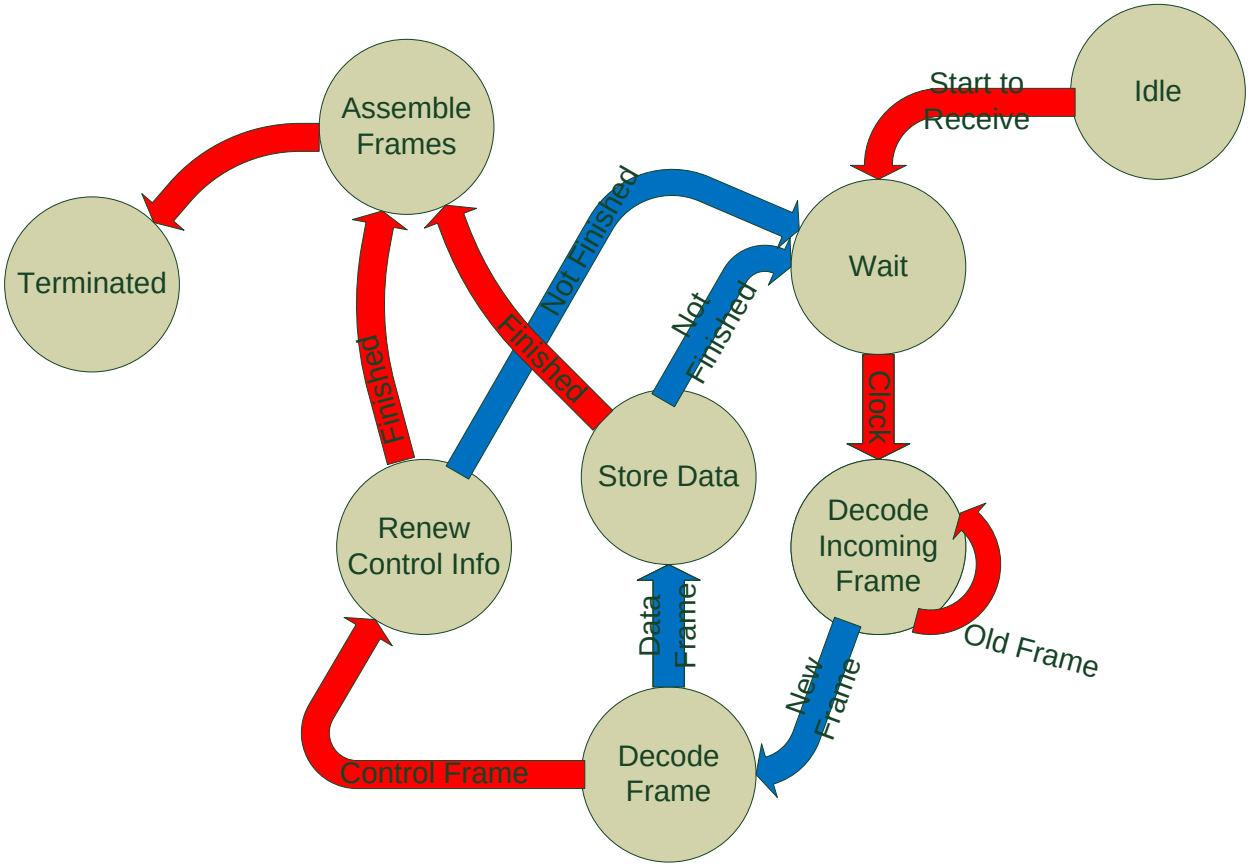
Appendix1 [Receiver State Machine]



Appendix2 [Receiver State Machine]



Appendix3 [Receiver State Machine]



Appendix4 [CamActivity.java]

```
1  package cam.home;
2  import java.io.BufferedOutputStream;
3  import java.io.ByteArrayOutputStream;
4  import java.io.DataOutputStream;
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.FileOutputStream;
8  import java.io.IOException;
9  import java.io.OutputStream;
10 import java.util.ArrayList;
11 import RQ.DEC.QRCodeDecoderCUIExample;
12 import android.app.Activity;
13 import android.content.Context;
14 import android.content.Intent;
15 import android.content.pm.PackageManager;
16 import android.graphics.Bitmap;
17 import android.graphics.BitmapFactory;
18 import android.graphics.BitmapFactory.Options;
19 import android.graphics.Canvas;
20 import android.graphics.ImageFormat;
21 import android.graphics.Matrix;
22 import android.graphics.Paint;
23 import android.graphics.Rect;
24 import android.graphics.YuvImage;
25 import android.hardware.Camera;
26 import android.hardware.Camera.PictureCallback;
27 import android.hardware.Camera.ShutterCallback;
28 import android.hardware.Camera.Size;
29 import android.os.Bundle;
30 import android.os.Environment;
31 import android.os.Handler;
32 import android.os.Vibrator;
33 import android.util.Log;
34 import android.view.SurfaceHolder;
35 import android.view.SurfaceView;
36 import android.view.View;
37 import android.view.View.OnClickListener;
38 import android.view.Window;
39 import android.view.WindowManager;
40 import android.widget.Button;
41 import android.widget.EditText;
42 import android.widget.ImageView;
43 import android.widget.TextView;
44 public class CamActivity extends Activity implements SurfaceHolder.Callback {
45       private Camera mCamera;// CameraObject
46       private ImageView mButton;// Button for CLear
47       private SurfaceView mSurfaceView;// Image holder surfaceView
48       private SurfaceHolder holder;// SurfaceView controller
49       private AutoFocusCallback mAutoFocusCallback = new AutoFocusCallback();// AutoFocusCallback to define what to do when capeturing
50       EditText result;
51       private TextView status;
52       private int Resolution=80;
53       private long delta;
54       Bitmap catchedBitmap;
55       private ControlFrame mControlFrame=null;
56       private TextView wordCount;
57             private ArrayList<QRFrame>ResultPool=new ArrayList();
58             private QRFrame CurrentFrame=new QRFrame(null);
59             private QRFrame PreviousFrame=new QRFrame(null);
60             private File testFile=new File("/sdcard/","B.txt");
61       Bitmap processedBitmap;
62       QRCodeDecoderCUIExample QR=new QRCodeDecoderCUIExample();
63       private static String strCaptureFilePath = Environment
64                   .getExternalStorageDirectory() + "/DCIM/Camera/";// path to save
65       Handler handler = new Handler();
66       Runnable updateThread = new Runnable(){
67             public void run(){
68               //mCamera.autoFocus(mAutoFocusCallback);// call mCamera
```

```
                //takePicture();
                long start=System.currentTimeMillis();

                //decodebyVideo(catchedBitmap);
                decodeBytebyVideo(catchedBitmap);

                long end=System.currentTimeMillis();
                Control();
                delta=end-start;
                handler.postDelayed(updateThread, 300);
                /*Vibrator mVibrator;
                mVibrator = (Vibrator) getSystemService(VIBRATOR_SERVICE);
                mVibrator.vibrate(40);      */
            }
        private void decodeBytebyVideo(Bitmap bm){
            if(bm!=null){
                    bm=Scale(bm,2);
                    CurrentFrame=new QRFrame(QRdecodingByte(bm));
                    if(CurrentFrame.getIndex()==-1){status.setText("Decode Failed");}
                            else{
                    if(CurrentFrame.getIndex()==PreviousFrame.getIndex()){status.setText("Idel");}//if the frame stays still, nothing shall be done.
                    else{
                            boolean hasthis =false;
                            for(int i=0;i<ResultPool.size();i++){
                                    if(ResultPool.get(i).getIndex()==CurrentFrame.getIndex())hasthis=true;
                            }
                            if(!hasthis){
                            ResultPool.add(CurrentFrame);
                            status.setText("Incoming"+CurrentFrame.getIndex());
                            }else status.setText("Duplicate"+CurrentFrame.getIndex());
                    }
                    wordCount.setText("Frame Count:"+ResultPool.size());
                    if(CurrentFrame.getIndex()!=-1)PreviousFrame=CurrentFrame;
            }
            }
            else status.setText("No Pic");
        }
                    private void decodebyVideo(Bitmap bm) {

                            if(bm==null){
                                    status.setText("No Pic");
                            }
                                    else{
                                            bm=Scale(bm,2);
                                            QRResult mqresult=QRdecoding(bm);
            if(mqresult.getStatus()==0){
            result.setText(result.getText()+mqresult.getResult());
            wordCount.setText("Word Count:"+result.length());
            status.setText("OK-"+String.valueOf(delta)+"ms");
            }
            else status.setText("Error:"+mqresult.getStatus());
                                    }
                            // TODO Auto-generated method stub

                    }

};
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN, WindowManager.LayoutParams.FLAG_FULLSCREEN);
    this.requestWindowFeature(Window.FEATURE_NO_TITLE);
    if (checkCameraHardware(this)) {
        Log.e("============", "Cam Exist");
    }
    /* Hide status bar */
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    /* Hide title */
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    /* Horizontal Screed */
    // this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

```
142             setContentView(R.layout.main);// ----------------------
143             /* SurfaceHolder setting */
144             mSurfaceView = (SurfaceView) findViewById(R.id.mSurfaceView);
145             holder = mSurfaceView.getHolder();
146             holder.addCallback(this);
147             // holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
148             /* set Button's OnClick event */
149             result=(EditText) findViewById(R.id.editText1);
150             wordCount=(TextView) findViewById(R.id.textView2);
151
152             result.setText("");
153             status=(TextView) findViewById(R.id.textView1);
154             /*mButton = (ImageView) findViewById(R.id.myButton);
155             mButton.setOnClickListener(new OnClickListener() {
156                 @Override
157                 public void onClick(View arg0) {
158                     capture after auto focus
159                     mCamera.autoFocus(mAutoFocusCallback);// call mCamera
160                     takePicture();
161                 }
162             });*/
163
164             sendImageIv = (ImageView) findViewById(R.id.send_image);
165
166
167
168
169
170             //new thread created
171
172 try {
173               testFile.createNewFile();
174 } catch (IOException e) {
175               // TODO Auto-generated catch block
176               e.printStackTrace();
177 }
178
179             handler.post(updateThread);
180             sendImageIv.setOnClickListener(new OnClickListener() {
181                 @Override
182                 public void onClick(View v) {
183                     Intent i = new Intent();
184                     i.setType("image/*");
185                     i.setAction(Intent.ACTION_GET_CONTENT);
186                     startActivityForResult(i, Activity.DEFAULT_KEYS_SHORTCUT);
187                 }
188             });
189             Button Button1= (Button)findViewById(R.id.button1);
190             Button1.setOnClickListener(new OnClickListener(){
191                 @Override
192                     public void onClick(View v) {
193                         result.setText("");
194                         wordCount.setText("Word Count:");
195                         ResultPool.clear();
196                     }
197             });
198             ResultPool.clear();
199         }
200         // ///////----------overwrite the interface of SurfaceHolder.Callback
201         @Override
202         public void surfaceCreated(SurfaceHolder surfaceholder) {
203             try {
204                 mCamera = null;
205                 try {
206                     mCamera = Camera.open();
207                 } catch (Exception e) {
208                     Log.e("============", "Camera In Use!");
209                 }
210                 if (mCamera == null) {
211                     Log.e("============", "No Camera");
212                     System.exit(0);
213                 }
214                 mCamera.setPreviewDisplay(holder);//Setting of display pad
```

```
215
216                 priviewCallBack pre = new priviewCallBack(){//create preview call back
217                 ///////backtohere///////
218                 @Override
219                 public void onPreviewFrame(byte[] data, Camera camera) {
220                         Camera.Parameters parameters=camera.getParameters();
221                         parameters.setPreviewFrameRate(50);//frame rate
222                           int w = parameters.getPreviewSize().width;
223                         //int h = parameters.getPreviewSize().height;
224                         //int w = parameters.getPictureSize().width;
225                         //byte[] mByteArray = new byte[w * h * 3];
226                         catchedBitmap=decodeToBitMap(data,camera);
227                         //int [] mArray=new int[w*h*3];
228                         //catchedBitmap = BitmapFactory.decodeByteArray(data, 0,data.length);
229                 }
230                 };
231                 mCamera.setPreviewCallback(pre);
232                 //mCamera.getParameters().setPreviewFormat(ImageFormat.JPEG);
233                 mCamera.startPreview();//preview start
234             } catch (IOException exception) {
235                 mCamera.release();
236                 mCamera = null;
237             }
238         }
239         @Override
240         public void surfaceChanged(SurfaceHolder surfaceholder, int format, int w,
241                 int h) {
242             initCamera();
243         }
244         @Override
245         public void surfaceDestroyed(SurfaceHolder surfaceholder) {
246             stopCamera();
247             mCamera.release();
248             mCamera = null;
249         }
250
251         /* method of taking picture*/
252         private void takePicture() {
253             if (mCamera != null) {
254               //mCamera.setPreviewCallback(cb)
255                 mCamera.takePicture(shutterCallback, rawCallback, jpegCallback);
256             }
257         }
258
259         private ShutterCallback shutterCallback = new ShutterCallback() {
260             public void onShutter() {
261                 /* call when shot */
262             }
263         };
264
265         private PictureCallback rawCallback = new PictureCallback() {
266             public void onPictureTaken(byte[] _data, Camera _camera) {
267             }
268         };
269
270         private PictureCallback jpegCallback = new PictureCallback() {
271                         public void onPictureTaken(byte[] _data, Camera _camera) {
272                 try {
273                     /* Get Picture */
274                     bm = BitmapFactory.decodeByteArray(_data, 0,
275                             _data.length);
276
277                     /* Create file */
278 //                    File myCaptureFile = new File(strCaptureFilePath, "1.jpg");
279 //                    BufferedOutputStream bos = new BufferedOutputStream(
280 //                            new FileOutputStream(myCaptureFile));
281                     /* Zip */
282 //                    bm.compress(Bitmap.CompressFormat.JPEG, 100, bos);
283
284                     /* Reset BufferStream */
285 //                    bos.flush();
286
287                     /* Terminate OutputStream */
```

```
//                      bos.close();

                        /* Reset camera after 2 secodes */
                        // Thread.sleep(2000);
                        /*Reset Camera */
                        stopCamera();
                        initCamera();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    //bm=Scale(bm,2);
                    long start=System.currentTimeMillis();
                    QRResult mqresult=QRdecoding(bm);
                    long end=System.currentTimeMillis();
                    delta=(long)(end-start);
                    if(mqresult.getStatus()==0){
                    result.setText(result.getText()+mqresult.getResult());
                    wordCount.setText("Word Count:"+result.length());
                    status.setText("OK");
                    }
                    else status.setText("Error:"+mqresult.getStatus());

                }
            };

            /* Set class AutoFocusCallback */
            public final class AutoFocusCallback implements
                    android.hardware.Camera.AutoFocusCallback {
                public void onAutoFocus(boolean focused, Camera camera) {
                    /* Get focus */
                    if (focused) {
                        takePicture();
                    }
                }
            };
            /* Camera Initialization */
            private void initCamera() {
                if (mCamera != null) {
                    try {
                        Camera.Parameters parameters = mCamera.getParameters();
                        // parameters.setPictureFormat(PixelFormat.JPEG);
                        parameters.setPictureSize(320,240);
                        parameters.setPreviewSize(640,480);
                        parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE);
                        mCamera.setParameters(parameters);
                        /* Start to Preview */
                        mCamera.startPreview();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }

            private QRResult QRdecoding(Bitmap bm) {
                 QRResult Qresult = new QRResult();
                int i=0;
                String info=null;
                info=QR.decodeByteInfo(bm);
                Qresult.setResult(info.substring(1, info.length()));
                Qresult.setStatus(Integer.parseInt(""+info.charAt(0)));
                return Qresult;
                            // TODO Auto-generated method stub
                 }
            private byte[] QRdecodingByte(Bitmap bm){
                 byte[] result=QR.byteDecode(bm);
                            return result;
            }
            private void stopCamera() {
                if (mCamera != null) {
                    try {
                        mCamera.stopPreview();
                    } catch (Exception e) {
                        e.printStackTrace();
```

```
361                 }
362             }
363         }
364
365         // Check Camera
366         private boolean checkCameraHardware(Context context) {
367             if (context.getPackageManager().hasSystemFeature(
368                     PackageManager.FEATURE_CAMERA)) {
369                 // Exist
370                 return true;
371             } else {
372                 // Nonexist
373                 return false;
374             }
375         }
376         class priviewCallBack implements Camera.PreviewCallback {
377                     @Override
378             public void onPreviewFrame(byte[] data, Camera camera) {
379                 // TODO Auto-generated method stub
380                 // Log.w("wwwwwwwww", data[5] + "");
381                 // Log.w("Support Format", mCamera.getParameters().getPreviewFormat()+"");
382                                 decodeToBitMap(data, camera);
383
384                 //////extra/////
385                 Options opts = new Options();
386                                 opts.inJustDecodeBounds = true;
387                                 int xScale = opts.outWidth;
388                                 int yScale = opts.outHeight;
389                                 opts.inSampleSize = xScale > yScale ? xScale : yScale;
390                                 catchedBitmap=BitmapFactory.decodeByteArray(data, 0, data.length, opts);
391                                 ///end extra/////
392             }
393         }
394         public Bitmap decodeToBitMap(byte[] data, Camera _camera) {
395             Size size = mCamera.getParameters().getPreviewSize();
396             Bitmap bmp = null;
397                         try {
398                 YuvImage image = new YuvImage(data, ImageFormat.NV21, size.width,
399                         size.height, null);
400                 Log.w("wwwwwwwww", size.width + " " + size.height);
401                 if (image != null) {
402                     ByteArrayOutputStream stream = new ByteArrayOutputStream();
403                     image.compressToJpeg(new Rect(0, 0, size.width, size.height),0, stream);
404                     bmp = BitmapFactory.decodeByteArray(
405                             stream.toByteArray(), 0, stream.size());
406                     Log.w("wwwwwwwww", bmp.getWidth() + " " + bmp.getHeight());
407                     Log.w("wwwwwwwww",
408                             (bmp.getPixel(100, 100) & 0xff) + "    "
409                                     + ((bmp.getPixel(100, 100) >> 8) & 0xff) + "    "
410                                     + ((bmp.getPixel(100, 100) >> 16) & 0xff));
411                     stream.close();
412                 }
413             } catch (Exception ex) {
414                 Log.e("Sys", "Error:" + ex.getMessage());
415             }
416             return bmp;
417         }
418           @Override
419           protected void onStop() {
420                         handler.removeCallbacks(updateThread);
421                         if(mCamera!=null){
422                                     mCamera.stopPreview();
423                                     mCamera.setPreviewCallback(null);
424                                     mCamera.release();
425                                     }
426                         //mCamera.stopPreview();
427                         // TODO Auto-generated method stub
428                         super.onStop();
429           }
430           public static Bitmap smallBitmap(Bitmap bm, int size) throws IOException {
431
432                                     Options op = new Options();
433                                     op.inJustDecodeBounds = true;
```

```
                                int xScale = op.outWidth / size;
                                int yScale = op.outHeight / size;
                                op.inSampleSize = xScale > yScale ? xScale : yScale;
                                op.inJustDecodeBounds = false;
                                File myCaptureFile = new File(strCaptureFilePath, "1.jpg");
                                                BufferedOutputStream bos = new BufferedOutputStream( new
FileOutputStream(myCaptureFile));
                                                bm.compress(Bitmap.CompressFormat.JPEG, 100, bos);
                                                bos.flush();
                                                bos.close();
                                bm = BitmapFactory.decodeFile("1.jpg", op);
                                return bm;
                                }
        public Bitmap Scale(Bitmap bm, int times){
                Matrix matrix = new Matrix();
                matrix.postScale((float)1.0/times, (float)1.0/times);
                Bitmap bm2=Bitmap.createScaledBitmap(bm, 4*Resolution , 3*Resolution, false);
                //Bitmap bm2=Bitmap.createBitmap(bm,0,0,bm.getWidth()/times,bm.getHeight()/times,matrix,true);
                return bm2;
        }
        @Override
        protected void onResume() {
       handler.postDelayed(updateThread, 300);
                // TODO Auto-generated method stub
                super.onResume();
        }
        @Override
        protected void onPause() {
                                handler.removeCallbacks(updateThread);
                // TODO Auto-generated method stub
                super.onPause();
        }
        @Override
        protected void onDestroy() {
                // TODO Auto-generated method stub
                super.onDestroy();
        }
        private void Control(){
                //checking whether the incoming frame is control frame
                if(CurrentFrame.getIndex()==0)//Incoming Control Frame
                {
                                if(mControlFrame!=null){
                                                //duplicated control frame incoming
                                }else{ mControlFrame=new ControlFrame(CurrentFrame.getAll());
                                status.setText("Control Frame"+mControlFrame.getName());
                                }
                }
                //checking whether received all the frames
                if(mControlFrame!=null){
                                result.setText("Incoming File "+mControlFrame.getName()+"of length"+mControlFrame.getLength());
                                boolean[] receivedAll=new boolean[mControlFrame.getLength()+1];
                                int size=ResultPool.size();
                                for(int i=0;i<size;i++){
                                int localsize =ResultPool.get(i).getIndex();
                                if(localsize<size)
                                receivedAll[localsize]=true;
                                }
                                for(int i=0;i<receivedAll.length;i++)
                                                if(receivedAll[i]==false)return;
                                handler.removeCallbacks(updateThread);
                                toFile();
                }
        }
        private void toFile() {
                // TODO Auto-generated method stub
                status.setText("Complete"+ mControlFrame.getName()+mControlFrame.getName().length());
                FileOutputStream stream=null;
                File file = new File(Environment.getExternalStorageDirectory()+"/QRM/","A.txt");
       try {
                stream = new FileOutputStream(file);
                status.setText("File "+mControlFrame.getName()+" saved");
                Log.i("FILE", "opened");
       } catch (FileNotFoundException e) {
```

```
507                     // TODO Auto-generated catch block
508                     e.printStackTrace();
509                     //status.setText("File "+mControlFrame.getName()+"not saved");
510                     Log.i("FILE", "NOT opened");
511             }
512             for(int i=1;i<mControlFrame.getLength()+1;i++){
513               for(int j=0;j<ResultPool.size();j++)
514                       if(ResultPool.get(j).getIndex()==i){
515                                                       try {
516                                                               Log.i("FILE", "Writing frame"+i+"@"+j);
517                                                               stream.write(ResultPool.get(j).getData());
518
519                                                       } catch (IOException e) {
520                                                               // TODO Auto-generated catch block
521                                                               e.printStackTrace();
522                                                       }catch (NullPointerException q){
523                                                               q.printStackTrace();
524                                                       }
525               break;}
526             }
527             try {
528                                     stream.close();
529                     } catch (IOException e) {
530                                     // TODO Auto-generated catch block
531                                     e.printStackTrace();
532                     }catch (NullPointerException q){
533                                     q.printStackTrace();
534                     }
535              }
536     }
537
```

Appendix5 [QRFrame.java]

```
1  package cam.home;
2  public class QRFrame {
3  private byte[] head=new byte[2];
4  private byte[] data=new byte[128];
5  private byte[] all;
6  private int index=-1;
7  Boolean More=false;
8  Boolean Encripted=false;
9  QRFrame(byte[] RawInfo){
10                 all=RawInfo;
11                 if (RawInfo!=null){
12                                 head[0]=RawInfo[0];
13                                 head[1]=RawInfo[1];
14                                 index=(int)head[0];
15                                 if((head[1]&(byte)(1))==(byte)(1))More=true;
16                                 else More=false;
17                                 if((head[1]&(byte)(2))==(byte)(2))Encripted=true;
18                                 else Encripted=false;
19                                 for(int i=0;i<(RawInfo.length-head.length);i++){
20                                                 data[i]=RawInfo[head.length+i];
21                                 }
22                 }else index=-1;
23
24 }
25 QRFrame(int index,Boolean More,Boolean Encripted,byte[] data){
26                 this.index=index;
27                 this.More=More;
28                 this.data=data;
29                 this.Encripted=Encripted;
30                 head[0]=(byte)index;
31                 head[1]=(byte)0;
32                 if(More==true)head[1]+=1;
33                 if(Encripted==true)head[1]+=2;
34                 for(int i=0;i<head.length;i++){
35                                 all[i]=head[i];
36                 }
37                 for(int i=0;i<data.length;i++){
38                                 all[head.length+i]=data[i];
39                 }
40 }
41 public int getIndex() {
42                 return index;
43 }
44
45 public Boolean getMore() {
46                 return More;
47 }
48
49 public byte[] getData() {
50                 byte[] psudo={77};
51                 if(data!=null)
52                 return data;
53                 else return psudo;
54 }
55 public byte toByte(){
56                 return (Byte) null;
57 }
58 public byte[] getAll(){
59                 return all;
60 }
61
62
63 }
64
65
```

Appendix6 [Control Frame.java]

```
package cam.home;
//|////////////////////////////////////////////Frame Structure//////////////////////////////////////////////////////////|
//|/Frame Srtucture/////////|/Index///|//More///|///Encrypted//|///////|/////////////////////data/////////////////////|
//|/Control Frame Structure/|/Index=0/|//More///|///Encrypted//|///////|//Type////|//Length//|//Version//|/////Name///ETX|
//|/Size/////////////////////|/1Byte///|//1Bit///|///1Bit///////|//6Bit/|////1Byte/|//1Byte///|///1Byte///|/Not Specified/|
//|/Index/////////////////////|0/////////|1.0//////|1.1////////////|1.2////|2/////////|3/////////|4/////////|5/////////////|
//|/Data Type/////////////////|///int///|/Boolean/|////Boolean///|///////|///Int////|///int////|////int////|////String/////|
//|/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////|
/**
 * @author wanghaoxuan
 *
 */
public class ControlFrame extends QRFrame {
	int Type=0;
	int Length=0;
	int Version=0;
	String name="";
	ControlFrame(byte[] RawInfo) {
		super(RawInfo);
		Type=RawInfo[2];
		Length=RawInfo[3];
		Version=RawInfo[4];
		for(int i=5;i<RawInfo.length;i++){
			if(RawInfo[i]!=3)//ASCII 3 for End of Text
				name+=(char)RawInfo[i]+"";
				else break;
		}
	}
	public int getLength() {
		return Length;
	}
	public void setLength(int length) {
		Length = length;
	}
	public int getType() {
		return Type;
	}
	public void setType(int type) {
		Type = type;
	}
	public int getVersion() {
		return Version;
	}

	public void setVersion(int version) {
		Version = version;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```