Check Image Processing: WebP Conversion and MICR Scan Android Application

Senior Project

Computer Engineering Department

California Polytechnic State University, San Luis Obispo

Trevor Bliss

April 2012

TABLE OF CONTENTS

I. INTRODUCTION

Problem Summary

This Android application was created for a check image processing company (hereafter referred to as the "Company") to mobilize their desktop software. Currently the Company only offers desktop applications that utilize their check image processing technology. Going mobile with their software provides more versatility to users and, provided they have a smartphone, the ability to access the Company's services from anywhere. Android was chosen as the target mobile operating system for it's widespread popularity and ease of distribution.

The Company's software attempts to successfully scan images of checks for the MICR code. The MICR code is located at the bottom left corner of a check and consists of two parts. The first 9 digits make up the routing number that maps the check to the issuing bank. The next set of numbers is the specific customers account number. The mobile application will mimic the functionality of the desktop software by sending an image of a check to a remote server, which processes and scans the check for the MICR code, and then the server will return the MICR code so the application can display it to the user. The advantage of this mobile application versus the existing software is that the check image will be retrieved by taking a picture with the Android's built-in camera and can be done from anywhere the phone can access the Internet (3G/4G network or Wifi). If the user does not receive the desired results they will have the option to retake the image and resend to the server to be processed again.

Objectives

The main objective for this project is to create an Android application that is easy to use, reliable, and fast. The user should be able to get the full functionality of the application the first time they open it without any instruction or guidance. Ease of use is a primary objective because the Company should be able to distribute the application to customers without making them read an instruction manual or go through any training.

The application, when handed off to the Company, should be usable to current customers. It is also important that the application be maintainable in the future. If the Company wishes to add/modify functionality they should be able to do so without hindering the performance of the rest of the application. Since the application is most likely to be used non-developers outside the company, reliability is key. A range of users at varying levels of technical knowledge should be able to use the application without it crashing or having erratic behavior.

II. BACKGROUND

Android

Android is a software stack developed by the Open Headset Alliance, which is led by Google. This software stack includes a Linux based operating system, middleware, and a variety of applications. Below in Figure 1 are the major components of the Android operating system. At the top level are applications that the user directly interacts with such as the phone or web browser. Below that are framework APIs that applications use to do things such as access hardware or build Views for the users to see. Android also includes libraries written in C/C++ to handle things such as image manipulation or SQLite databases. At the very bottom is the Linux kernel, which handles core functionality. It is also

important to note that each Android application runs on a Dalvik virtual machine and is its own process[2].

The first commercial version of Android was released in September 2008. Since its initial release, it has become massively popular at an extremely fast rate. As of December 30, 2011 it owns 47% of market share, followed by iOS at 28.7%. Android's large market share makes it a desirable option for mobile development. Selling apps on the marketplace is also very easy. All that is required is a one-time payment of $25 to Google. From there Google takes 30% of profits made from the sale of applications[1].

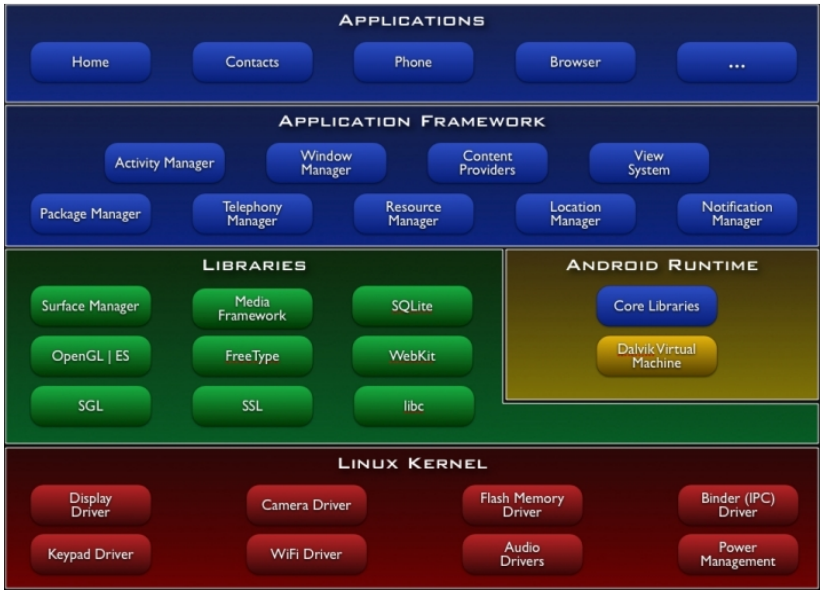

Figure 1. Android Architecture

Android NDK / JNI

The Android Native Development Kit (NDK) works in conjunction with the Standard Development Kit (SDK) to allow developers to run native C/C++ code with their standard Java code. Native code is placed in the <root>/jni/ folder of the project directory and compiled using an Android.mk file. This makefile sets compiler flags and links any other libraries you may be using in your code. The native C code will be compiled into a shared library, which can be imported into an Android project, and from there it is possible to call native functions from Java code. Java Native Interface (JNI) is used as a link between native code and the Android APIs[4]. JNI is also used for conversion of types between languages. In this project, for example, an Android Bitmap object is passed to the NDK. The Bitmap is taken in as type jobject and then JNI is used to get a pointer to the Bitmap data. Below is a mapping of the Java primitive types to their corresponding JNI type[3].

Java Language Type	Native Type	Description
boolean	jboolean	unsigned 8 bits
byte	jbyte	signed 8 bits
char	jchar	unsigned 16 bits
short	jshort	signed 16 bits
int	jint	signed 32 bits
long	jlong	signed 64 bits
float	jfloat	32 bits
double	jdouble	64 bits

Table 1. JNI primitive types

Using the Android NDK has advantages such as an increase in speed for CPU-intensive operations. For this project, however, I am using the NDK simply because the WebP libraries are only available in native C/C++. Thus, I am compiling the WebP libraries into a static library and writing my own native functions to use the conversion tools and then returning a converted image to my Java code. This may lead to a small speed boost but is generally not worth the added complexity if an equivalent Java solution is available.

WebP

WebP is Google's latest open source image format that provides both lossy and lossless compression. The project emerged from its sister project, WebM, which is an open source video format started by On2 technologies and acquired by Google in early 2010. WebP uses much of the same technology as WebM, but is designed specifically for still images[6]. The overall goal of WebP is to provide smaller file sizes without significant loss of quality. According to the WebP home page, images are 25-34% smaller than equivalent JPEG images and 28% smaller than PNG[5]. Google's motive in creating this format is an attempt to make browsing the web faster and smoother, thus encouraging users to use Google's services more often.

The biggest challenge WebP faces today is gaining the support of the Internet community. Currently Chrome and Opera are the only two browsers to natively support WebP[6]. Even native Android browsers before version 4.0 are unable to view these images. Files necessary for conversion are available for download for free from the WebP website. The two options for download are command line tools for converting individual images and libraries written in C for encoding/decoding images.

III. REQUIREMENTS

List of Requirements

Listed below are requirements for the project with a brief explanation. Some requirements are explained in greater detail below.

Requirement	Brief Explanation
Open Source	The project will be open source and available online.
Android 2.2	This application must work with Android 2.2 and above.
Compatible with the Company's Server	Check images will be sent directly to the Company's server with predetermined headers to specify incoming data.
WebP Images	All images are to be converted to Google's WebP file format before being sent to the Company's server for processing.
Check Image Dimensions	Images sent to the server should be as close to 1200x900 pixels as possible. This size will be large enough for the server to process but small enough for fast transfer speeds.
Camera Integration	Camera must open seamlessly within the application to take pictures of checks.
Function on Samsung Nexus S Phone	All testing will be done on the Samsung Nexus S phone.

Android 2.2

For this project, Android 2.2 (Froyo) was selected. With each Android version, new APIs are released which developers can use to write their applications. Froyo requires API level 8, which means it can use API calls from all earlier versions but nothing released after API level 8. Table 2 below shows the distribution of Android users. As you can see, this version includes almost 90% of Android users[7]. All new Android devices released ship with an OS compatible with API level 8 and some older devices receive over-the-air updates that bump them up in OS version. Froyo also contains memory and performance optimizations that may improve the usability of the application. Android 2.2 was ultimately chosen because it is the best compromise between performance improvements, API availability, and compatibility with devices currently on the market.

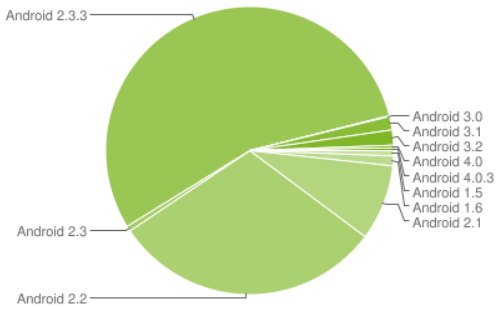

Figure 2. Usage share of the different versions as of February 1, 2012

Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.6%
Android 1.6	Donut	4	1.1%
Android 2.1	Eclair	7	8.5%
Android 2.2	Froyo	8	30.4%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.6%
Android 2.3.3 - Android 2.3.7		10	54.9%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.5%
Android 3.2		13	1.7%
Android 4.0 - Android 4.0.2	Ice Cream Sandwich	14	0.3%
Android 4.0.3		15	0.3%

Table 2. Distribution of Android versions

WebP Image Conversion

The images in this project are required to be converted into WebP format before being sent to the server. Conversion is to be done using the source code libraries written in C/C++ that are available for download from the WebP website. Time of conversion and picture quality are the two biggest concerns regarding the conversion of the image to WebP format. Although there are no hard limits on time for the conversion, it should not hinder the overall usability of the application. Picture quality also must be good enough for the server to scan the image and read the MICR code. If the user takes a picture properly, the server should successfully retrieve the code at a very high rate (>95%).

Server Communication

The overall goal of communicating with the server is to accurately retrieve the MICR code from the check image as fast as possible. Server communication time should not severely hinder usability of the application. After a taken picture is converted to WebP format it is sent to the server via a WiFi connection or over the 3G/4G network. Communication is done via sockets so the appropriate IP address and port number must be hardcoded into the application. Once the application successfully connects to the server and sends the check image it will return the scanned MICR code. If the MICR code is unable to be scanned an empty response will be returned, in which case the user will have the option to retake the picture. It should be noted that the operation and maintenance of the server are

done by the Company, and this application simply sends the check image to the server and acts accordingly depending on the response.

IV. DEVELOPMENT

Application Flow

The screen flow of the application is fairly simple. There are four screens for the user to interact with, plus the native camera app that is called within the application to take the picture. The main screen gives a welcome message and contains buttons that give the user options for how to take a picture (Screen 1). The returned image is then displayed to the user and gives him/her the option to convert the image to WebP format and send the converted image to the server. They also have the option to reopen the camera and retake the photo (Screen 2). Screen 3 shows a loading bar as the application is converting the image and sending it to the server. Finally, the scanned MICR code is displayed to the user (Screen 4). If the MICR code could not be read then an error message will be displayed.

Screen 1. Main/Home screen

Screen 2. Check image screen

Screen 3. Loading screen

Screen 4. MICR code screen

While navigating the application, the orientation of the screen is locked in portrait mode. This is done for consistency in UI design and because there is no need for the user to operate in landscape mode. By keeping the screen in a single orientation it also eliminates many potential bugs the application may encounter while switching between the two orientations. Locking the screen is done by simply adding the following line to the AndroidManifest.xml file:

```
android:configChanges="orientation|keyboardHidden"
```

The default back button functionality is also overridden to provide expected transitions between screens. If the back button is pressed from the home screen the application is closed and destroyed. From all subsequent screens the back button returns the user to the home screen. For this to be achieved the onBackPressed function must be overridden.

```
@Override
public void onBackPressed() {
   // Add desired screen navigation here
}
```

Camera

The camera is an integral part of this project. To upload a check image to the server the user must first open the camera on their phone. The decision was made to run the native camera application on the device rather than implementing a custom class to take the picture. The main reason for this decision is that the native camera is more reliable. All Android devices come with camera hardware built-in, as well as the software to allow users to take pictures. The app leverages the work done by the devices manufacturers and opens the native camera by calling the following code:

```
ContentValues values = new ContentValues();
values.put(MediaStore.Images.Media.TITLE, "pic");
values.put(MediaStore.Images.Media.DESCRIPTION,"Image capture by camera for WebPConv");
m_imageUri = getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, m_imageUri);
startActivityForResult(intent, WebPConv.CAMERA_NATIVE_RC);
```

This code gives control to the camera application and after the picture is taken the method "onActivityResults()" is called with the result of the image capture (picture taken or cancelled) and contains the raw image data in "m_imageUri". To turn the image URI into a Bitmap the following code is run:

```
InputStream inputStream = this.getContentResolver().openInputStream(m_imageUri);
BitmapFactory.Options opts = new BitmapFactory.Options();
opts.inPreferredConfig = Bitmap.Config.ARGB_8888;
opts.inSampleSize = 2;
m_imageBitmap = BitmapFactory.decodeStream(inputStream, null, opts);
```

Here setting the configuration to ARGB_8888 specifies the type of Bitmap ('A' in "ARGB" shows that there is an alpha and the "8888" means there are 8 bytes of precision on each channel). The sample size of 2 also scales the raw image by a factor of 2 before creating the bitmap. This not only creates the image close to the target dimensions, but it also significantly cuts back on memory and time of conversion. By running this code, consistency is maintained no matter what hardware the Android application is run on. A custom camera class may have different functionality based on where it is run.

Another reason the native software was chosen is because it is more maintainable. When the above code is called, the camera will return with an image and will always return in the same way. Implementation details of a custom camera class may change with new software/hardware updates. The introduction of front facing cameras is an example of having to modify the code to accommodate changes. Managing multiple cameras on a single device is more complex using the custom camera class approach, but the same call seen above may be made to get a picture from the native camera application.

Image Conversion

There are several steps involved in converting an image to WebP format. Once the Bitmap object is created from data returned from the camera, it can be passed to a background thread that will call the necessary functions to do the actual conversion. This thread is implemented as an AsyncTask. AsyncTask's allow you to run code on a separate thread in the doInBackground() method and also run code on the UI thread through other provided methods. This application runs a loading bar in the form of a spinner as the image is being converted and sent to the server. The spinner displays a short message to the user and allows the user to cancel the process by pressing the back button on their device. The following code is added to the onPreExecute(), which is executed on the UI thread before doInBackground() is called:

```
dialog.setOnDismissListener((OnDismissListener) this);
dialog.setMessage("Converting image to WebP format");
dialog.setCancelable(true);
dialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
dialog.show();
```

While this spinner is being displayed to the user, the image bitmap is sent to the NDK. This is a simple method call placed in the doInBackground() method:

```
doConvJniGraphics2(m_bitmap, WebPConv.IMG_QUALITY_FACTOR, m_fileName);
```

For the Android Java code to be able to communicate with the NDK the methods must be declared and the library loaded in the main Activity. The NDK library for this application is named “webpconv”.

```
public static native int doConvJniGraphics2(Bitmap pic, float quality_factor, String filename);
static {
        System.loadLibrary("webpconv");
}
```

At this point it is up to the NDK to convert the Bitmap to WebP format and save it to the phones SD card. The NDK provides a bitmap library that can be used to get bitmap information, simply by including android/bitmap.h. Using these libraries the bitmaps width, height, stride, and a pointer to the bitmap can be obtained. With this information the WebP library functions are called to produce a WebP image, saved to the SD card. See Appendix A for full source code.

Server Communication

Similar to converting the image to WebP format, server communication is done in the background using an AsyncTask. Instead of calling the NDK, however, a ClientActivity objet is created that contains methods for all communication with the server. TCP is the protocol used to communicate with the server. To establish an initial connection with the server, a socket must be opened on a specified IP address and port number. The IP address is hardcoded to the server run by the Company and the port number is held constant at 10000.

```
InetAddress serverAddr = InetAddress.getByName(SERVER_IP);
Socket m_socket = new Socket(serverAddr, SERVER_PORT);
OutputStream m_out = m_socket.getOutputStream();
InputStream m_in = m_socket.getInputStream();
```

The InputStream is used to read the byte stream received from the server. The most important information the application will receive from this stream will be the MICR code, but other information such as the user ID while connecting or text messages may also be received. The text message exchange functionality with the server is not implemented for this project, but may be incorporated by the Company in the future. The OutputStream is the channel used to send data from the application to the server. All messages sent to the server must begin with a header that describes the data contained in the rest of the message. The header is 64 bytes and is broken down into the following ints.

I. int 0: Held constant at 0x55AA33CC
II. int 1: Data type
    i. 0: connection/disconnect message
    ii. 1: start message for check image transfer
    iii. 2: check image segment
    iv. 4: request MICR data
III. int 2: Message type
    i. 0: error message
    ii. 1: connect
    iii. 4: disconnect
IV. int 3: Number of bytes that contain actual data
V. int 4: Segment number
VI. int 5: Total number of segments

VII. int 6: Information on how check image should be read if requesting MICR code
    i. 0: none
    ii. 1: rotated
    iii. 2: black edges
    iv. 3: both
VIII. int 7-15: Set to 0

To send the header to the server in a readable format to the server it must be sent in little endian and sent a single int at a time.

```
ByteBuffer byteBuffer = ByteBuffer.allocate(HEADER_INTS*4);
byteBuffer.order(ByteOrder.LITTLE_ENDIAN);
for(int i = 0; i < HEADER_INTS; i++){
        byteBuffer.putInt(header[i]);
}
byte[] data = byteBuffer.array();

// Send header file
for(int i = 0; i < HEADER_INTS; i++){
        int startIndex = i*4;
        m_out.write(data, startIndex, 4);
}
```

The data portion of the message is fixed at 448 bytes and can be sent as a single chunk. For connect and disconnect messages no data is necessary (all 0's). When sending the check image, however, the file must be broken down into chunks and sent 448 bytes at a time.

V. TESTING

Method of Testing

Thorough testing of the final application was done to choose the optimal configuration that will most accurately scan the MICR codes as well as minimize the time of conversion to WebP format and time communicating with the server. Three separate factors were chosen and altered in a number of combinations while running the tests. These factors are listed below:

1. **Quality Factor**: A number ranging from 0 to 100 used to specify the desired quality of the image. As shown below, the higher the number is the clearer the image will be, but the file size will also be larger.
2. **Method Parameter:** A quality/speed tradeoff parameter in the NDK when converting the image to WebP format. Set to 0 for fastest possible conversion and 6 for slower conversion time, but better image quality.
3. **Setting**: This includes multiple environment factors when taking the picture such as the surface on which the check is placed, lighting, distance from check, and whether the check is wrinkled or in perfect condition.

For these tests, quality factors of 20, 50, 80, and 100 were selected. The method parameter was set to 0, 1, or 2. Lighting can be ideal or requiring the flash from the phone. Settings for the pictures include black background with and without flash, wood background with and without flash, white tile

background, a picture from about 12” away (6” is ideal), and a picture of a wrinkled check. Most tests were run 5 times to get time averages, but the wrinkle and distance tests were only run twice.

Test Results

Tables 3, 4, and 5 show a comparison of different method parameters run in ideal conditions (black background with no flash). As you can see, as method parameter and quality factor increase, the time to convert and send to the server increases.

Quality Factor	Conversion Time	Server Time	Total Time
20	6.02	1.24	7.26
50	5.97	1.51	7.48
80	6.03	1.46	7.49
100	6.53	3.18	9.71

Table 3. Times (in seconds) of ideal picture settings with Method parameter 0

Quality Factor	Conversion Time	Server Time	Total Time
20	6.03	1.23	7.28
50	5.99	9.35	6.92
80	6.04	1.16	7.20
100	6.59	2.55	9.14

Table 4. Times (in seconds) of ideal picture settings with Method parameter 1

Quality Factor	Conversion Time	Server Time	Total Time
20	7.87	1.30	9.17
50	8.00	1.06	9.06
80	8.65	3.58	12.24
100	14.70	5.66	20.36

Table 5. Times (in seconds) of ideal picture settings with Method parameter 2

Below are averages from 19 separate runs of the tests described in the Method of Testing section. Note that the time to convert the image is similar for method parameters 0 and 1 but increases significantly for method parameter 2. It is also important to note that quality factor has a rather insignificant affect on time of conversion, but greatly affects the time it takes to communicate with the server.

Quality Factor	Conversion Time	Server Time	Total Time
20	6.03	1.94	7.97
50	5.98	2.07	8.04
80	6.03	2.15	8.18
100	6.49	4.00	10.50

Table 6. Times (in seconds) for different quality factors with method parameter set to 0

Quality Factor	Conversion Time	Server Time	Total Time
20	6.08	1.62	7.70
50	6.06	2.40	8.45
80	6.06	2.45	8.52
100	6.52	4.44	10.96

Table 7. Times (in seconds) for different quality factors with method parameter set to 1

Quality Factor	Conversion Time	Server Time	Total Time
20	7.93	2.45	10.38
50	8.12	2.60	10.73
80	8.86	2.97	11.84
100	14.15	6.21	20.35

Table 8. Times (in seconds) for different quality factors with method parameter set to 2

Besides time, image quality and file size were also taken into account when running the tests. Comparing the three method parameters at a quality factor of 80, you can see that image quality and file size are very similar in images 1 through 3. If you compare quality factors, however, there is a noticeable jump in quality from 20 to 100. The file size increases significantly, however, as seen in Tables 9 through 11.

Image 1. Method parameter 0, quality factor 80, 91KB file size

Image 2. Method parameter 1, quality factor 80, 91KB file size

Image 3. Method parameter 2, quality factor 80, 102KB file size

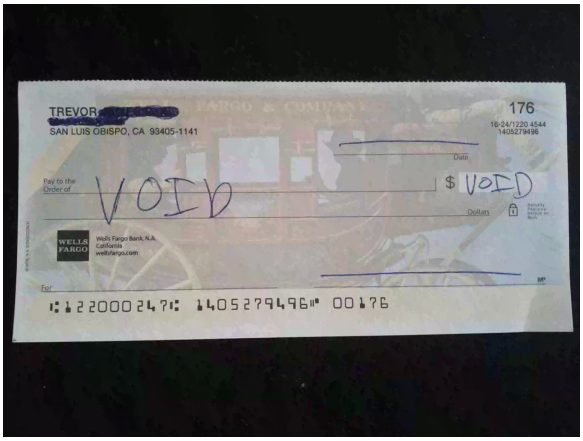

Image 4. Method parameter 1, quality factor 20, 60KB file size

Image 5. Method parameter 1, quality factor 100, 449KB file size

Quality Factor	Average File Size
20	55.32
50	68.11
80	93.31
100	388.58

Table 9. Average file size (bytes) for method parameter 0

Quality Factor	Average File Size
20	60.00
50	76.50
80	112.72
100	449.56

Table 10. Average file size (bytes) for method parameter 1

Quality Factor	Average File Size
20	57.35
50	72.45
80	103.85
100	381.85

Table 11. Average file size (bytes) for method parameter 2

Recommended Configuration

The recommended configuration for this application is setting the method parameter to 1 and quality factor to 80. Method parameter 2 provided a small increase in image quality but significantly increased the overall time required (~2.5 – 10 seconds increase from method parameter 1). Method parameters 0 and 1 were very similar in all areas, but method 1 provided slightly better images that returned correct MICR codes a higher percentage of the time with only a small increase in time. Quality factor 80 was chosen because although it took an average of 0.7 seconds longer to process compared to a quality factor of 20, it returned the correct MICR code a higher percentage of the time and produced noticeably better check images. Chart 1 shows method parameter 1 has slightly higher image size, but also produces higher quality images. A quality factor of 100 was omitted from the chart because the image size became significantly larger and made it harder to compare more realistic settings. Chart 2 shows the similar conversion times of method parameters 0 and 1 compared to method parameter 2.

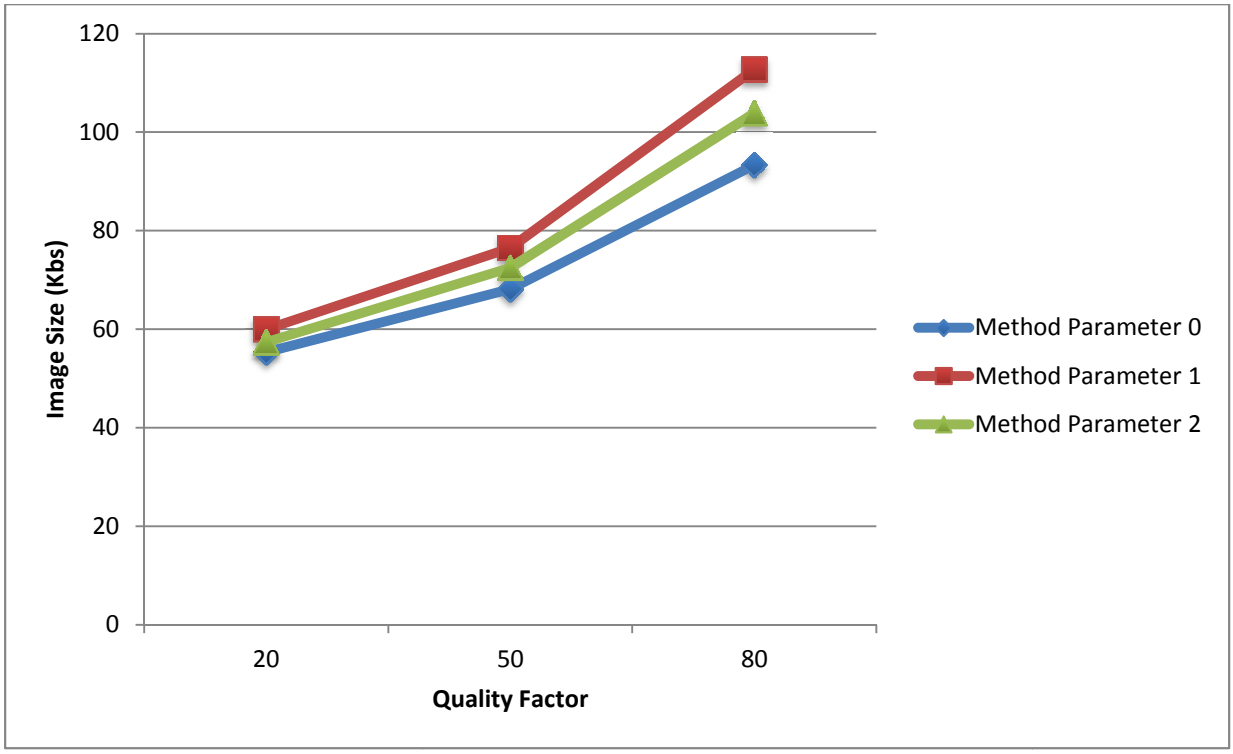

Chart 1. Image file size vs. quality factor. Quality factor of 100 omitted.

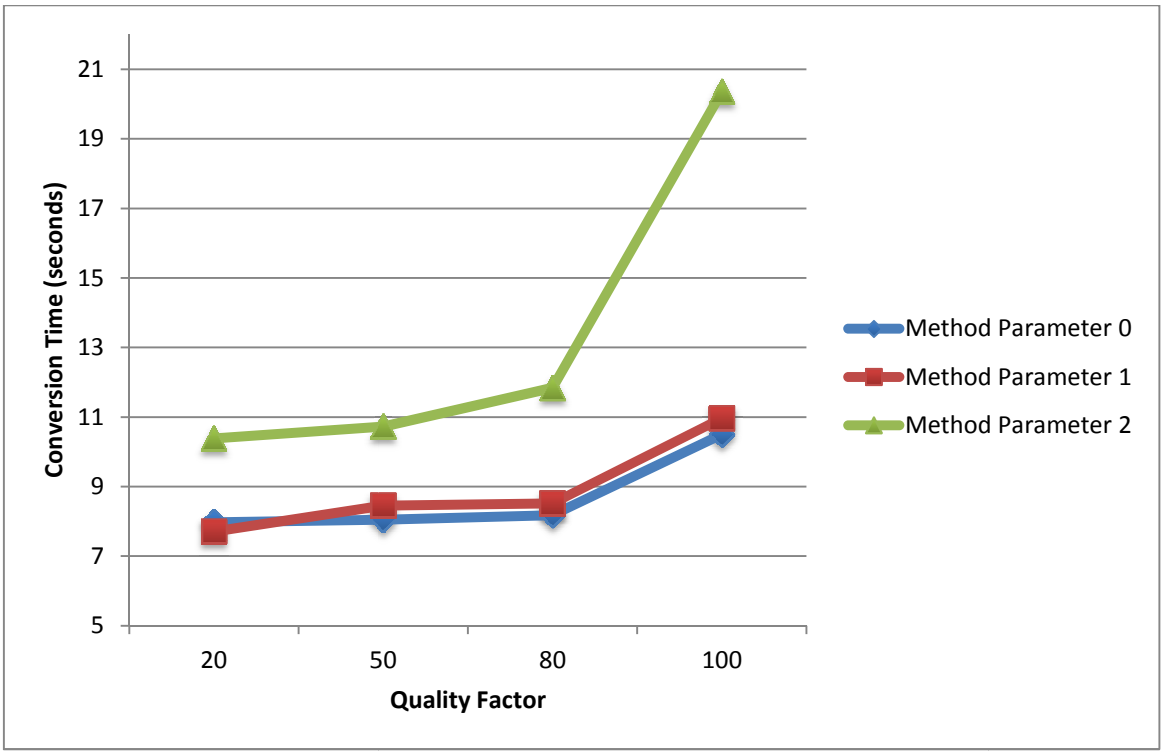

Chart 2. Conversion Time vs. Quality Factor

Sources of Error

There are some sources of error in my method of tests for this application. The most prominent is the number and variety of tests run. If a more thorough test is desired, each test case should be run many more times. There are also several more possibilities of settings the user could take a picture of a check in. Different surfaces such as carpet, colored wood, and lined paper could have also been tested. Pictures taken at different angles or in different lighting (besides just flash on or off inside) are also possibilities. It is also worth noting that no tripod was used to take images so there is some variance in exact distance form the check and angle in taking the photographs. The tests run for this project were an attempt to accurately portray functionality in a reasonable amount of tests without testing every possible scenario.

VI. CONCLUSION

Difficulties

This project experienced several difficulties that had to be confronted for the project to be successful. The biggest struggle was getting the WebP libraries to successfully convert the image. The hardest part of getting the WebP libraries to work is the lack of documentation and example code provided by Google and on forums such as stackoverflow.com. The README included with the WebP source code is one of the only sources of example code I could find to use as reference. Another struggle came from a bug in the Android.mk file included in the source files. This file did not link certain source files to properly build the conversion library and led to multiple issues that were difficult to debug.

Trying to figure out exactly what type of Bitmap to pass to the WebP libraries was also a significant hurdle. The documentation did not provide detailed information on the type of Bitmap that must be sent to the library so trial and error was the main method of figuring out how to manipulate the Bitmap.

One final difficulty came from working with an external company's server application to convert the image, which I had little control over. Partway through the project the server stopped scanning check images for an unknown reason. After multiple reinstalls, the server was never properly restored to its original state on my personal machine. The Company eventually set up the server on their end and the project could continue, but this took a few weeks to sort out.

Future Plans

Although the application is fully functional, there are aspects of the application that can potentially be improved upon in the future. The most important is optimizing the WebP conversion process. One potential improvement could be to convert the image to greyscale when it is taken before converting it to WebP. The greyscale image would most likely be smaller in size and could potentially cut down on overall time, especially in transit time between the server and application. Currently, the converted image is saved to the SD card immediately after conversion and loaded in the Android Java code and sent to the server from there. This was mainly done to monitor converted images during development. Sending the image without saving it as a file would probably cut down the overall time.

An original desire for the project was to overlay a frame to the camera preview to guide the user in aligning the check while taking a picture. To implement this a custom camera class must be created rather than using the native camera application on the device. For the sake of simplicity and reliability, the native camera class was chosen for this application, but an overlaid frame would most likely lead to a higher percentage of images returning the correct MICR code.

Before the application is published or given to clients, several UI improvements should be made. The main thing to change would be the home screen. Currently the home screen displays a message to the user and provides four buttons, three of which are used for testing or experimentation. Only one button provides the core functionality. Displaying the image and MICR response could also be done in a more attractive manner.

Wrap-up

Overall I feel as though the project was a success. An application has been produced which meets almost all of the Company's original desires. The application also functions very consistently if used correctly. The biggest source of error comes from how the user takes the picture on the camera of the Android device. If the image is centered on a black background the MICR code will be successfully scanned almost every time. Once the user starts taking pictures against off colored backgrounds or from different angles and distances, the server has trouble reading the check image. The application can only help guide the user into taking better pictures, and to make scanning the MICR code more reliable, improvements to the Company's server must be made. Thus, from the applications standpoint, I feel proud to pass off the application to the Company with hopes that they will take the base I have provided and eventually distribute this software to their clients to use regularly.

BIBLIOGRAPHY

[1] Love, Dylan. "COMSCORE: Android Has 47% Of The Mobile Market". *Business Insider.* Web. 30 Dec. 2011. <http://articles.businessinsider.com/2011-12-30/tech/30571342_1_android-platform-smartphone-windows-phone>.

[2] "What is Android?" *Android Developers.* Web. 5 Mar. 2012. <http://developer.android.com/guide/basics/what-is-android.html>.

[3] "JNI Types". *The Java Native Interface Programmer's Guide and Specification.* Web. 5 Jun. 2011. <http://java.sun.com/docs/books/jni/html/types.html>.

[4] "JNI Functions". *Java Native Interface Specification.* Web. 14 Apr. 2011. <http://docs.oracle.com/javase/1.4.2/docs/guide/jni/spec/functions.html>.

[5] "WebP: A new image format for the Web". *WebP Google Code Site.* Web. 11 Mar. 2012. <http://code.google.com/speed/webp/>.

[6] "WebP". *Wikipedia, The Free Encyclopedia.* Web. 3 Mar. 2012. <http://en.wikipedia.org/wiki/WebP>.

[7] "Platform Versions". Android Developers. Web. 23 Apr. 2012. <http://developer.android.com/resources/dashboard/platform-versions.html>.

Appendix A

```
jint Java_com_tbliss_android_seniorproject_webpconv_WebPConv_doConvJniGraphics2( JNIEnv* env
                                                                                 jobject javaThis,
                                                                                  jobject jbitmap
                                                                                jfloat jqualityFactor,
                                                                                   jstring  filename)
{
        int ret = 0;
        float cqualityFactor;
        int stride, width, height;
        void* pixels; // pointer to address of bitmap
        int outputSize = 0;
        AndroidBitmapInfo bitmapInfo;
        uint8_t* outputPointer;
        uint8_t* cbitmapPointer;
        WebPConfig config;
        WebPMemoryWriter wrt;
        size_t dataWritten = 0;
        int bytesWritten = 0;
        FILE* fileout = NULL;
        const char* fname       = (*env)->GetStringUTFChars(env, filename, NULL);

        // Get Bitmap info (height/width/stride)
        if ((ret = AndroidBitmap_getInfo(env, jbitmap, &bitmapInfo)) < 0){
                LOGV("Could not get Bitmap info. error=%d", ret);
                return 0;
        }
        width = (int)bitmapInfo.width;
        height = (int)bitmapInfo.height;
        stride = (int)bitmapInfo.stride; // width * 4

        // Lock Bitmap pixels
        if ((ret = AndroidBitmap_lockPixels(env, jbitmap, &pixels)) < 0) {
                LOGV("AndroidBitmap_lockPixels() failed ! error=%d", ret);
                 return 0;
        }

        cbitmapPointer = (uint8_t*)pixels;
        outputSize = stride * height;
        LOGV("output_size: %d", outputSize);
        outputPointer = (uint8_t*) malloc(outputSize);
        cqualityFactor = (float)jqualityFactor;

        // Setup a config
        if (!WebPConfigPreset(&config, WEBP_PRESET_PICTURE, cqualityFactor)) {
                LOGV("WebPConfigPreset failed");
```

```
        return 0;   // version error
}

// ... additional tuning
config.method = 1;
LOGV("config.method = %d", config.method);

if (WebPValidateConfig(&config) != 1) {
        LOGV("Error with config");
}

// Setup the input data
WebPPicture pic;
if (!WebPPictureInit(&pic)) {
        LOGV("WebPPictureInit failed");
        return 0;  // version error
}

pic.width = width;
pic.height = height;

if (!WebPPictureImportRGBA(&pic, cbitmapPointer, stride)) {
        LOGV("WebPPictureImportRGB failed");
        return 0;
}

// Set up a byte-output write method. WebPMemoryWriter, for instance.
pic.writer = MyMemoryWriter;
pic.custom_ptr = &wrt;

//InitMemoryWriter(&wrt);
wrt.mem = &outputPointer;
wrt.size = &dataWritten;
wrt.max_size = outputSize;

// Compress!
ret = WebPEncode(&config, &pic);   // ok = 0 => error occurred!
if (!ret) {
        LOGV("ret == 0, WebPEncode fail");
}

WebPPictureFree(&pic);  // must be called independently of the 'ok' result.

// Write to phone
fileout = fopen(fname, "wb");
if(!fileout){
        LOGV("cannot open output file %s", fname);
        return 0;
```

```
        }
        bytesWritten = fwrite(outputPointer, 1, dataWritten, fileout);
        LOGV("bytesWritten: %d", bytesWritten);

        // Unlock pixels of Bitmap
        fclose(fileout);
        AndroidBitmap_unlockPixels(env, jbitmap);
        free(outputPointer);
        (*env)->ReleaseStringUTFChars(env, filename, fname);

        return dataWritten;
}


/**
 * Method to write converted picture to pointer
 */
int MyMemoryWriter(const uint8_t* data, size_t data_size, const WebPPicture* const picture) {
  WebPMemoryWriter* const w = (WebPMemoryWriter*)picture->custom_ptr;
  size_t next_size;
  if (w == NULL) {
    return 1;
  }
  next_size = (*w->size) + data_size;
  if (next_size > w->max_size) {
    uint8_t* new_mem;
    size_t next_max_size = w->max_size * 2;
    if (next_max_size < next_size) next_max_size = next_size;
    if (next_max_size < 8192) next_max_size = 8192;
    new_mem = (uint8_t*)malloc(next_max_size);
    if (new_mem == NULL) {
      return 0;
    }
    if ((*w->size) > 0) {
      memcpy(new_mem, *w->mem, *w->size);
    }
    free(*w->mem);
    *w->mem = new_mem;
    w->max_size = next_max_size;
  }
  if (data_size) {
    memcpy((*w->mem) + (*w->size), data, data_size);
    *w->size += data_size;
  }
  return 1;
}
```