ENHANCED DETECTION OF LINEARLY DISTORTED QUICK RESPONSE (QR) CODES IN SMART PHONES

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Manav Singhal

Prof. Yi Shang, Advisor

April 2015

**ACKNOWLEDGEMENTS**

It gives me great pleasure to acknowledge the continuous support and guidance of my advisor Prof. Yi Shang throughout the development of my master’s project. His scholarly advice, meticulous scrutiny and patient encouragement have helped me immensely in development of this project. I would also like to thank the members of Wireless Sensor Network Lab for insightful advice and assistance.

I would like to thank Dr. Tim Trull and Dr. William Harrison for being a part of my defense committee and for their invaluable suggestions and comments on this project.

Finally I express my deepest gratitude goes to my family, who has provided me with unconditional love and support throughout the period of my studies at Mizzou. I would like to thank my parents for raising me to who I am. Their advice has always been the best guidance in my life.

**Table of Contents**

1. Introduction ……………………………………………………………………………….. 5
   1. Problem Statement and Objective ……………………………………………………. 7
2. Background and Related Work …………………………………………………………… 10
   1. QR Code History ……………………………………………………………………... 10
   2. QR Code Reader Apps – Google Play ……………………………………………….. 11
   3. QR Code – Structure and Characteristics ……………………………………………. 12
      1. Structure ………………………………………………………………………. 12
      2. Characteristics ………………………………………………………………... 13
   4. Traditional Steps for Detection ……………………………………………………… 14
   5. QR Code Detection – Previous Approaches ………………………………………… 18
3. Algorithm Design ………………………………………………………............................ 21
4. Implementation and System Design ……………………………………………………… 26
   1. Development Tools and SDK ………………………………………………………... 26
   2. App Specifications …………………………………………………………………… 26
   3. User Interface and Manual …………………………………………………………… 27
5. Experimental Results and Analysis ………………………………………………………. 31
6. Conclusion ………………………………………………………………………………... 34
7. References ………………………………………………………………………………… 35
8. Appendix ………………………………………………………………………………….. 36

**Abstract**

With the advancement in time, the need to store more and more data in a very concise space has reached to its maximum, and hence 2-Dimension codes e.g. Quick Response Code (QR code) were invented, which have relatively high scanning speeds, more data storage capacity, and a smaller size. QR Codes are small patterns printed onto a surface (advertisements, posters etc.) and their intent is only to help you get data from a printed medium to a digital medium.

A lot of work has been done previously to decode the QR codes on mobile phones by using the alignment patterns and finder patterns. But, the traditional approaches do not work well with linearly distorted QR codes and for certain rotations with mobile camera. In this project, a new fourth corner point detection method that is critical step in any QR code detection has been implemented using edge detection. To evaluate the performance of the approach, *QR Scanner* app has been developed for Android mobile platform to scan linearly distorted QR codes efficiently. A comparison study with few other popular QR code reader apps on Google Play has also been done to test the QR code detection rates on a sample set of 32 QR codes with various distortions and rotations. The results show that *QR Scanner* app provides ~75% more detection and better scanning rate compared to other apps.

1. **Introduction**

A barcode is an optical machine-readable representation of data, which shows data about the object to which it is attached. Quick Response (QR) Code is a two-dimensional symbol that was invented in 1994 by Denso, one of major Toyota group companies, and approved as an ISO international standard (ISO/IEC18004) in June 2000 [1]. Barcodes have found varieties of applications in different areas, for example libraries and information centers. In developed countries barcode patterns have become a familiar symbol for public due to their appearance in all daily life products.

2D Barcode readers are used to decode the data stored in the QR codes, but a major constraint with them is their high cost that reduces their field of applicability. In recent years, advanced technology has succeeded in continuously producing smaller yet smarter devices. The trend has contributed to leading our societies to the world of ubiquitous computing, where connectivity is always available unnoticed. Now mobile phones can implement many new kinds of applications such as taking photos and movie shooting by using embedded camera devices. So mobile phones with embedded camera devices can be used to recognize and read the QR code.

QR codes have come a long way since their creation, having first been developed to help track parts in the manufacturing process of vehicles. Today they have a number of purposes, including transport ticketing, entertainment, commercial tracking, and product labeling/marketing, just to name a few. You can find QR codes being used to send audiences to a website for browsing, to bookmark a webpage, to initiate phone calls, send short messages, send emails, generate links to web URL’s, start chats with users, connect to WI-FI networks, access information, get coupons, view videos, purchase items, process orders, advertise products, etc. Here are some statistics for QR code usage by various companies.

1. 1. 18% of retailers believe that the ability to scan QR codes, compare products and pricing is having a significant impact on their business [*KPMG July 2013 Reference*] [2]
2. 60.5% of marketers have adopted scannable QR codes or tags as a mobile marketing tactic [*Chief Marketer, 2013 Reference*] [2]
3. 42.4% of multi-channel merchants used QR codes in 2013 [*Multichannel Merchants June 2013Reference*] [2]







**Figure 1** – Usage and application of QR codes at various places

* 1. **Problem Statement and Objective**

The QR codes were initially discovered to improve data storing capacity for industrial applications only. However at later time, when these were implemented into mobile phones supporting built-in cameras to scan, with applications to decode data, these QR codes can be used as portable databases, thereby allowing users to access information anytime, anywhere, irrespective of their mobility and network connectivity.

To retrieve the data or information stored in the QR code using mobile camera efficiently is little challenging since it depends on lot of factors such as alignment of camera position and QR code. A lot of work has been done previously [*3*][4][5] to decode the QR codes using the alignment patterns and finder pattern. The traditional approach follows the concept of determining the alignment pattern that uses the basic property of QR code.

So, we tried popular QR code reader apps such as *QR Code Reader [6]*, *QR Droid Code Scanner [7]* and *Barcode Scanner [8]* live on Google Play and tested them on our sample set of linearly distorted QR codes. Few examples of linearly distorted QR codes are –

(A) (B)

**(C) (D)**

Figure 2 (A - D) – Sample set of Linearly Distorted QR Codes

The current popular QR code reader apps were not able to scan and read these linearly distorted QR codes. This is the biggest problem reading QR codes using mobile camera since it can have different alignments the way the user is scanning the code.

According to previous research works [3] [4] [5], it has been determined that recognition of QR code in mobile phone is little difficult due to reasons like: high noise, non-uniform illumination and most importantly linear distortion. It is very difficult to robustly extract accurate features such as edges and patterns from the QR code images taken by a camera phone.

Hence, our problem statement and objective is defined as –

*“Given a linearly distorted input QR code image, the problem is to* *efficiently estimate the fourth corner point leveraging finder and alignment patterns of QR code or other image processing techniques.* *The processing has to be optimized enough to run on mobile platforms such as Android OS”*

This is the real problem since a little distortion can throw pixels enough to make the QR code unreadable. Thus, the aim of this project work lies to develop scanner app determining fourth corner point using edge detection method to decode the data from linearly distorted QR codes efficiently.

Due to the ubiquitous nature of Android enabled devices and due to the open source nature of the Android SDK (Software Development Kit) [9], Android Operating System was considered as a platform to develop and deploy the QR Scanner application in this project.

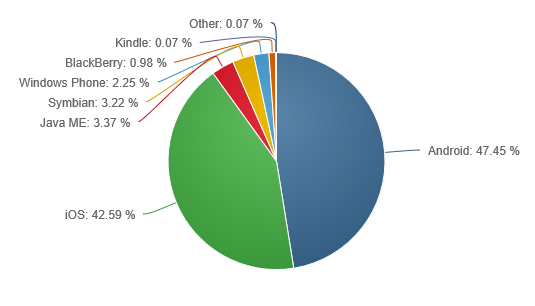


Figure 3 – Global Mobile/Tablet Operating System Market Share (February 2015)

1. **Background and Related Work**

**2.1 QR Code - History**

Barcodes are used widely because barcode code technology and processing provide a fast and accurate tool to enter data without keyboard data entry. The stages of development of QR code [10] can be depicted through a time graph as in the following Figure 4.

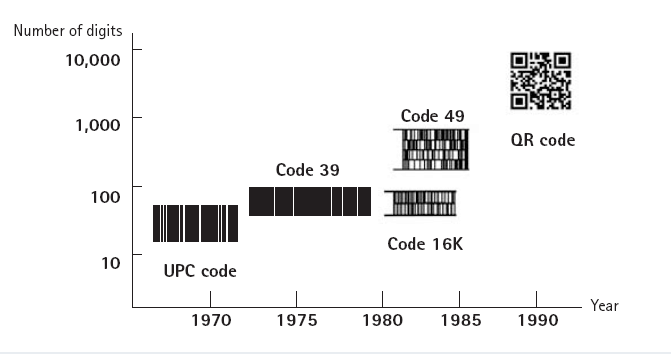


Figure 4 – History of Symbols

The research for storing more data in barcodes led to the development of 2D barcodes that can store large amount of data in a small area to support information distribution and detection without accessing a database.

Compared with 1D barcodes which hold varies limited information data, 2D barcodes has a much larger capacity to hold more information data. A QR code can hold up to7089 digits, 4296 letters, and 2953 binary data [10]. Selecting and using 2D barcodes must consider the following factors:

a) The Application usage,

b) Standard,

c) Implementation,

d) The data need to encode in barcodes and

e) How you wish to print the barcode?

**2.2 QR Code Reader Apps – Google Play**

These are the popular QR Code reader apps available on Google Play that we tested for linearly distorted QR codes and compared our ‘QR Scanner’ app results.

1. **QR Code Reader**

It is the most popular app for QR Decoder on Google Play under category ‘Tools’ [6]. It has been rated 4 stars out of 5 stars based on reviews/feedback by the people.

1. **QR Droid Code Scanner**

It is another popular app under category Productivity on Google Play [7]. It has been rated 4.2 stars out of 5 stars based on reviews/feedback.

1. **Barcode Scanner**

This app is by Zxing team for scanning and reading Barcodes and QR codes both under category ‘Shopping’ on Google Play [8]. It has been rated 4.1 stars out of 5 stars based on users’ reviews/feedback.

We installed and tested these apps on Android [9] smart phone (Samsung Galaxy S III Mini) and tried to scan some sample linearly distorted QR codes (described in appendix) with these apps individually. The detailed scanning results are mentioned in Section V. But, in nutshell none of them was able to read our set of linearly distorted QR codes.

**2.3 QR Code – Structure and Characteristics**

**2.3.1 Structure**

QR Code is a matrix type symbol with a cell structure arranged in a square. It consists of the functionality patterns for making reading easy and the data area where the data is stored. QR Code has finder patterns, alignment patterns, timing patterns, and a quiet zone. The structure of QR code is shown in Figure 5. [11]

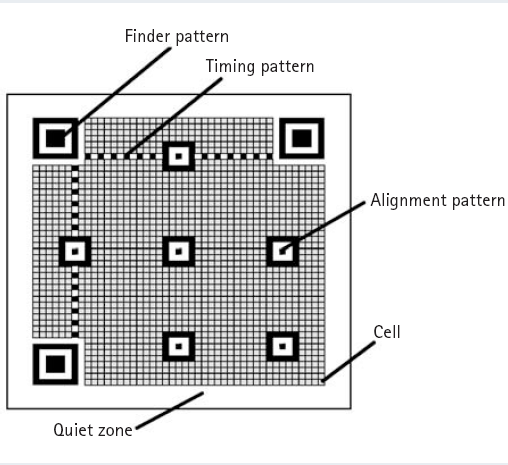


Figure 5 – QR Code Structure

1. **Finder Pattern-** Finder patterns are used for detecting the QR Code in scanning area. This pattern is at the three corners of a symbol. So, the position, the size, and the angle of the symbol can be detected. Also with the help of this pattern QR code can be detected in all directions (360°).
2. **Alignment Pattern-** Alignment pattern is helpful in correcting the distortion of the QR Code. The central coordinate of the alignment pattern is identified to correct the distortion of the symbol. For this purpose, a black isolated cell is placed in the alignment pattern to make it easier to detect the central coordinate of the alignment pattern.
3. **Timing Pattern-** The timing pattern is used for identifying the central coordinate of each cell in the QR Code with black and white patterns arranged alternately. It is used for correcting the central coordinate of the data cell when the symbol is distorted or when there is an error for the cell pitch. It is arranged in both vertical and horizontal directions.
4. **Quiet Zone-** Quiet zone is a marginal space necessary for reading the QR Code. This quiet zone makes it easier to have the symbol detected from among the image read by the CCD sensor. Four or more cells are necessary for the quiet zone.
5. **Data Area-** The QR Code data will be stored (encoded) into the data area. The grey part in Figure 4 represents the data area. The data is encoded into the binary numbers of ‘0’ and ‘1’ based on the encoding rule. The binary numbers of ‘0’ and ‘1’ are then converted into black and white cells. The data area has codes for error correction functionality.

**2.3.2 Characteristics**

Additional to the characteristics for two-dimensional symbols such as large volume data (7,089 numerical characters at maximum), high-density recording (approx. 100 times higher in density than linear symbols), and high-speed reading, QR Code has other superiority in both performance and functionalities aspects.

1. **All-Direction (360**°**) High-Speed Reading**

QR Code has finder patterns for notifying the position of the symbol arranged in three of its corners to enable high-speed reading in all directions (360°). The ratio between black and white among the scan line that runs through the finder patterns is always 1:1:3:1:1 when seen from any direction among the 360° surrounding it. [10][11]

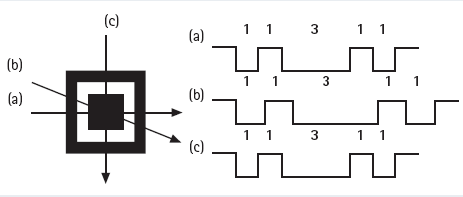


Figure 6– Finder Pattern

1. **Confidentiality of the Code**

By making the relationship between the character type and the stored data unique for a special usage, QR Code can be easily encrypted. Unless the conversion table between the character type and the stored data is deciphered, no one will be able to read the QR Code.

**2.4 Traditional Steps for QR Code Detection**

The traditional steps for any QR code detection are explained in this sub section -

1. **Gray Conversion**

QR Code symbol is captured by mobile phone with camera, and images are captured in Y’UV format by most phones, but QR Code symbol is a set of dark and light pixels [11]. It is needless to deal with color information and the gray image calculated quickly with little space, so gray conversion is needed to do firstly. Before performing the binarization of image, a barcode area in image is extracted that can reduce computational power and increase the accurate rate of binarizing bar code.

1. **Binarization**

Binarization is a necessary step in recognition bar code, and it is used in most recognition algorithms. Selection of a proper binarization method is critical to the performance of barcode recognition system. In binarizing an image, a simple and popular method is threshold. Ohbuchi [4] proposed an improving global threshold method. First the image was divided into nine blocks. Then the gray histogram is calculated for each block, and the gray value was sorted. The middle value was choosing as the threshold of each block. Finally the smallest value of these middle values is the global threshold of this whole image. This method is existed two drawbacks:

1. It is based on the assumption that the barcode symbol must be in the center of the captured image otherwise it fails.
2. This method results in excessive segmentation.

Using a global threshold method, the resulting binary image will be very bad, if an image was in variable lighting conditions. In this case, a local threshold method performs better. The main problems with a local threshold method are hard to set a right window size, eliminate the block effect, and reduce the execution time. Among more than 20 threshold methods, concluded that Otsu’s [12] method is the best, which chooses the threshold that minimizes within-group variance.

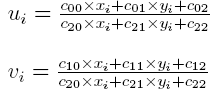
1. **Locating the finder patterns and Fourth Corner Point**

Locate the finder patterns is premise of getting version, orientation and distortion of QR Code. There are three identical position detection patterns located at three of the four corners of QR Code. Three dark-light dark squares are overlapped in every finder pattern, and the dark-light ratio is 1:1:3:1:1.

Determining the orientation of symbol after getting the location of finder patterns, QR Code is readable from any direction from 360 degree. This is the traditional approach that is used to find the three finder patterns present in the QR code image.

1. **Performing inverse perspective transformation**

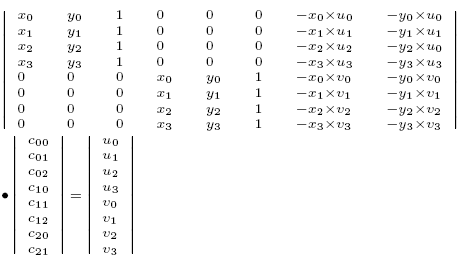
Generally, the input image has a deformed shape because of being captured from the embedded camera device. We thus use the inverse perspective transformation to normalize the code shape. The equation of calculating coefficients of perspective transformation, which maps vertexes (*xi, yi*) to vertexes (*ui, vi*) (i=1, 2, 3, 4): [11]

****

Equation (2)

Equation (1)

Coefficients are calculated by solving linear system:



Where, *cij* are matrix coefficients, *c*22 = 1.

As shown in equation below [11], let *f* (*x, y*) represent the Y’UV value of a pixel in the position of (x, y) in a new image, *g* (*u, v*) represent the Y’UV value of a pixel in the position of (u, v) in a origin image. The new image is mapped from the origin image, and then we can get the Y’UV value of a pixel by following equation:

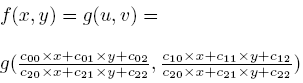


Figure 7, depicts [11] how inverse perspective transformation is done on a single pixel.

****

**Figure 7 -** Inverse perspective transformation

1. **Decoding**

The binary image is converted to a 2D bit matrix, and then mapping between bit 0 and 1 to data is used to extract data.

**2.5 QR Code Detection – Previous Approaches**

We studied various approaches for QR code detection that have been defined and used previously. One of most popular approach is Zxing Library [13]. Zxing library is available for both barcodes and QR codes on Android Operating system.

Zxing follows generic steps as described in previous section at all steps except Fourth Point Detection. The ‘Fourth point’ in Zxing approach is computed through geometrical calculations:

X4 = X2- X1 + X3 and Y4 = Y2-Y1+ Y4

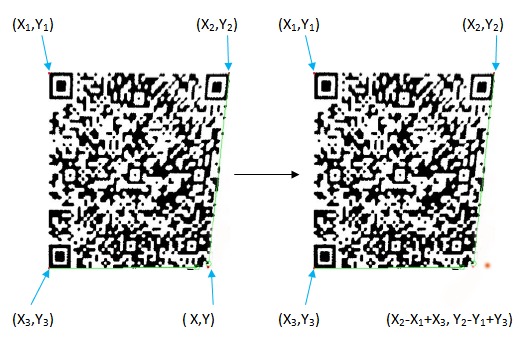


Figure 8 – Zxing Approach

The problem with Zxing [13] approach is that it assumes the QR code will be scanned with all edges parallel to the X and Y-axis. It does not consider the real use case. In this case, decoding is maximal when the QR code is aligned along X-axis and Y-axis with rotations around z-axis at 0, 90,180 or 270 degrees. The main issue lies with z-axis rotations at other angles like 45, 135, 225 degrees.

Another approach was given by Ohbuchi et al. [*4*] who presented an algorithm capable of doing real-time recognition of two-dimensional codes on mobile platforms. But the drawback of the method is that it has been hand tailored i.e. especially designed for one certain hardware device. Also, the algorithm depends on an assumption that the point in the middle of the screen is located in the code. This assumption is not realistic in usual environment. So the algorithm is difficult to be used in 2D bar code decoding.

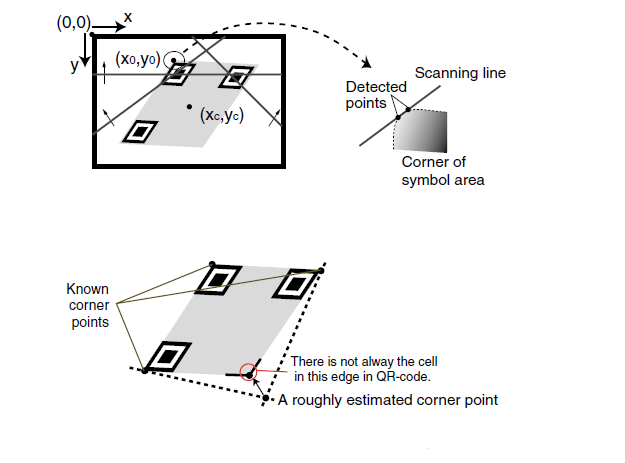


Figure 9– Ohbuchi Algorithm for estimating fourth corner point

The algorithm described by Chen and Yang e.t. al [*3*] didn't utilize the methods such as edge detection / line detection. It used alignment patterns to adaptively sample the QR code in terms of regions that greatly improved the recognition rate. Their algorithm was only simulated on PC and not truly tested in embedding system, so further work is required for optimization of their program codes, and embedding it into mobile terminals, and developing correlative research. Hence, we are not able to test their algorithm and compare their results with ours. Also, the authors have written their code is not optimized for mobile terminals, so we assume that it may take more time.

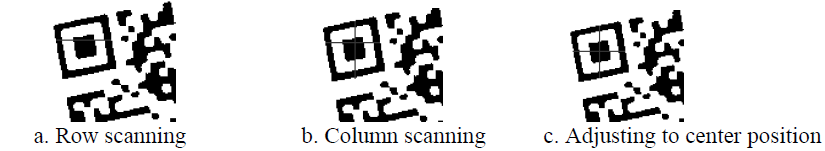


Figure 10 – Chen and Yang Approach for estimating fourth corner point

1. **Algorithm Design**

QR code with its large properties can be used in many fields for data capture and data processing. Our approach is to enhance QR code detector and decoder on android platform in order to provide fast and efficient data capture facility for general users. In this section, we have discussed how our edge detection approach is helpful in reading linearly distorted QR codes on smart phones.

Once the QR code image is extracted from the image taken from the embedded camera on phone and the features of the QR code image have been identified, it is then sent to the traditional QR decoding algorithm as we discussed in previous section. Although this algorithm works well when the QR code image is not distorted or is not taken at a correct angle, but when there is some distortion then problem arises.

Our Method uses the traditional detector to locate the finder and alignment patterns based on their specifications of black and white cells. If the 3 finder patterns are found, then the sides of 3 finder patterns are used to find approximate bottom and right edges. Using these approximate bottom and right edges, we use our *Edge Detection* method to perform more accurate search for determining the real bottom and right edges. The intersection of these real edges is used as the bottom-right corner, i.e. fourth corner point.

In detection of QR codes, there is not only the problem for image rotation, but perspective distortion, wherein the code surface is not parallel to that of the camera and then there is the problem of a warped barcode and distortion, which traditional approach does not account for.

The traditional QR Code detection requires three finder patterns at three corners of the image. Unfortunately this is not enough to determine perspective distortion correctly. This is a real problem since a little distortion can easily throw off enough pixels to make the code unreadable.

Algorithm 1

Objective *To get the transformed image for given input image*

Input*Given raw input image (pixels matrix)*

Output *Transformed image*

CREATE-TRANSFORM (InputImage)

{

FourthCornerPoint = FIND-CORNER (InputImage, TopLeftPoint, TopRightPoint, BottomLeftPoint)

{}

TransformedImage = PERSPECTIVE-TRANSFORM (SourcePoint1, SourcePoint2, SourcePoint3,

SourcePoint4, TargetPoint1, TargetPoint2,

TargetPoint3, TargetPoint4)

**return** TransformedImage;

}

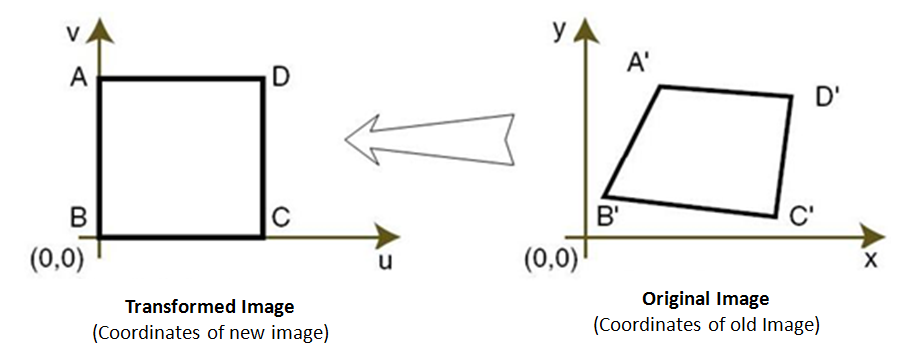


Figure 11 – Inverse Perspective Transformation

Algorithm 1.1

Objective *To determine the bottom-right corner of a QR code*

Input *Input Image and 3 Finder pattern points*

Output *Fourth corner point of QR Code*

FIND-CORNER (*InputImage*, *TopLeft*, *TopRight*, B*ottomLeft*)

{

BottomRight = GUESS-LAST-PATTERN (*TopLeft*, *TopRight*, B*ottomLeft*) {

BottomRight [x] = BottomLeft [x] + (TopRight [x] – TopLeft [x]);

BottomRight [y] = BottomLeft [y] + (TopRight [y] – TopLeft [y]);

}

BottomEstimatedEdge = FIND-PATTERN-EDGE (*InputImage*, *BottomLeft*, *BottomRight*) {

// finds 2 points on the bottom estimated edge, specifying the line

}

RightEstimatedEdge = FIND-PATTERN-EDGE (*InputImage*, *TopRight*, *BottomRight*) {

// finds 2 points on the right estimated edge, specifying the line

}

BottomRealEdge = EDGE-DETECTOR.FIND-LINE (*InputImage*, *BottomEstimated [0],*

*BottomEstimated [1]*) {

// finds bottom real edge, specifying the line

}

RightRealEdge = EDGE-DETECTOR.FIND-LINE (*InputImage*, *RightEstimated [0]*, *RightEstimated [1]*) {

// finds right real edge, specifying the line

}

IntersectionPoint = EDGE-DETECTOR.INTERSECTION (*BottomRealEdge*, *RightRealEdge*){

// intersection of bottom and right real edge is fourth-corner point

}

**return** IntersectionPoint;

}

Algorithms 1.2

Objective *To determine real bottom and right edge of given QR code image*

Input *Input image and Estimated Bottom/Right edge points*

Output *Real Bottom and Right edges*

Given a line and maximum deviation from the line, we find the line going through the most edge points. First, we find the edge points (all points that could possibly be on the edge, within the specified deviation). At each of the line's end points for each possible line, we draw orthogonal line up to 'deviation' pixels in each direction. Each pair of points on these two lines (one on each) forms a possible line. Each of these lines is part of test under consideration.

For each probable line, we iterate through all the edge points. The distance from the edge point to the line is used to calculate the "cost" for the line. “The line with the best *cost* is returned”.

FIND-LINE (*InputImage*, *Point1*, *Point2*)

{

BestPoint1 = null, BestPoint2 = null;

PointCount = 0;

Deviation = 0; // Moving in Anti-Clockwise direction

**for** each *Deviation* greater than -45 degree {

EndPoint = Point2 + Deviation;

EdgePoints = FIND-EDGE-POINTS (*InputImage*, *Point1*, *EndPoint*);

**if** (EdgePoints.length > PointCount) {

PointCount = EdgePoints.length;

BestPoint1 = EdgePoints [0];

BestPoint2 = EdgePoints [length - 1];

}

Deviation--;

}

Deviation = 0; // Moving in Clockwise direction

**for** each *Deviation* less than 45 degree {

EndPoint = Point2 + Deviation;

EdgePoints = FIND-EDGE-POINTS (*InputImage*, *Point1*, *EndPoint*);

**if** (EdgePoints.length > PointCount) {

PointCount = EdgePoints.length;

BestPoint1 = EdgePoints [0];

BestPoint2 = EdgePoints [length - 1];

}

Deviation++;

}

**return** BestPoint1 and BestPoint2;

}

Algorithms 1.3

Objective *To find all points that could be part of an edge*

Input *Input Image, start Point and end Point*

Output *List of all edge points*

FIND-EDGE-POINTS(*InputImage, StartPoint, EndPoint*) {

EdgePoints = null;

DiagnolLine = GET-LINE (*TopLeft*, *BottomEstimatedRight*);

DiagnolCutPoint = INTERSECTION (*TopLeft*, *BottomEstimatedRight*, *StartPoint*, *EndPoint*);

Point = StartPoint;

**while** each *Point* less than *DiagnolCutPoint* {

**if** (*Point* instance of *InputImage*){

EdgePoints.add (*Point*);

}

Point ++;

}

**return** EdgePoints;

}

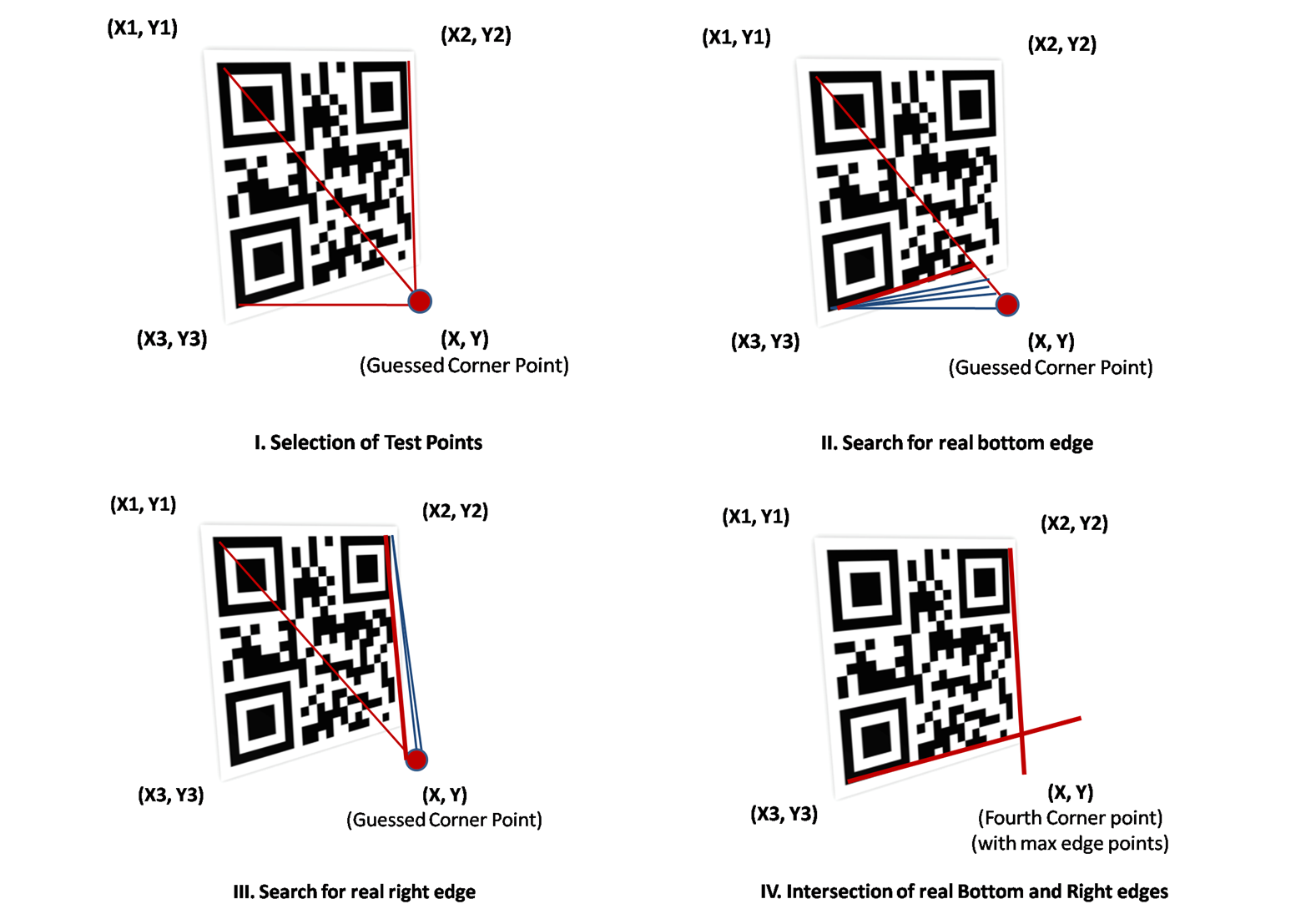


Figure 12 – Visual Representation of Fourth Corner-point Estimation Algorithm Design

1. **Implementation and System Design**

Based on the edge detection algorithm described in previous section, we modified the Zxing source code (i.e. open source) to include our approach for detecting the fourth corner point and reading the content of QR code successfully.

* 1. **Development Tools and SDK**

We used the following tools for developing QR Scanner Android app.

* Integrated Development Environment (IDE) Eclipse Indigo [14]
* Programming Tool Java
* External Library Zxing QR Code Library
* Development Kit Android SDK [9]
* Testing PhonesSamsung Galaxy S III Mini

(Android 4.1)

**4.2 QR Scanner App - Specifications**

This is the specification of our QR Scanner Android application.

1. android:version Android 1.6 and above
2. android:minSdkVersion Android 2.1 (API Level 7)
3. android:targetSdkVersion Android 3.0 (API Level 11)
4. User Permissionsandroid.hardware.camera

android.permission.CAMERA

android.permission.INTERNET

android.permission.ACCESS\_WIFI\_STATE

android.permission.ACCESS\_NETWORK\_STATE

1. External Library Zxing QR Code Library Source Code [13]
   1. **QR Scanner App – User Interface and Manual**

This sub-section contains the user interface information about QR Scanner App developed using modifying Zxing source code with our proposed fourth corner point estimation algorithm.

1. **Splash Screen**

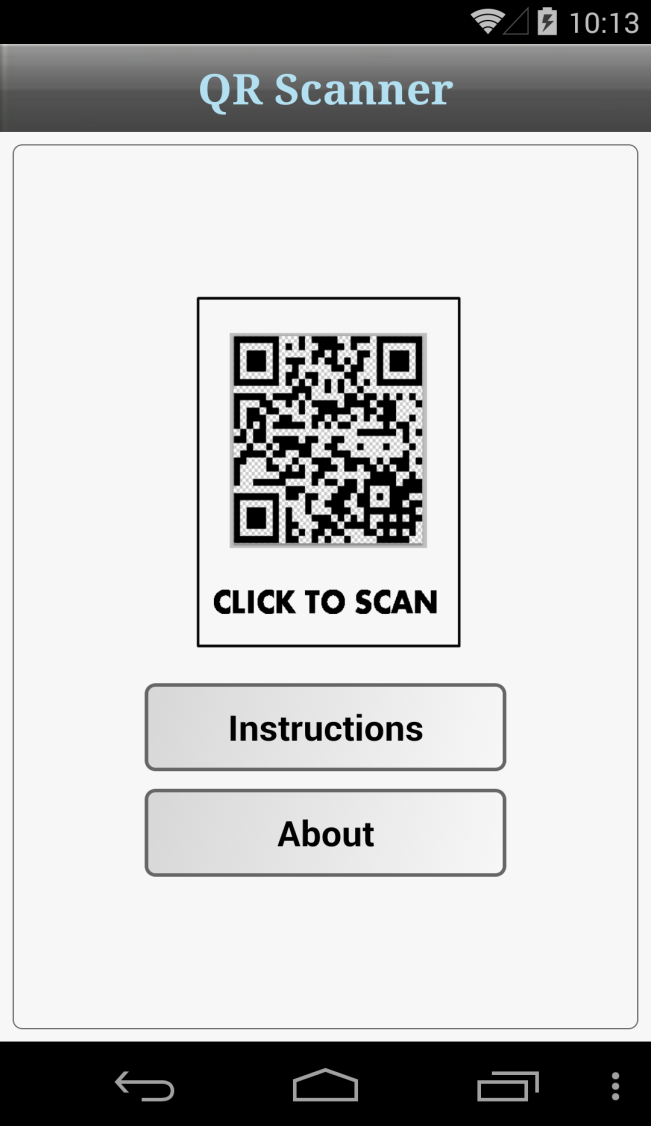
This is the splash screen that gets launched once user opens the application. It goes away within 3 seconds (by default timer).



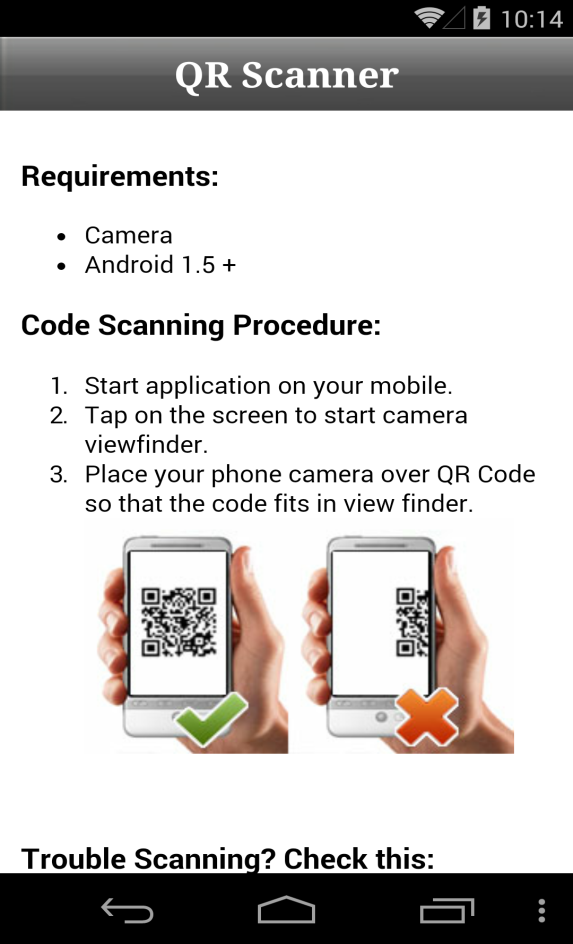
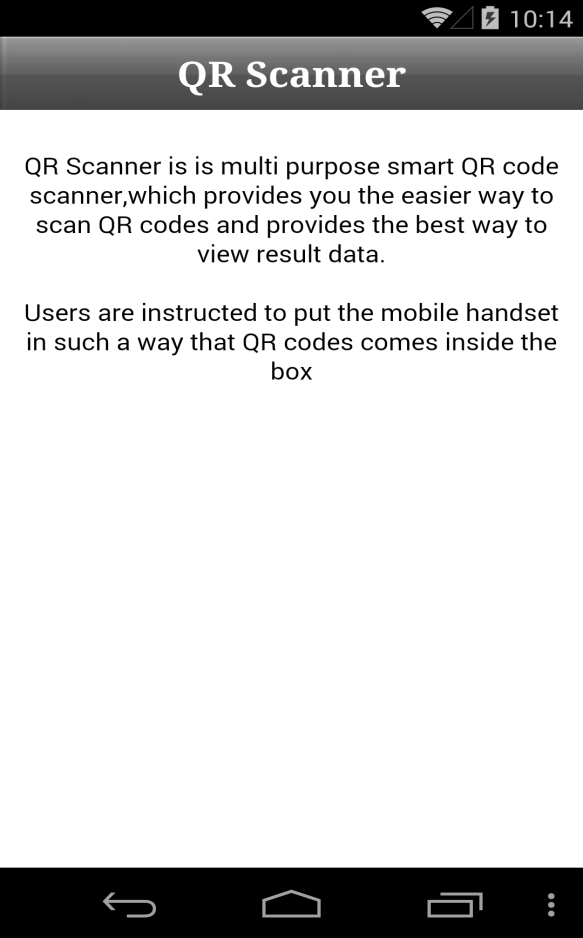
1. **Home Screen**

This is the home screen of the application after splash screen. Here the user has 3 options –

* *Scan QR Code* - To start scanning QR codes
* *Instructions* - To read the instructions on how to use the application
* *About us* - Displays information about the application



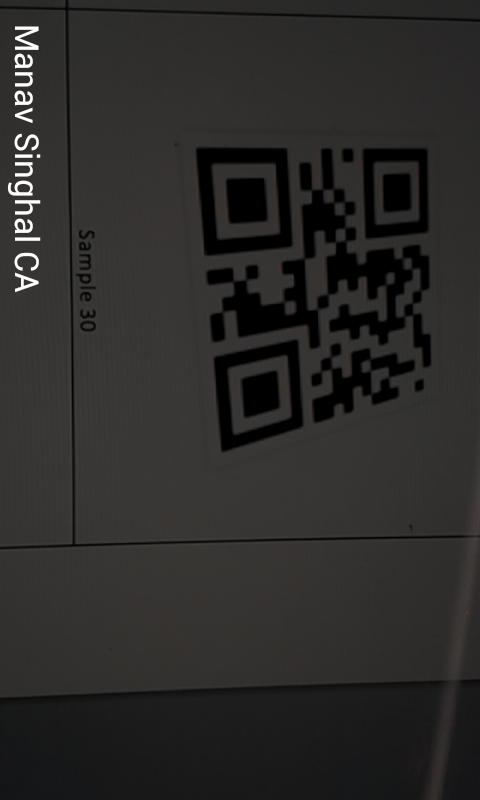
1. **Instructions and About Screen**

** **

1. **Scanner Screen**

Once the user touches ‘Click to Scan’ button on home screen, the camera is launched within the app and the user can scan the QR code.

Note, the camera has to be atleast 12 inches distance from the QR code image you are scanning and the QR code must fit within blue box in the camera.

****

**5. Experimental Results and Analysis**

Here are the scanning results for the QR code samples using the available apps (described in Section 2.2) and our implemented app ‘QR Scanner’ implementing fourth corner point detection based on edge-detection approach. The sample sets of QR codes that we tested our app are mentioned in appendix at the end.

For each sample image, we performed the scanning tests multiple times (at least 10 times) for each application (QR Code Reader, QR Droid Code Scanner, Barcode Scanner and proposed QR Scanner). It has been mentioned in the instructions while using QR Scanner app that the camera has to be at least 12 inches away from the QR code the user is scanning and the code must fit within the blue box in the camera.

The scanning times mentioned in table below for each application are the averages of scanning times taken over a number of experiments. It helped us to reduce the errors and give deterministic results. Since QR Scanner app involves image processing, sometimes if the smart phone is resource-constrained (i.e. many apps are running in background), the scanning time might increase by 1-2 seconds due to processing of image.

While we tested the QR code scanning time for each application, we cleared out the memory of smart phone before starting the testing in order to get best accurate results.

Table 1 - Experimental Detailed Scanning Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **QR Code**  **Sample #** | **QR Code Reader**  **App**  (Average Scanning Time) | **QR Droid Code Scanner**  **App**  (Average Scanning Time) | **Barcode Scanner**  **App**  (Average Scanning Time) | **Proposed**  **QR Scanner**  **App**  (Average Scanning Time) |
| **Sample 1** | **No** | **No** | **No** | **Yes, 2-3 sec** |
| **Sample 2** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 3** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 4** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 5** | **No** | **No** | **No** | **Yes, 2-3 sec** |
| **Sample 6** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 7** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 8** | **No** | **No** | **No** | **Yes, 1-2 sec** |
| **Sample 9** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** |
| **Sample 10** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 4-5 sec** | **Yes, 1-2 sec** |
| **Sample 11** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** |
| **Sample 12** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 2-3 sec** | **Yes, 1-2 sec** |
| **Sample 13** | **No** | **No** | **No** | **Yes, 4-5 seconds** |
| **Sample 14** | **No** | **No** | **No** | **No** |
| **Sample 15** | **Yes, 4-5 sec** | **No** | **Yes, 4 - 5 sec** | **Yes, 2-3 seconds** |
| **Sample 16** | **No** | **No** | **No** | **No** |
| **Sample 17** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 18** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 19** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 20** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 21** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **No** | **Yes, 1-2 sec** |
| **Sample 22** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** | **Yes, 1-2 sec** |
| **Sample 23** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 24** | **Yes, 4-5 sec** | **No** | **Yes, 2-3 sec** | **Yes, 1-3 sec** |
| **Sample 25** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 26** | **Yes, 2 sec** | **Yes, 7 - 8 sec** | **No** | **Yes, 1-3 sec** |
| **Sample 27** | **No** | **No** | **Yes, 1-2 sec** | **Yes, 1-2 sec** |
| **Sample 28** | **No** | **No** | **Yes, 1-2 sec** | **Yes, 1-2 sec** |
| **Sample 29** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 30** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 31** | **No** | **No** | **No** | **Yes, 1-3 sec** |
| **Sample 32** | **No** | **No** | **No** | **Yes, 1-2 sec** |

Our result is shown in Table above where we see that our approach does a significant improvement in the speed of detection. In 19 cases out of 32 sample cases, none of the current apps were able to read codes and in 6 cases, 1 or 2 apps were able to read QR codes but our app read all 25 codes efficiently within ~1 sec. This gives the accuracy of *approximately* ***~75%***more detection rate than traditional apps. Also, our approach is able to detect highly distorted images not only correctly but also at considerable fast speed than Zxing approach.

Table 2 – Scanning Result Analysis for different Apps

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sample Count | QR Code Reader App | QR Droid Code Scanner App | Barcode Scanner App | Proposed QR Scanner App |
| # 25 | **No** | **No** | **No** | **Yes** |
| # 5 | **Yes** | **Yes** | **Yes** | **Yes** |
| # 2 | **No** | **No** | **No** | **No** |

It can be seen clear that our app works for linearly distorted QR codes detection at a much faster rate than currently available popular apps.

The results have shown that the algorithm implemented in QR Scanner App to detect QR Code performs better than the existing approaches.

1. **Conclusion**

QR codes can be used on various mobile device operating systems e.g. Blackberry, Symbian OS, iOS and Android. There are many business use-cases where QR codes can be utilized for marketing and promotions. Few of them are –

* Product Info and Coupons
* Social Media
* Real Estate and Loyalty Programs
* Mobile Payments
* Real Time Info and Paperless Tickets
* Movie Trailers

The estimated number of QR codes outside Japan is expected to be 532 million in 2015 [17] and Android mobile platform is more popular than other mobile platforms. Hence, we implemented our edge detection method to develop Android app ‘QR Scanner’.

Our approach used for estimating the fourth corner point using finder pattern and alignment pattern is an improvement from the traditional approach, which usually fails with linearly distorted QR codes and different rotations of codes. *QR Scanner* app can read the codes even when the image taken is not parallel to the surface and has various z-axis rotations. This is a real problem since a little distortion can easily throw off enough pixels to make the code unreadable. But, with *QR Scanner* app it is very much feasible to read such codes and also, this method can be easily extended to other mobile platforms.

1. **References**
2. Soon, Tan Jin. "QR code." *Synthesis Journal* 2008 (2008): 59-78.
3. Data Dive: QR Codes, http://www.marketingcharts.com/online/data-dive-qr-codes-29525/
4. Chen, Weibing, Gaobo Yang, and Ganglin Zhang. "A Simple and Efficient Image Pre-processing for QR Decoder." *2nd International Conference on Electronic & Mechanical Engineering and Information Technology*. Atlantis Press, 2012.
5. Ohbuchi, Eisaku, Hiroshi Hanaizumi, and Lim Ah Hock. "Barcode readers using the camera device in mobile phones." *Cyberworlds, 2004 International Conference on*. IEEE, 2004.
6. Scheuermann, Constantin, et al. "Evaluation of barcode decoding performance using zxing library." *Proceedings of the Second Workshop on Smart Mobile Applications, SmartApps*. 2012.
7. QR Code Reader Android App, <https://play.google.com/store/apps/details?id=me.scan.android.client>
8. QR Droid Code Scanner, <https://play.google.com/store/apps/details?id=la.droid.qr>
9. Barcode Scanner, <https://play.google.com/store/apps/details?id=com.google.zxing.client.android>
10. Android Software Development Kit (SDK), <http://developer.android.com/sdk/index.html>
11. QR Code History, <http://www.qrcode.com/en/about>
12. T. Pavlidis, J. Swartz, and Y.P. Wang, “Information Encoding with Two-Dimensional Bar Codes”, IEEE Computer, pp. 18-28, Volume 25, June 1992.
13. Otsu Threshold Method, <http://en.wikipedia.org/wiki/Otsu's_method>
14. Zxing – QR Code Decoding Library, <https://github.com/zxing/zxing>
15. Eclipse Integrated Development Environment (IDE), <https://eclipse.org>
16. QR Code Usage Statistics, <http://visual.ly/qr-code-usage-statistics>
17. Jerry Zeyu Gao, Lekshmi Prakash, and Rajini Jagatesan , “Understanding 2D-BarCode Technology and Applications in M-Commerce– Design and Implementation of A 2D Barcode Processing Solution.” Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International Volume: 2 Publication Year: 2007
18. Mobile/Tablet OS Market Share – Feb 2015, <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>

**Appendix**

|  |  |
| --- | --- |
| **Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_1:QR_Code_Sample_1_0.png**  Sample 1 | **Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_1:QR_Code_Sample_1_90.png**  Sample 2 |
| **Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_1:QR_Code_Sample_1_180.png**  Sample 3 | **Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_1:QR_Code_Sample_1_270.png**  Sample 4 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_2:QR_Code_Sample_2_0.png  Sample 5 | **Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_2:QR_Code_Sample_2_90.png**  Sample 6 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_2:QR_Code_Sample_2_180.png  Sample 7 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_2:QR_Code_Sample_2_270.png  Sample 8 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_3:QR_Code_Sample_3_0.png  Sample 9 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_3:QR_Code_Sample_3_90.png  Sample 10 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_3:QR_Code_Sample_3_180.png  Sample 11 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_3:QR_Code_Sample_3_270.png  Sample 12 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_4:QR_Code_Sample_4_0.png  Sample 13 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_4:QR_Code_Sample_4_90.png  Sample 14 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_4:QR_Code_Sample_4_180.png  Sample 15 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_4:QR_Code_Sample_4_270.png  Sample 16 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_5:QR_Code_Sample_5_0.png  Sample 17 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_5:QR_Code_Sample_5_90.png  Sample 18 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_5:QR_Code_Sample_5_180.png  Sample 19 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_5:QR_Code_Sample_5_270.png  Sample 20 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_6:QR_Code_Sample_6_0.png  Sample 21 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_6:QR_Code_Sample_6_90.png  Sample 22 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_6:QR_Code_Sample_6_180.png  Sample 23 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_6:QR_Code_Sample_6_270.png  Sample 24 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_7:QR_Code_Sample_7_0.png  Sample 25 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_7:QR_Code_Sample_7_90.png  Sample 26 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_7:QR_Code_Sample_7_180.png  Sample 27 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_7:QR_Code_Sample_7_270.png  Sample 28 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_8:QR_Code_Sample_8_0.png  Sample 29 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_8:QR_Code_Sample_8_90.png  Sample 30 |
| Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_8:QR_Code_Sample_8_180.png  Sample 31 | Mac HD:Users:manav.singhal:Desktop:MS_Project_Progress_Update:sample_qr_codes:Sample_8:QR_Code_Sample_8_270.png  Sample 32 |