Extending the Storage Capacity And Noise Reduction of a Faster QR-Code

Kishor Datta Gupta
Department of Computer Science
Lamar University, Beaumont, USA
400 S M L King Jr Pkwy, Beaumont, TX 77705, United States of America
Tel. +1 409-880-7011
kgupta@lamar.edu

Md Manjurul Ahsan
Department of Industrial Engineering
Lamar University, Beaumont, USA
400 S M L King Jr Pkwy, Beaumont, TX 77705, United States of America
Tel. +1 409-880-7011
mahsan2@lamar.edu

Dr Stefan Andrei
Department of Computer Science
Lamar University, Beaumont, USA
400 S M L King Jr Pkwy, Beaumont, TX 77705, United States of America
Tel. +1 409-880-7011
s.andrei@lamar.edu

Abstract

Quick Response Code has been widely used in the automatic identification fields (Liu, Ju, & Mingjun, 2008). The present work illustrates an image processing system able to discover, split and decodes the most common 2D symbol used in bar code applications. The different symbol is processed by manipulating their similarities, to achieve an integrated computational structure (Ouaviani, Pavan, Bottazzi, Brunelli, Caselli, & Guerrero, 1999). There is not enough novel approach which could be effective for data transferring to alter various sizes, a little noisy or damaged and various lighting conditions of bar code image. We proposed a faster QR code which has more storage and can scan faster. The new QR code generation takes twice the time than normal QR code but it can also store double QR code data. It's scanning is as faster as other QR code scanning technique. we deducted the level of color percentage despite positions of color bits in that module, which increased its scanning speed and reduced noise error rate to less than 10 percent.

Keywords: Quick response code, recognition, Scanning technique.

1.Introduction

In 1994, A Japanese Hardware Company first introduced “Quick Response Code”, which is known as QR codes. Back then, it was designed to allow high-speed component (Furht, Borko ,2011). Currently, it is used not only for getting information of commercial products but also used for smartphone tagging. Beginning of computer age, it was necessary to have machine readable medium or automated data medium, by automated data medium we meant a form of something which could store data that only a computer can read. There are many forms available but here we will discuss only one of the optical medium. It is printed data matrix generated image on any substance. The first example of it is Barcode. The Barcode was influenced by Morse code, its primary use was for tagging each product (Figure 1).

Figure 1. A sample Linear Barcode

In starting all barcodes was 1D or known as linear, but still, there are many standards and many types of barcode. To increase data capacity and reduce the error rate, 2D barcode was introduced. The most popular one of them is QR code. But QR code also has the limitation of data capacity so researcher introduced High Capacity Color Barcode. However, the introduction of color in QR codes created new challenging problems. As consequence, the color barcode concept did not get too much popularity.

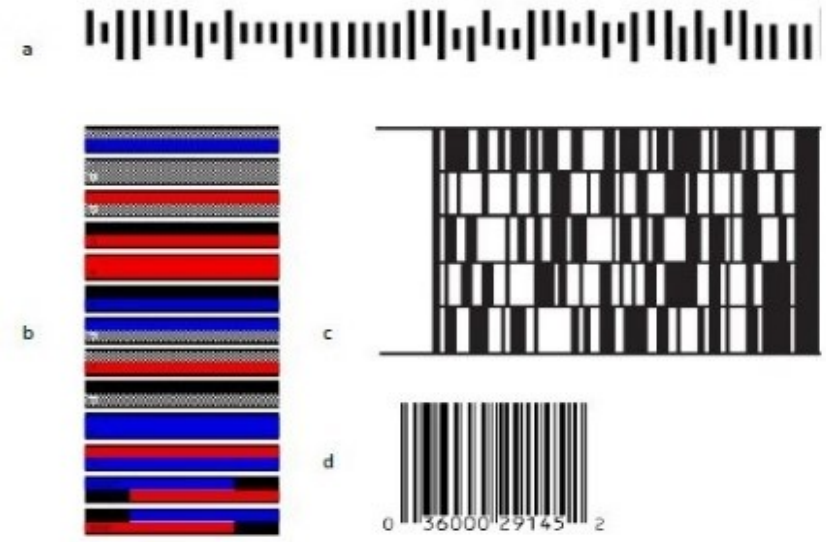


Figure 2. 1D barcode examples a) 2d postal barcode b) Colored railroad barcode c) code 49 barcode d) The Universal Product Code

The 1D or linear bar codes are mainly based on the width of each line by a predefined standard width. For the width based barcode an example is shown in Figure 2 (d).

Here, the big limitation is for more data barcode width size will get bigger. As we can see barcode to bitwise waveform transfer in Figure 3.

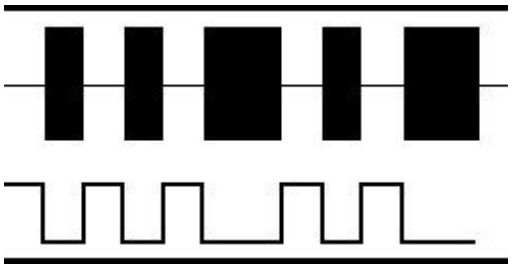


Figure 3. Barcode width read

For this reason, the linear code is used to only store small code words data, such as product number or country code. To store more data, 2D barcode was introduced, for example Figure 4 shows a 2D barcode. Advantages of 2D barcodes over linear barcode are not only storage of more information, but also reduced the error rate, the data encryption, the much less space, the possibility

to scan in all directions and printing in curve area.[5] This makes 2D barcodes more usable in all fields. When Microsoft introduced High Capacity Color code (Figure 4(d)) it comes with more storage and more features. However, it also faces some image processing issues. As other standard 2d codes are based on black and white, this does not depend that much on scanner quality or printed material quality. As a result, Microsoft discontinued its research on color codes. Several attempts to solve this problem taken by many other researchers notably by Homayoun Bagherinia and Robert Manduchi in 2011 (Bagherinia Homayoun and Roberto Manduchi, 2011).

Figure 4. 2D barcode examples a) MaxiCode b) CrontoSign c)ShotCode d)HCCB e)QRcode f)HCCBQ

2. Standard QR code:

To the human eye, all barcodes will look very similar. But if we look it more closely we will see some differences, although without most trained eye we cannot recover any meaningful data. This is because all data are generally encrypted. In linear barcodes, normally binary data or symbol for each letter of English alphabet and number are used. For 2D code, we normally directly use binary codes of respective data. Information in QRCodes can split up in 6 parts as shown in Figure 5. As like as: Quiet zone, Finder patterns, Alignment pattern, Timing pattern, Version information, Data cells. (Hartley, Richard, and Andrew, Zisserman,2003)

[5] "How QR codes (and other 2D barcodes) work." Explain that Stuff. July 02, 2016. Accessed June 17, 2017. http://www.explainthatstuff.com/how-data-matrix-codes-work.html

Figure 5. QRCode Data blocks information

3. Our Proposed QR Method

Based on our work on QR code scanning, we think there could be new QR code possible which able to store more data. Our idea is use half the area of black bit instead of full area. It has faster decoding and encoding speed.

3.1. Encoding Method

First, we will convert data to normal QR code, then the black bits we will make them one third instead of full, and place in the middle, then we prepare another QR code same way, rotate that in 90 degrees and place it on first QR code. And we should add a finder-bit so while decoding we get the info of decoding bit size, which is essential to start decoding method (Yang, Zhibo, Huanle Xu, Jianyuan Deng, Chen Change Loy, and Wing Cheong Lau, 2017).

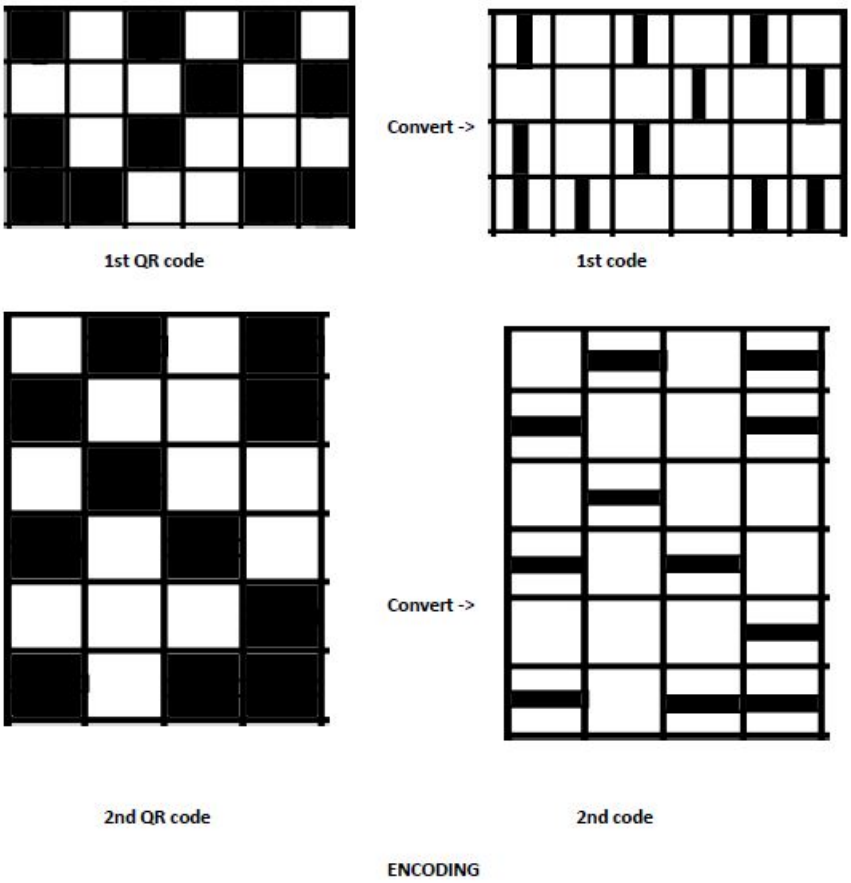


Figure 6. Encoding QR code step1

In our Experimental setup, we placed a black bit in top-left most corner. In Figure 6 and Figure 7 these encoding steps are explained.

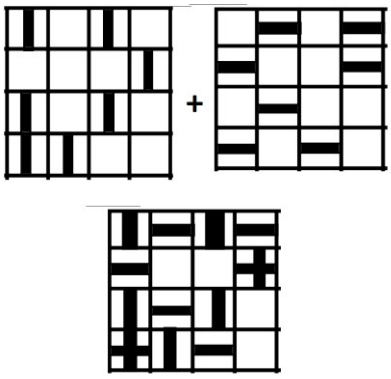


Figure 7. Encoding QR code Step 2

3.2. Procedural steps for encoding

Steps of encoding are as follows:

Step 1. Convert the data in Binarystring consisting only 1 and 0.
Step 2. Split the Binarystring in two part, part1 and part2
Step 3. Add ending bit 1 in the end of both part1 and part2
Step 4. Divide imagesize with the SQRT of Part2 data length. This will be the bitsize
Step 5. Place a Rectangle in BitSize at the top-left most corner in image.
Step 6. Take another two samesize bitmap.
Step 7. In bitmap1, Take each character from part1 and started to draw line if value which width is one third of the bitsize but length is equal to bit size. After reach at image end increase X position by bit-size and start again.
Step 8. In bitmap2, Take each character from part1 and started to draw line if value which width is one third of the bitsize but length is equal to bit size. After reach at image end increase X position by bit-size and start again.
Step 9. Merge bitmap1 and bitmap2 and get the image

The image we get from step 9 is our desired new QR code. Based on image size, our bit limit increase and decrease. If we use 1024 as image size, theocratically we will be able to keep 262144 bit data in our code, but Technically it betters to keep one fifth of that to avoid decoding issues.

3.3. Pseudocode for encoding

The pseudocode of the encoding process is explained below and details source code are available in appendix A.

pseudocode:

```
  var result = StringToBinary(InputText);
  var part1 = first half of result + "1";
  var part2 =  second half of result  + "1";
  var sqrt =sqrt(part2 data size);
  var bitsize =  imagesize/sqrt;
  var x =  sizeofbit / 3;
  var y = 0;
  foreach (char c in part1) {
   if(c==1) var bitmap1 =  draw line in x and y position with width of bitsize/3 and
lenght of bitsize;
     if(y>=imagesize)  {y = 0;
     x = x + sizeofbit;} }
    y = sizeofbit / 3;
    x =0;
   foreach (char c in part2) {
   if(c==1) var bitmap2 =  draw line in x and y position with width of bitsize/3 and
lenght of bitsize;
   if(y>=imagesize)  {x = 0;
     y= y+ sizeofbit;}}
  var QRcodeimage = new image (imagesize + bitsize,imagesize + bitsize);
  QRcodeimage draw rectengale at 0,0 point with bitsize as height and width;
  QRcodeimage =draw bitmap1 and bitmap2 from bitsize,bitsize location;
  QRcodeimage save at filelocation;
```

3.4. Implementation of Encoding

We implemented in visual studio. We use default c# image library for decoding purpose and after that, we save the image in jpeg format. In Figure 8, we paste a text from CNN news at our inputbox and in Figure 9 we converted it to binary, third image is our desired QR code.

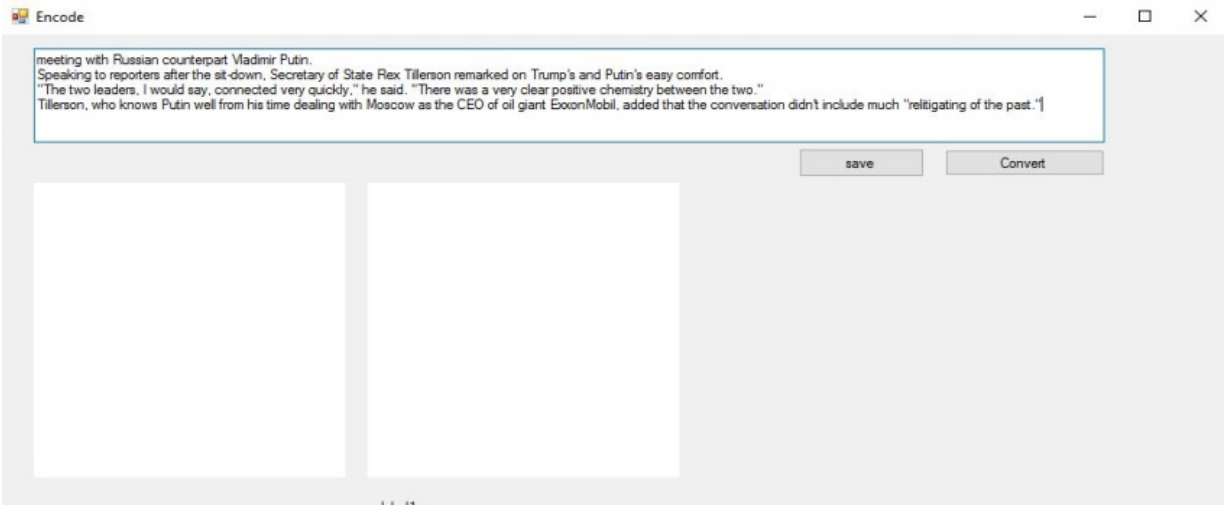


Figure 8. Encoding in visual studio for datainput.

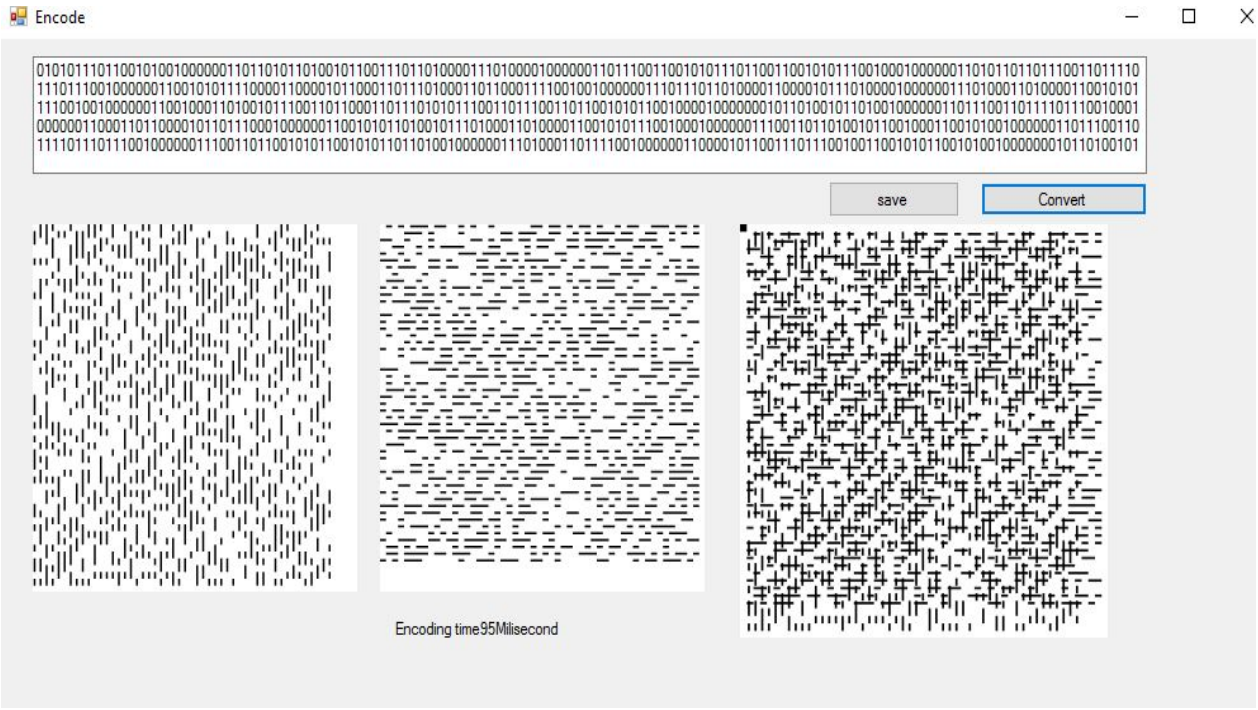


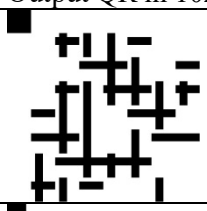
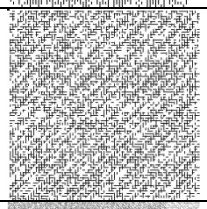
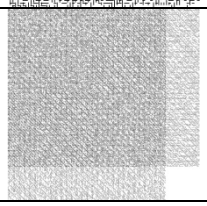
Figure 9. Generate QR code

We can see in figure 3, in topleft corner we placed bit size. This we will use as finder pattern in decoding steps. After that, we can save the image as QR code in the file directory. We used 1024 as image size. Add a rectangle for of size-bit add extra size, based on length of data bit. Time to encode normally done below 100ms for normally 300 to 500 words, largely based on data content, as for 0 we don't use draw functions.

4. Experimental Results of Encoding

We ran several tests on our application, and save the resulting image in our file directory. Also, we measure the processing time for encoding method. In Table 1, we showd some of the examples alongside results and processing time in milliseconds.

Table 1. Encoding Experimental result

Input Text	Output QR in 1024size	Process time in Milliseconds
"Hello World"		75
"Lamar University Kishor Datta Gupta L20421951"		73
Abstract of this paper		103
Page 1 of this paper		115
Article from"http://artssciences.lamar.edu/computer-science/news/lamar-university-virtual-map.html"		555

5. Analysis of Encoding

From the Table, we can see if data size increases our encoding speed also decrease. For more data, we will be needed to increase our image size. The Zbar and Zxing QRcode reader software need more time than our encoding method, mostly because they need to draw the rectangle and fill that while are drawing the only line which gives us the better performance as we are drawing on sorted image matrix.

Our time complexity is $T(N) = T(N/2) + T(N/2) \Rightarrow O(n) = N$

One of the issues in the Encoding process is rounding up the floating bit size to integer point, As we flooring the results, we have to waste some space. If data is big, then this waste of space gets more visible, In Figure 10 there is an example shown.

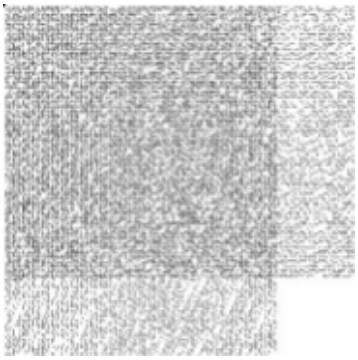


Figure 10. Waste of space in right bottom corner

In Figure 10 we can see due to floor the float value to nearest integer value creates a blank space in right bottom corner. For decoding purpose we mark last bit as black one, so after the last data, we have to keep all white. In future, we could use this space to store error correction bits. Another possibility is brought that in the center and use for curvature detection.

5.1. Decoding Method

We can apply bar code decoding system here that's why it will be much faster, and we will ignore black length less than the module size. This way it would be faster than other QR code, we have to scan horizontal and vertical and ignore lest than module size black color block as 1. As seen in Figure 11 and Figure 12.

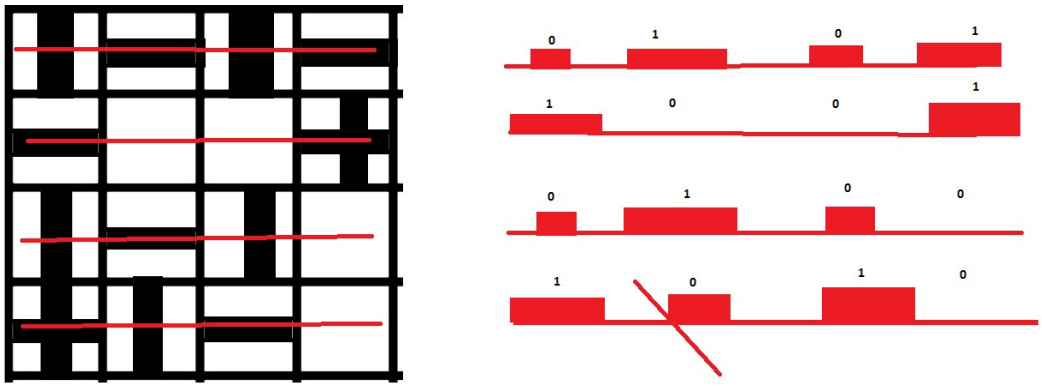


Figure 11. Decoding vertical scanning

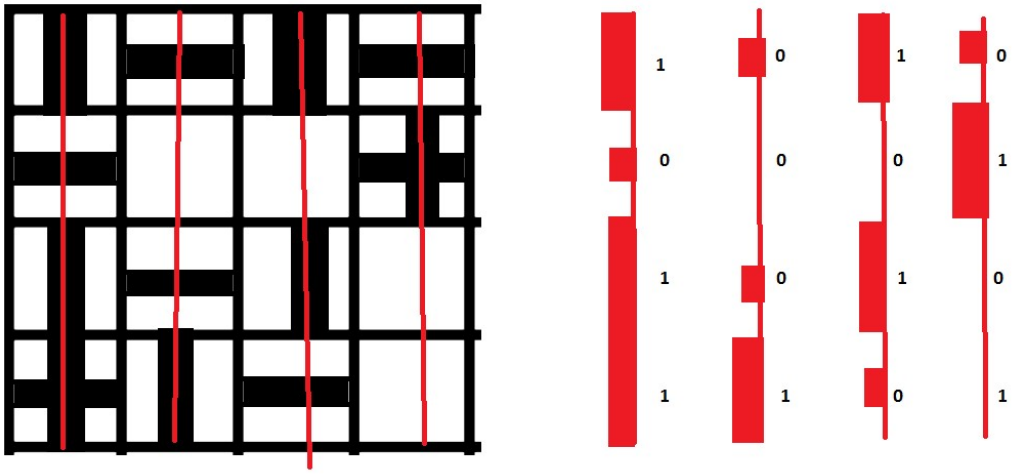


Figure 12. Decoding Horizontal scanning

5.2. Procedural steps for decoding

In decodings, steps can be divided into three parts. One gets the bit size from processed image, second is scan and get the data bits, third and last is decode the data in text, in below pseudocode step 1 to 4 is for getting the bit size and 5 to 8 is for scanning purpose.

Steps of decoding are

Step 1. Get the input image .
Step 2. Convert the image in greyscale.
Step 3. Convert input image in binary image. Using threshold value 128.
Step 4. Started from top left corner to width or height, check every pixel value. If get 255 , that pixel position represent the bitsize
Step 5. Start to go pixel by pixel horizontally and check blackbit are equal or atleast $3/4^{th}$ of bitsize or not. If yes we will add that as 1 bit . if not that will add as 0 bit.
Step 6. If it is the the most right bottom pixel we go step 7. Else after reach the end of image height, will increase position of X by bitsize and Go to Step 5.
Step 7. Start to go pixel by pixel vertically and check blackbit are equal or atleast $3/4^{th}$ of bitsize or not. If yes we will add that as 1 bit . if not that will add as 0 bit.
Step 8. If it is the the most right bottom pixel we go step 9. Else after reach the end of image width, will increase position of X by bitsize and Go to Step 7.
Step 9. Convert the data from binary to ASCII.

After step 9, we will get our resulted text in ASCII format. We have to remember before converting to ASCII we have to remove every bit after last 1. As that's the ending flag bit.

5.3. Pseudocode for decoding

The pseudocode of the Decoding process is explained below and details source code is available in Appendix B.

Pseducode for Sizebit

```
var image = QRcodeimage
var greyscaleimage = Convert QRcodeimage to Greyscale
Vat Bbinaryimage =Convert Greyscaleimage to binary using threshold 128;
for (int j = 1; j < img.Height; j++){
var pixel = GetPixelat 1,j point;
if (pixel == 255){bitsize = j; break;}
}
After getting the bit size we can start scanning the image file and decoding to binary.
```

Pseducode for Decoding

```
for (int i = bitsize +bitsize/3; i < img.Width;  i = i + bitsize; )  {int
ptb,ptw,bitsizecount = 0;
  for (int j = bitsize ; j < img.Height; j++) { bitsizecount++
                    int pixel = Bbinaryimage.GetPixelat 1,j point;
                    if (pixel == 255){  ptb++;ptw = 0; }
                    if (pixel == 0)  {ptb=0;ptw++; }
                    if (bitsize == bitsizecount) {
                        if (ptb >= ptw)  horijontal = horijontal + 0;
                        else horijontal = horijontal + 1;
                       ptb,ptw,bitsizecount = 0;}} }

 for (int j = bitsize   + bitsize / 3; j < img.Height;  j = j + bitsize;)  { int
ptb,ptw,bitsizecount = 0;
        for (int i = bitsize ; i < img.Height; i++) { bitsizecount++
                   int pixel = Bbinaryimage.GetPixelat 1,j point;
                   if (pixel == 255){  ptb++;ptw = 0; }
                    if (pixel == 0)  {ptb=0;ptw++; }
                    if (bitsize == bitsizecount) {
                        if (ptb > ptw)  vertical = vertical + 0; else vertical = vertical
+ 1;
                        ptb,ptw,bitsizecount = 0; }}  }
var binaryresult = horijontal+vertical;
Var result = binrayresult to ascii convertion
```

5.4. Implementation of Decoding

We use default Aforge image library for decoding purpose and after that, we use normal file reader to scan the image. In Figure 13, we load a QR code from our file directory and in Figure 14 we decoded it to binary and convert that binary in the text, Results are shown in two textboxes.

Figure 13. QR code load in application

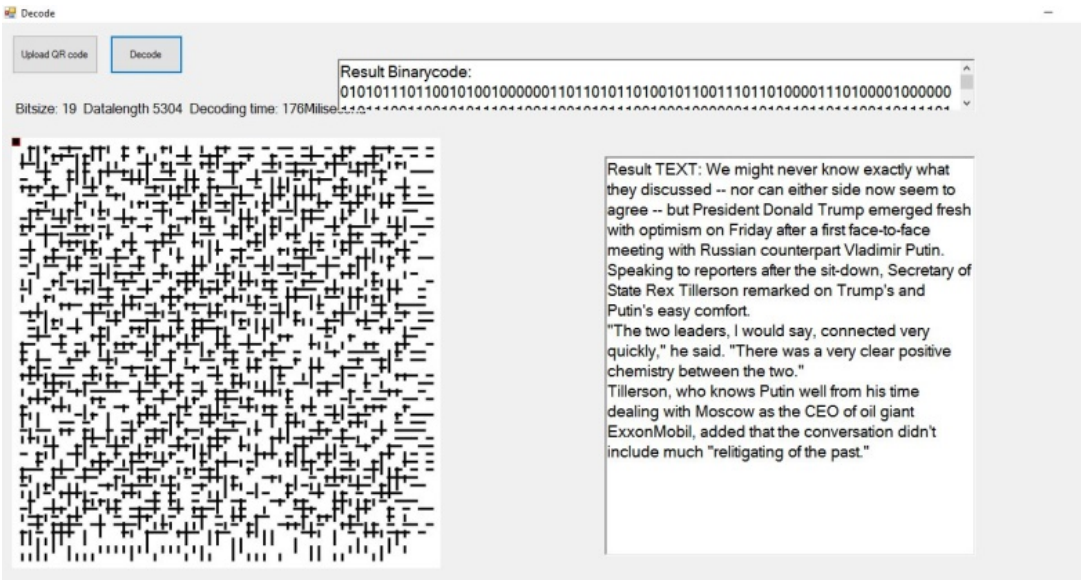


Figure 14. QR code decoded and result shown.

After image convert to binary, we start for determining the size bit. In Figure 15 at top left corner, sizebit is present. Using that sizebit we started to scan the image.

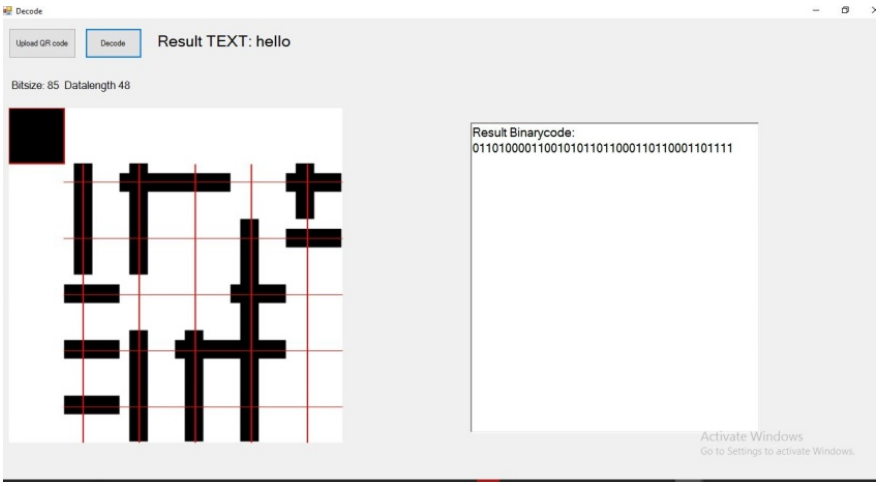


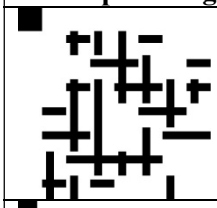
Figure 15. Decoding with grid line

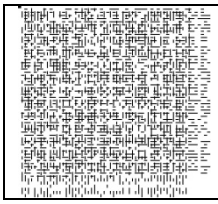
For better understanding, We draw redline while scanning in Figure 15 when we get the constant black line which length is almost equal to bit size, we will consider that as 1 otherwise, that's a 0. We also have to remember some digit does convert to binary ok, and it still does not support other than ASCII code.

6. Experimental Results of Decoding

We ran several decoding tests from the QR code which we encoded and saved in our file directory. We calculated the processing time for encoding method. In below Table, we showed some of the examples alongside results and processing time in milliseconds.

Table 16. Decoding Experimental result

Input Image	Output	Processing time ms	Data length
	"Hello world"	32	96
	"Lamar University"	26	136
	Thesis name	12	536
	Abstract of this thesis	210	8800

	Acknowledgment of this thesis	165	6424

7.Analysis

The decoding is much faster than encoding, and also compare to other QR code decoding systems. Its speed is faster as Linear bar code. Its time complexity depends on bitsize. If N = image size/bitsize . then time complexity is 2T(N) => O(N). Also as lines are sequential we do not have to worry about cross color modulation error. Also as we are using one-third area of the bit module, it has more error prone than other QR code scanning, that's why in this case our zone based error detection system will be helpful as it will work none the less the printer or printing materials faults.

7.1. Advantages of new QR code

There are several new advantages of our proposed QR code over currently most used QR codes, These are

- Our QR code can store almost double data than Universal QR code.
- Our QR code encoding and decoding time are faster than Universal QR code.
- Our QR code can detect curvature distortion issue better.
- Cross module color issue can be ignore .

As we are reduced the size of blackbits one third , it giving us the option to add more data than normal QR code. Also while encoding instead of draw a black rectengualr bit , we draw a single line. Also in a 2d matrix which is sorted. It makes the encoding speed faster . In decoding time, we use to scan in single line. So its take less time than othe QR code which read a full rectengular bit and determine its color to measure its black bit or white bit. While we are just chking for black line. This procedure also help us to avoid curve distortion and cross module distortion issue.

7.2. Limitations of New QR code

Our new QR code has some big limitations , these are

- For large data if image size small it will not work.
- If image size can not be divisible by bitsize , for large data size we will be lost line after read some data.
- This new QR code issue is more vulnarbel for light glare and illumination issue.

For solving, these issues we need to increase image size respect to data size, and we can always add garbage data to keep bit size a divisor for image size. In future, we can try to solve these issues.

8. Conclusion

In our new QR code method, we can also add the third and 4th layer of QR code by rotating 45 and 135 degrees respectively, that's why it can store 4X data than normal QR code. And if we apply HiQ color bits instead of binary bits, that could make this QR code a huge storage of database to use securely. Also, it needs to make applicable for camera work, especially for mobile camera. Another room of improvement is, adding curvature detection bits. If we could do this than it will be possible to make it user-friendly in consumer level.

References

Bagherinia, Homayoun, and Roberto Manduchi. *"A theory of color barcodes"*. In Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, pp. 806-813.

Furht, Borko, ed. Handbook of augmented reality. Springer Science & Business Media, 2011.

Hartley, Richard, and Andrew Zisserman. *„Multiple view geometry in computer vision”.* Cambridge University Press, 2003.

Liu, Yue, Ju Yang, and Mingjun Liu. (2008). *"Recognition of QR Code with mobile phones".In Control and Decision Conference, 2008.* CCDC 2008. Chinese, pp. 203-206. IEEE.

Ouaviani, E., A. Pavan, M. Bottazzi, E. Brunelli, F. Caselli, and M. Guerrero. (1999*)"A common image processing framework for 2D barcode reading.",*pp. 652-655.

Yang, Zhibo, Huanle Xu, Jianyuan Deng, Chen Change Loy, and Wing Cheong Lau. *"Robust and Fast Decoding of High-Capacity Color QR Codes for Mobile Applications".* arXiv preprint arXiv:1704.06447 (2017).

Ştefan ANDREI received his BSc in Computer Science (1994) and MSc in Computer Science (1995) from Alexandru Ioan Cuza University of Iasi, and PhD in Computer Science (2000) from Hamburg University. He was awarded with four competitive scholarships, as follows: the Singapore - MIT Alliance Computer Science Fellowship, 2002-2005, the World Bank Joint Japan Graduate Scholarship Program, 1998-2000, the TEMPUS Fellowship, 1998-1998, and DAAD Scholarship, German Government, 5/1997-7/1997. Dr. Andrei wrote over 100 publications published in prestigious journals and conference proceedings which have more than 200 non-self international citations in reputable publications. He has given invited talks at more than 20 reputable universities and industrial companies. He has been a Program Committee member or co-Chair of more than 40 international reputable conferences and a PI, co-PI, or Senior Personnel of more than 11 funded research grant proposals. He was promoted to ACM Senior Member in April 2013. Currently, he is an Associate Professor and Chair of the Department of Computer Science with Lamar University, Beaumont, TX, U.S.A. His research interests are in the areas of optimization techniques, verification, and scheduling analysis for multiprocessor platforms for real-time embedded systems, software engineering and translation systems.