Solving QR Code Distortions using a Recursive-based Backtracking Algorithm

Kishor Datta Gupta
Department of Computer Science
Lamar University, Beaumont
4400 S M L King Jr Pkwy, Beaumont, TX 77705, USA
Phone: +1 409-880-7011
kishordatta@lamar.edu

Stefan Andrei
Department of Computer Science
Lamar University, Beaumont
4400 S M L King Jr Pkwy, Beaumont, TX 77705, USA
Phone: +1 409-880-7011
stefan.andrei@lamar.edu

Md Manjurul Ahsan
Department of Industrial Engineering
Lamar University, Beaumont
4400 S M L King Jr Pkwy, Beaumont, TX 77705, USA
Phone: +1 409-880-7011
mahsan2@lamar.edu

Abstract

Quick Response (QR) codes are getting widely popular as the demand for mobile computing is increasing, too. However, a QR code has apparently a data limit problem when we consider designing color QR codes. Despite having color QR codes decoding from printed material, new problems appear due to different printer's color depth, paper quality, environmental effect, dust, light glare and other environmental effects. In our model, we proposed a new algorithm to find an area consisting of several bit positions. We have applied an area-based bit detection method based on the percentage of the color level instead of the general binary color bit detection system. This is independent of the printer and the printed substances effect. There is no need for the color correction palette. Based on our experimental results, our findings demonstrate that we can produce a better result without having any image filtering first.

Keywords: QR Code; Image Processing; Backtracking Algorithm Distorted Image; Uneven Light.

1. Introduction

In 1994, a Japanese hardware company first introduced "Quick Response Code", which is known as QR code. Back then, it was designed to allow high-speed components (Furhy, 2011). Currently, it is used not only for getting information of commercial products but also for smartphone tagging.

2. Distortions in QR code

To the human eye, all barcodes look very similar. Despite that, if we analyze them 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, ordinary binary data or symbol for each letter of English alphabet and number are used. For the 2D code, we normally directly use binary codes of respective data.

2.1. Standard QR Code Encoding and Decoding

Information in QR Codes can split up in to 6 parts (example in Figure 1).



Figure 1. The six zones of a QRCode data (Ohbuchi, Eisaku & Lim, 2004)

Each of these six zones described as follows (Woodford, 2018):

1) *Quiet zone*: An empty white border which makes it possible to separate the code from other printed information (for example, on a dirty envelope, among the black and white print of a newspaper, or on smudged product packaging).

2) *Finder patterns*: Large black and white squares in three of the corners which make it easy to confirm that this is a QR code (and not, say an Aztec code). Since there are only three of them, it is instantly obvious which way up the code is and which angle it is indicating at (unless the code is partly obscured or damaged in some way).

3) *Alignment pattern*: The alignment pattern assures that the code can be decoded even if it is distorted (viewed at an angle, printed on a curved surface, and so on).

4) *Timing pattern*: This runs horizontally and vertically between the three finder patterns and consists of alternate black and white squares. The timing pattern composes it easy to analyze the individual data cells within a QR code and is exclusively useful when the code is damaged or distorted.

5) *Version information*: There are various versions of the QR code standard; the version information (positioned near two of the finder patterns) simply identifies which one is being used in a code.

6) *Data cells*: Each individual black or white square that is not part of one of the standard features (the timing, alignment, and other patterns) contains some of the actual data in the code.

2.2. Distortions of QR Codes

When a QR Code is enlarged or made smaller using an image processing tool or the other ways, like illustrated in Figure 2, every module becomes distorted. It may look like a normal QR Code, but it may be difficult or impossible to read the code (Denso Wave Incorporated, 2017).

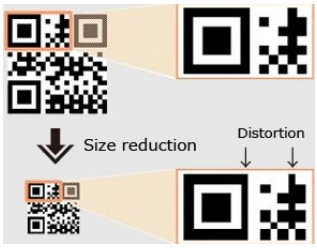


Figure 2. A distortion sample (Denso Wave Incorporated, 2017)

The camera's position while taking image capture of QR code can create some issues, like shown in Figure 3. These issues can be solved by the timer pattern and the position of finder pattern. For a QR code which is printed on a curved material, distortion also happened for curvature, like described in Figure 4. Using alignment patterns data bits, this issue can also be solved only if every data bits printed correctly.



Figure 3. A QR-code with camera angle distortions



Figure 4. A QR-code printed on a curved material



Figure 5. An uneven light example

These above distortions are mainly geometric distortions. Hence, the standard QR code decoding algorithms can correct geometric distortion via perspective projection (Hartley & Zisserman, 2003). This method estimates a projection matrix from four spatial patterns (known as the so-called *finder pattern and alignment pattern*) in the four corners of the QR codes (Yang, Huanle, Jianyuan, Chen & Wing, 2017). In practice, it is very likely that the estimated positions of the patterns are inaccurate. While small deviation is tolerable when decoding low-density QR codes, the perspective projection becomes unreliable for high-density ones as each module (module refers to the small square unit that makes up QR code) only contains few pixels (Yang, Huanle, Jianyuan, Chen & Wing, 2017).

2.3. Distortion Issue Analysis

After converting the image to grey-scale we will get data matrix like shown in Figure 6. If there is no noise due to any print related issues it will be easy to create a grid for each bit. This grid is imaginary in the sample image described in Figure 6. In fact, we can measure the grids as soon as we know the timing pattern positions.

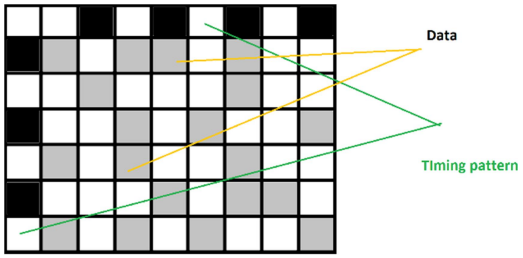


Figure 6. Timing pattern and data when no error or noise (grids are imaginary and gray are black bits)

Let us consider now the same image, but when QR code is on a curve material as shown in Figure 7. So, if we scan grid by grid comparing the size and the color of the neighboring bits, the

method will most likely face less chance to get misplaced error bits. The curvature and color features are very useful in minimizing the grid distortions.

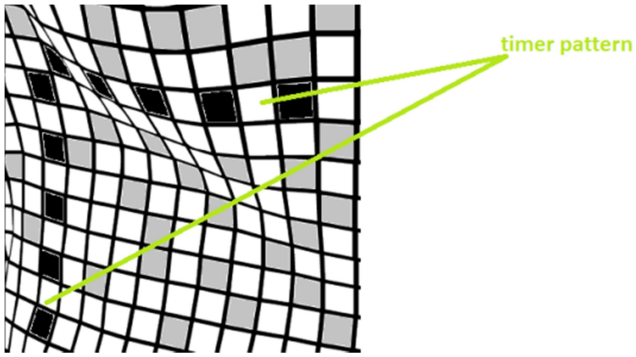


Figure 7. The timing pattern and data when placed on curved images (grids are imaginary and gray are black bits)

The QR code from Figure 8 illustrates different types of issues related to how they are present in the QR code. Without loss of complexity, we will ignore the curve effect as that can be recoverable by using timer pattern and alignment pattern angle calculation. It is known that without the curve effect, it will be easier to understand other issues.

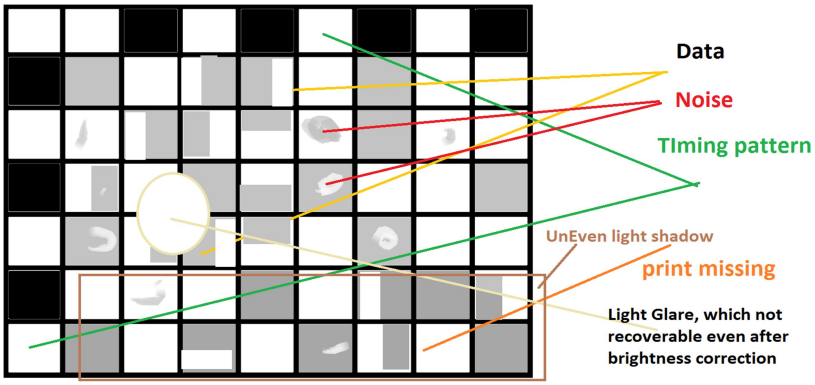


Figure 8. An error in the QR code (Grids are imaginary and gray are black bits)

The noise can be added from dust or other environmental issues. Sometimes, the color bit is missing because of printer issue. For a normal image or text one or two-pixel image, one bit missing is not a problem. However, in QR code where each and every pixel can represent a data, it is hard to ignore. The lighting glare is a known illumination issue. In a dark environment, using a flash light to capture a QR code can often contribute to this lighting glare problem. The glare size could make QR code completely unreadable. By correcting the brightness level and other color filtering features, the glare effect could be reduced, but often it makes some bits totally unreadable. Uneven light contributes to shadowing other objects over QR code while taking the image using a camera. If the image turns to binary image where only black and white color bits are present, then this issue is actually negligible. But in the case of a HiQ color QR code, this issue cannot be ignorable as we discussed earlier in this paper.

From finder pattern showing the QR code, we can detect the timing pattern and get the data area of QR code image, as illustrated in Figure 9.

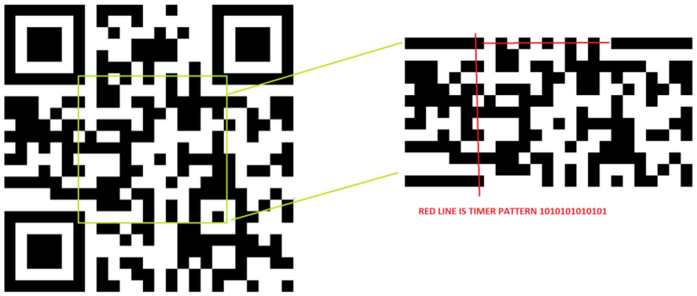


Figure 9. The areas of data extraction

If we see carefully the red line of Figure 9, we will see a sequence of white and black bits created, such as 10101010101010…. This string is static and used as a reference to measure all the bits in the QR code. Instead of using this sequence for full QR code, our algorithm uses neighboring bits as a reference sequence. In this way, the uneven light or color variations will have no effect in the scanning process.

3. Our Method to Solve QR Distortions

We proposed a new algorithm to check an area consisting of several bit positions when one third of those bits are known. Then we compare with the other features, such as total area color, the percentage of the area color and the percentage of the number of bits. Since one third of these data bits positions are known, there will be no chance of multiple/ambiguous solutions here.

3.1. The Matrix Based Calculation

Suppose we have an $i \times j$ matrix, that is, with i rows and j columns. We know the data of first row and column. We also know the total cumulative data of each row and column, that means the number of data bits. We assume the known data are 'a' and 'b' and the unknown data are represented as X. Let us consider the following $i \times j$ matrix:

$X_{1,1}$ $X_{1,2}$ $X_{1,3}$ $X_{1,4}$ ……$X_{1,j}$
$X_{2,1}$ $X_{2,2}$ $X_{2,3}$ $X_{2,4}$ ……$X_{2,j}$
$X_{3,1}$ $X_{3,2}$ $X_{3,3}$ $X_{3,4}$ ……$X_{3,j}$
……
$X_{i,1}$ $X_{i,2}$ $X_{i,3}$ $X_{i,4}$ …… $X_{i,j}$

For the sake of illustration purposes, let us take $i = j = 4$. We know the exact bits in the first row and first column, as well as the number of data bits in each row and column. For instance, we can take this 4×4 matrix, as follows:

a b a a
a X X X
b X X X
a X X X

The first row has three 'a's and one 'b', the second row has two 'a's and two 'b's, and the third row has three 'b's and one 'a'. The last row has two 'a's and two 'b's. Similarly, the first column has three 'a's and one 'b', the second column has one 'a' and three 'b's, and the third column has two 'b's and two 'a's. The last column has two 'a's and two 'b's. Using above information, by applying a backtracking algorithm we can generate the following data matrix:

a b a a
a b a b
b a b b
a b b a

This kind of calculation will be used for determining the data bits of a given QR code. The next subsection describes the pseudocode of all the data bits of a given QR code.

3.2. Algorithm

The backtracking algorithm is used for solving famous problems, such as N-queens problem or sudoku puzzle solving. Our input data is about 33% known matrix data bits. For simplicity, we will consider the black-white QR codes instead of the color ones. In terms of data encoding, we denote white as 1 and black as 0. We describe below our algorithm using the backtracking method defined recursively.

The first algorithm is called MatrixConstruction and has the responsibility to initialize the data bits from the matrix and call the MatrixGenerate algorithm.

Algorithm (MatrixConstruction):

Step 1. Initialize all elements of matrix *M* row(*i*)×column(*j*) with 0 initially.
Step 2. Initialize a list actualrowsum with all row's number of 1.
Step 3. Initialize a list actualcolsum with all columns' number of 1.
Step 4. Initialize two list size of '*i*' and '*j*' and set 0 for all value, and name the currentrowsum and currentcolsum.
Step 5. Initialize two integer value row and col with 0.
Step 6. Call MatrixGenerate(Matrix M, List currentrowsum, List currentcolsum, List actualrowsum, List actualcolsum, integer row, integer col)
Step 7. Print Matrix *M*

The second algorithm is called MatrixGenerate and has the responsibility to generate the matrix with the data transmitted from the MatrixConstruction algorithm.

Algorithm (MatrixGenerate):

Step 1.

```
If (List currentcolsum == actualcolsum & currentrowsum =actualrowsum)
        return true
```

Step 2.

```
If (col >=size of currentcolsum) {increase row=row+1 and call MatrixGenerate.}
              Else (If row>=size of currentrowsum )
                            { Set row = size of currentrow & col = size of currentcol}
                          Else {
             If (currentrowsum[row] < actualrowsum[row] &&
                 currentcolsum[col] < actualcolsum[col])
              { Matrix[row][col]=1,
                currentrowsum[row]++,
                currentcolsum[col]++.
 Boolean Flag =   MatrixGenerate(Matrix M, List currentrowsum,
                                 List currentcolsum,
                                 List actualrowsum,
                                 List actualcolsum,
                                 integer row, integer col++) }
If (Flag==false  &&  Matrix[row][col]==1) {
   Matrix[row][col]==1,
   Currentrowsum[row]--,
   Currentcolsum[col]--.
   Return Matrixgenerate(Matrix M, List currentrowsum,
                         List currentcolsum, List actualrowsum,
                         List actualcolsum, integer row,
                         integer col++)}
}
```

Step 3.

```
Return Flag.
```

The correctness of this algorithm follows from the efficient generate-and-test strategy from the backtracking method. The details are self-explanatory and follows from the fact the elements of the matrix will be in turn either 0 or 1 as needed from the matrix's data bits of the input.

As for the time complexity of this algorithm, let us denote by T(n) the number of execution steps needed by the algorithm with an input of size n. Analyzing the MatrixGenerate() method, we get the following recursive relation T(n) = n * T(n-1) + O(n^2) in the worst case. The solution of this relation is O(n!), hence an exponential time complexity.

3.3. A Java Implementation of Algorithm

A snippet of Java implementation of the above algorithm is as follows:

```
static boolean backTrack(short [][] arrMain, short[] curRow, short[] curCol,
                   short[] actRow, short[] actCol, int row, int col) {
  if (row>=curRow.length && col >= curCol.length) {
    for (int i = 0; i < curRow.length; i++) {
if (curRow[i] != actRow[i]) {
  return false;
}
    }
    for (int j = 0; j < curCol.length; j++) {
if (curCol[j] != actCol[j]) {
  return false;
}
    }
    return true;
  }
  if (col >= curCol.length) {
    return backTrack(arrMain, curRow, curCol, actRow, actCol, row+1, 0);
  }
  else
     if (row>=curRow.length) {
       return backTrack(arrMain, curRow, curCol, actRow, actCol,
                        curRow.length, curCol.length);
     }
     else {
 if (curRow[row] < actRow[row] && curCol[col] < actCol[col]) {
   arrMain[row][col] = 1;
   curRow[row]++;
   curCol[col]++;
 }
 boolean ret = backTrack(arrMain, curRow, curCol,
                         actRow, actCol, row, col+1);
 if (!ret && arrMain[row][col] == 1) {
   arrMain[row][col] = 0;
   curRow[row]--;
   curCol[col]--;
   return backTrack(arrMain, curRow, curCol,
                    actRow, actCol, row, col+1);
 }
 return ret;
    }
}  // end of method backTrack()
```

3.4. The Activity Diagram of our Algorithm

This section shows an activity diagram of our algorithm, by describing the big-picture of our algorithm. After scanning the QR code, we convert the image into a gray scale-based image. Since The timing pattern uses as reference pattern to identify other bits in QR code. Our method first sets it and use the data of timing pattern as our known dataset of the matrix. After that, we create a new matrix where the timing pattern remains on top of that and start to create different zones around the timing pattern. Figure 10 below shows all the activities performed by our algorithm from the Start state to the End state.

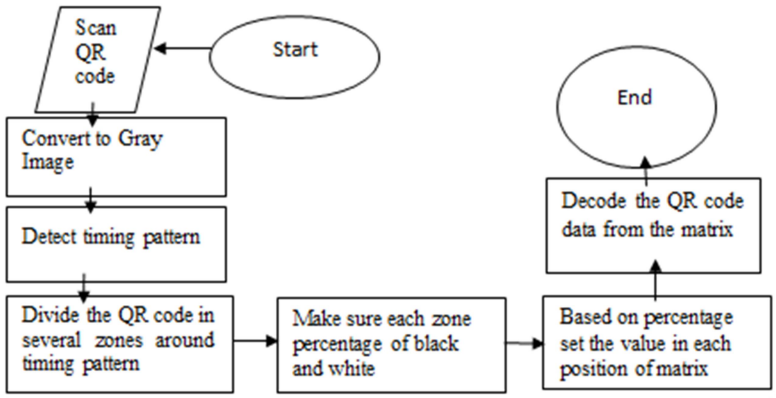


Figure 10. The activity diagram of our algorithm

3.5. The Zone Overlapping Method

In order to create a unique solution, our method needs at least 66% of block areas bits known, as well as information of the remaining three zones, such as their number of 0's and 1's. Hence, our method starts to analyze the area nearby the timing patterns, because the data of timing patterns is given as input. Therefore, if the base zone has N bit module, we need to know $N/2$ bits of data as well as every bit's information from three zones. At first, we divide the total *area*(A_t) with the number of module bits multiplied by 3. In this way, we guarantee that every area gets an equal number of *bits*(N_b).

In summary, the equation for the total number of zones is: $\sum Nz = \frac{Area(At)}{\sum bits(Nb)}$

To obtain N_z zones, we should split the total area. There are many approaches to split the total area, such as splitting horizontally or vertically, using circle areas or elliptic areas as long as we covered every bit's area from these three different zones. Figure 11 describes a splitting using elliptic curves.

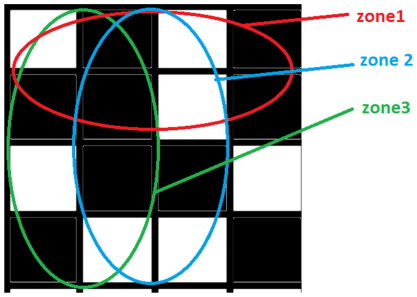


Figure 11. Splitting the total area using elliptic curves

After we identify every zone, we compare the percentage of the darker area and the brighter area. Based on that information we will know how many black and white bits exist on that zone. By comparing the three neighboring zones we set the bits based on the matrix construction. We are using some known bits area as we do not need any color correction. Our method checks for uneven light, correct noise, and more. Obviously, a larger known area for the data bits will decrease the time complexity of our algorithm.

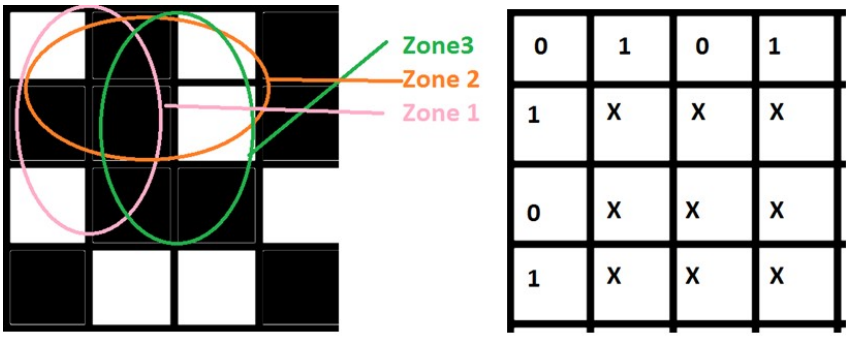
3.6. Some Essential Details of our Method

Once our method scans the image and gets the finder pattern and timing pattern, then our method crops the data part that includes the timing pattern. Then the image processing operation can start. Figure 12 describes the flow chart of a 4×4 example. Our method has the following six basic steps:

1. Find the timing pattern
2. Based on the timing pattern, divide the image in blocks

3. Based on each block number, start creating zones from each side of timing pattern
4. Get the zone percentages of darker and whiter area (black and white bits)
5. From all three different zones, set the values of black and white bits using the MatrixConstruction algorithm we proposed (Section 3.2)
6. Use the MatrixGenerate algorithm to generate the new QR code

Figure 12 shows the division of the matrix into three zones. We calculate the percentage of black and white bits for these regions and create a matrix where 1 means black, 0 means white, and unknown values as X. Using the value of timing pattern and the information from a full row and column, our method starts calculations on the nearby region.



Zone 1 66% black 34% white known bits 66%
Zone 2 50% black 50% white known bits 66%
Zone 3 66% black 34% white known bits 33% after first iteration 60%

Figure 12. The three zones and their corresponding block matrix

Based on the percentages of black and white bits, our method identifies value 1 for position 2,2 and value 0 for position 1,3. Figure 13 shows these values underlined with a red line.

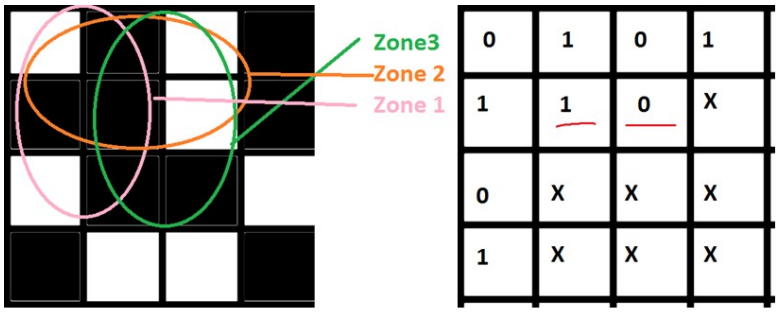


Figure 13. The data after the first iteration of our method

After the first iteration in Figure 14, our method created new zones. Since zone 3 was used in the first iteration, our method generates the new zones 4 and 5 by calculating their black and white bits percentage.

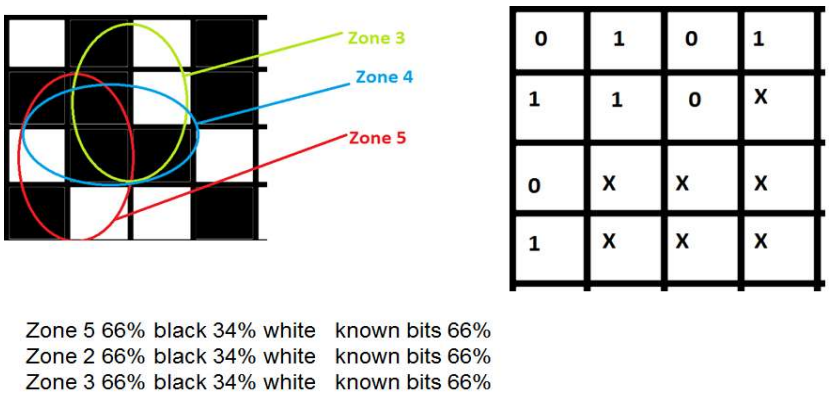


Figure 14. Generating new zones after the first iteration

By using the black and white percentages from all zones 3, 4 and 5, our method gets value 1 (corresponding to black) for positions 3,2 and 3,3. Figure 15 illustrates these two values with an underlined red line.

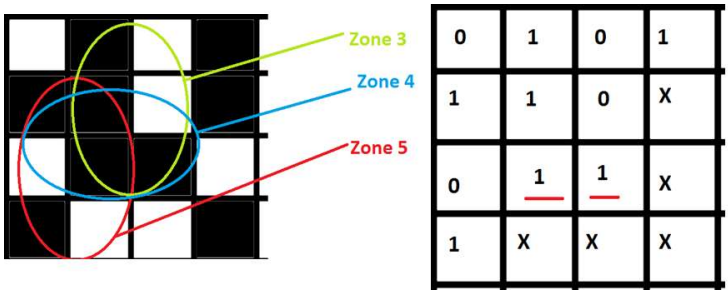
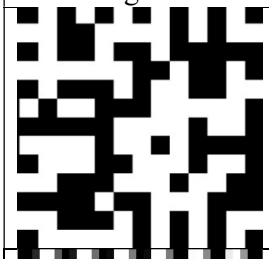
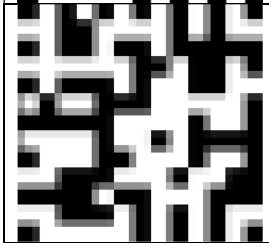


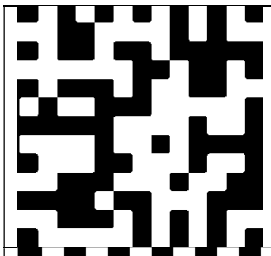
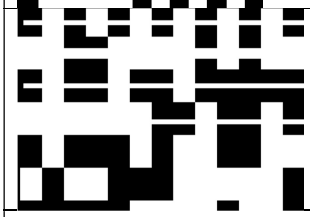
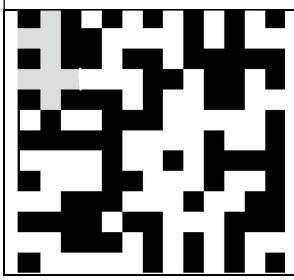
Figure 15. The data after the second iteration of our method

4. Results

We run our program in Visual Studio 15 using Aforge.net image library. Our C# framework version is DotNET-3.5. Table 4.1 shows the test results for the first 6×3 matrix results shown, where 8 values are known. The table used image after directly detached with timing pattern.

Table 1. Test Results

Test Image	Result first 6×3	Expected result	Note
	101010 001100 101101	101010 001100 101101	Image is clean, so no error
	101010 001100 101101	101010 001100 101101	Pixelated version of the previous one

	101010 001100 101101	101010 001100 101101	Image with edge problem
	101010 001100 101101	101010 001100 101101	Image with printer ink problem
	101010 001100 101101	101010 001100 101101	Uneven Color in upper corner

Based on our test results, we can see despite having more time complexity it can produce good results without having any image filtering first. Uneven light, shadow and printing issues can be completely recoverable as bits compared with its nearby bits. For the HiQ color, it could be a big help as we are not using a reference color from one area of the QR code with another area of the QR code. If the curvature is small and symmetric, we do not need to use curvature correction code while measuring the bits. But other QR code processing systems have to apply corrections because the timing pattern and faraway place bits curvature are much bigger than the curvature between two nearby bits.

5. Findings

In this research paper, we have used a recursive backtracking algorithm to implement our QR code recovery system. Since our method has a definite procedure, it could be a good start to become familiar with the algorithm implementation steps. Based on the problem statement, our algorithm leads to a unique solution. On the other hand, we have identified some limitations of our method as it could be very time consuming and it is ineffective when there are lots of branching from one state to the next. To remove these limitations we could implement other algorithms such as a genetic algorithm or based on machine learning techniques.

6. Conclusion

The use of QR codes is increasing significantly each year. It is used in the information technology sector, architectural engineers, contractors, building owners, developers, municipalities to improve the productivity and gain the marketing edge (Mitra, 2011). Hence, we believe it is imperative to study and analyze the QR code systems to make them more user-friendly and faster for all categories of users. A recursive backtracking algorithm was one of our first step to tackle the QR code systems and its behavior. Although using backtracking algorithm gives us a good explanation, it is also much more time consuming. In the near future, we will try to consider genetic algorithms or based on machine learning techniques.

References

Examples of problems encountered in reading a code. (2017). Examples of problems encountered in reading a code | QRcode.com | DENSO WAVE.

Furht, B. (2011). *Handbook of augmented reality*. Springer Science & Business Media.

Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge University Press.

Mitra, A. (2011). Innovative Uses of QR Codes by the AEC Industry. *Industrial Marketing Blog*, http://industrialmarketingtoday.com/innovative-uses-of-qr-codes-by-the-aec.

Ohbuchi, E., Hanaizumi, H., & Hock, L. A. (2004). Barcode readers using the camera device in mobile phones. (pp. 260-265). In *2004 IEEE International Conference on Cyberworlds*.

Woodford, C. (2018). How QR codes (and other 2D barcodes) work. .Explain that Stuff. Retrieved from http://www.explainthatstuff.com/how-data-matrix-codes-work.html.

Zhibo, Y., Xu, H., Deng, J., Loy, C. C., & Lau, W. C. (2017). Robust and Fast Decoding of High-Capacity Color QR Codes for Mobile Applications. *arXiv preprint arXiv:1704.06447.*