An Enhanced Two-Dimensional Color Barcode System

Md. Mashud Rana, M E Kawser, M-E-Rabbani, S M Rashid, K E U Ahmed

Department of Computer Science and Engineering, University of Chittagong, Chittagong-4331, Bangladesh.
2Shahjalal University of Science and Technology, Sylhet-3114, Bangladesh.
3Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Sylhet-3114, Bangladesh.

Corresponding Author: Md. Mashud Rana

Abstract
In a barcode system text information are encoded into image and an electronic device is used for retrieving encoded information by scanning that image. There are many kind of barcode systems are permitted by International Standard Audiovisual Number-International Agency (ISAN-IA). These barcode systems are used for different purposes like identifying retail and grocery items, encode Zip Codes, libraries, printed circuit board manufacturing etc. Nowadays, it has become a priority concern of storing more information into a barcode image from the perspective of the size of barcode, and 2-Dimensional (2-D) barcode are taking on increasing significance for their better capability to store massive amount of information than ever before. In this paper, we propose an enhanced encoding system for 2-D color barcode that can store more than thousand of characters, and a decoding system as well to retrieve stored information from the barcode image which applies a probabilistic error correction mechanism for reliability. The proposed barcode system uses a static hash table which contains the values of eighty seven different characters and a color table which contains 16 different colors for different patterns of nibble where each nibble pattern is unique. For information encoding, value of each character is converted into binary nibble, and then color pixel from the color table according to the nibble pattern. The decoding mechanism is the reverse of encoding along with the error correction mechanism.

Keywords: two dimensional barcode, color barcode, symbology, hash table, nibble

INTRODUCTION
Barcode provides a simple and inexpensive method of storing encoded text that can be easily retrieved by portable electronic readers with extreme accuracy. Early barcodes systems are typically one-dimensional (1-D) containing a series of vertical black lines separated by white spaces which represents encoded information. Conventional 1-D barcodes have an apparent weakness in terms of information density given a limited area (Hahn and Joung 2002). The necessity of increasing information density in a limited space led to the development of a black and white 2-D barcode system which is often just a series of stacked 1-D barcode and still constrained as to the amount of information require to be encoded by many real world applications. The need to increase the amount of information in a barcode symbol well beyond the capacity of conventional 1-D and 2-D barcode brought the introduction of a new form of barcode called 2-D color barcode.

However, the added benefits of large storage capacity in 2-D color barcode come with challenges of efficient encoding into, and reliable decoding of information from the barcode images. Only a few papers in the literature addressed this issue. Hahn and Joung, 2002, proposed a barcode reader to decode 2-D barcode symbology of PDF417. They also presented an algorithm based on the location and distance between extreme points of each row of the barcode image. In their proposed method, they applied Reed-Solomon error correction algorithm for error detection and correction in encoded code-words that are decoded afterwards. They argue that their proposed algorithm is robust from the high convolution distortion environments such as defocusing and warping, even under bad illuminating condition. Though the algorithm (Hahn and Joung 2002) is said to be shown good performance, it is only designed to work with typical 2-D barcode not 2-D color barcode. The impact of various factors such as number of print-scan (PS) or copy, cycles, image restoration to offset PS-induced degradation, the authentication algorithms used on the payload density (PD) of color barcode are described by Simske, Sturgill and Aronoff, 2009. From the empirical results, they showed that PS cycle reduces the PD by approximately 55% under all tested conditions, selecting better authentication algorithm increases PD roughly by 50%. Restoration however, was found to increase PD less substantially (around...
30%) when combined with spectral pre-compensation to optimize the color settings. Parikh and Jancke, 2008, presented a new approach for localizing and segmenting 2-D color barcode in the images taken by the consumers under various scenarios. They tested their proposed method using images of Microsoft’s recently introduced 2-D High Capacity Color Barcode (HCCB). They mainly focused on computational efficiency and accuracy in retrieving color barcode from the images taken under diverse environments. Tarassenko, Ivan Dychka and Sulema, 2003, discussed several issues regarding transforming data (such as compression) and storing data into a multi-color matrix barcodes. They showed that transformation of textual data in digital format not only raises the density of representation of the data but also allows applying an adjusting code with purpose of maintenance of a necessary level of noise stability. Villan, Voloshynovskiy, Koval and Pun, 2006, adapted the theory of multilevel coding with multistage decoding (MLC/MSD) to the print-and-scan channel. They presented experimental results confirming the utility of their proposed channel model, and showed that multilevel 2-D bar codes using MLC/MSD can reliably achieve the high-capacity storage requirements of many multimedia security and management applications. In this paper we analyze the efficiency of 2-D color barcode in storing large amount of data, propose an efficient algorithm for encoding data into a high capacity color barcode, and provide the details of how to decode the barcode image with a probabilistic error correction method. The rest of the paper is organized as follows: section-2 briefly describes available 2-D barcode symbologies, analysis and implementation details of our proposed system are included in section-3. Finally, section-4 includes concluding remarks and future research directions.

**Brief Overview of 2-D Barcode Systems**

A barcode symbology defines the technical details of a particular type of barcode such as the width of the bars, supported character set, method of encoding, checksum specifications, etc. Most users are more interested in the general capabilities of a particular symbology (how much and what kind of data can it hold, what are its common uses, etc) than in the excruciating technical details. The section includes very short overview of the available 2-D barcode symbologies. Figure-1 shows four well known 2-D barcodes.

![PDF417](a) ![MaxiCode](b) ![Aztec Code](c) ![Data Matrix](d)

Figure 1: (a) PDF 417 (b) MaxiCode (c) Aztec Code (d) Data Matrix

PDF417 (Portable Data File 417) is a high density 2-D bar code symbology that essentially consists of a stacked set of smaller bar codes (Wang and Yinjun, 1993). It can encode as many as 2725 data characters in a single bar code. The complete specification for PDF417 provides many encoding options including data compaction options, error detection and correction options, and variable size and aspect ratio symbols. The low level structure of a PDF417 symbol consists of an array of code-words (small bar and space patterns) that are grouped together and stacked on top of each other to produce the complete printed symbol. An individual code-word consists of a bar and space pattern of 17 modules wide. A complete PDF417 symbol consists of at least 3 rows of up to 30 code-words and may contain up to 90 code-word rows per symbol with a maximum of 928 code-words per symbol. The symbology also allows for varying degrees of data security or error correction and detection. MaxiCode (Chandler, Batterman, and Shah, 1989), is a fixed size matrix style symbology which is made up of offset rows of hexagonal modules arranged around a unique bulls-eye finder pattern. Each MaxiCode symbol has 884 hexagonal modules arranged in 33 rows with each row containing up to 30 modules. The maximum data capacity for a MaxiCode symbol is 93 alphanumeric characters or 138 Numeric characters. Although the capacity of a MaxiCode symbol is not as high as other matrix style bar code symbologies, it was primarily designed to encode address data which rarely requires more than about 80 characters. Aztec Code (Longacre, Andrew and Husse, 1997) is another high density 2-dimensional matrix style bar code symbology that can encode up to 3750 characters from the entire 256 byte ASCII character set. The symbol is built on a square grid with a bulls-eye pattern at its center. Data is encoded in a series of "layers" that circle around the bull’s-eye pattern. Its primary features include- a wide range of sizes allowing both small and large messages to be encoded, orientation independent scanning and a user selectable error correction mechanism. Data Matrix (http://en.wikipedia.org/wiki/Datamatrix) is a high density 2-D matrix style bar code symbology that can encode up to 3116 characters from the entire 256 bytes ASCII character set. The symbol is built on a square grid arranged with a finder pattern around the perimeter of the bar code symbol. Data Matrix is being used to encode product and serial number information on electrical rating plates, to mark of
surgical instruments in Japan, to identify lenses, circuit boards, and other items during manufacturing.

Recently, Microsoft Corp. and ISAN-IA jointly introduced 2-D High Capacity Color Barcode (http://www.seattlepi.com/business/311712_software_16.html). The main applications of High Capacity Color Barcode (HCCB) are to identify commercial audiovisual works such as motion pictures, video games, broadcasts, digital video recording and other media. It has rows of strings of symbols (triangles), each of which can be one of four colors: black, red, green and yellow. The number of symbols in each row is always an integral multiple of the number of rows, which can vary. HCCB is designed to have a black boundary around it, further surrounded by a thick white band. These patterns are designed to act as visual landmarks to locate the barcode in an image.

Figure 2: Microsoft's HCCB

ColorCode™ (Zap Code) is the world's first 2-D code system that integrates both online and offline, analog and digital components. ColorCodes can be as small as 1 square cm, and as big as your imagination. A ColorCode's™ pattern potential is determined by the number of colors it encompasses. In a standard, four-color 5 x 5 cell code, more than 17 billion patterns can be created. Based on the encoding and decoding algorithms, it is actually categorized as: ColorCode™, GrayCode™ and Numeric Code.

DESIGN AND IMPLEMENTATION

Colors and Size

In a linear barcode, black bars and white spaces are used to represent 1 and 0 respectively. So, for representing a pattern of 8-bits like “01101001” four black bars and four white spaces are needed. But for representing this 8-bits pattern by using four different colors, only as half number of bars as required in case of linear barcode are enough which results in an increased information density given limited area for barcode image. However, increasing number of colors doesn’t help significantly to improve the information density in all cases. This is apparent from the figure-3. A careful observation of this figure reveals that, using more than 16 colors in a barcode system actually will not help great deal in increasing the information density as the graph almost saturated after number of colors reaches to 16.

![Figure 2: Microsoft's HCCB](image)

![Figure 3: Number of Bits Vs Number of Colors](image)

This can also be modeled by the mathematical equation as - \( \text{Number of bit represent by each color} = \log_4(\text{Number of color}) \). As a consequence, our proposed high capacity color barcode system uses 16 different colors to build barcode image. Moreover, the length and height of our proposed barcode image are 1.15 inches and 1.5 inches respectively which is the standard size currently using by most of the available barcode symbologies.

Supported Characters

The proposed high capacity color barcode system supports wide range of characters set including alphabets, numbers, arithmetic operators, braces, punctuation & quotation symbols, special characters etc. The list of the supported charthers is shown in table-1.

<table>
<thead>
<tr>
<th>Types of Characters</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet</td>
<td>a-z, A-Z</td>
</tr>
<tr>
<td>Numerals</td>
<td>0-9</td>
</tr>
<tr>
<td>Arithmetic Operators</td>
<td>+,-,*,%,^</td>
</tr>
<tr>
<td>Braces</td>
<td>(,),{,}[,]</td>
</tr>
<tr>
<td>Punctuation &amp; Quotation Symbols</td>
<td>!, ?, ,, &quot;&quot;, .</td>
</tr>
<tr>
<td>Special Character</td>
<td>@,#,$,%,&amp;, new line, horizontal tab, ^</td>
</tr>
</tbody>
</table>

Table 1: List of Supported Characters

The index value of each character as shown in the hash table (table-2) is converted into 8-bit binary value, which is used to be encoded into color code with specific color and pattern.
Table 2: Hash Table

<table>
<thead>
<tr>
<th>Character (Key)</th>
<th>Index (Value)</th>
<th>Character (Key)</th>
<th>Index (Value)</th>
<th>Character (Key)</th>
<th>Index (Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>b</td>
<td>1</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>z</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>...</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Z</td>
<td>5</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>!</td>
<td>@</td>
<td>#</td>
<td>%</td>
<td>&amp;</td>
</tr>
</tbody>
</table>

Table 3: Color Description with Nibble Pattern

<table>
<thead>
<tr>
<th>Color</th>
<th>RGB Value</th>
<th>Nibble</th>
<th>Color</th>
<th>RGB Value</th>
<th>Nibble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>255,0,0</td>
<td>0000</td>
<td>Pink</td>
<td>238,130,238</td>
<td>1000</td>
</tr>
<tr>
<td>128,0</td>
<td>0011</td>
<td>0100</td>
<td>Black</td>
<td>128,128,128</td>
<td>1001</td>
</tr>
<tr>
<td>Blue</td>
<td>0,0,25</td>
<td>0010</td>
<td>0,0,0</td>
<td>128,0,128</td>
<td>1010</td>
</tr>
<tr>
<td>165,42,42</td>
<td>0010</td>
<td>1111</td>
<td>138,43,226</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>128,0,128</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>128,0,128</td>
<td>1110</td>
<td></td>
</tr>
</tbody>
</table>

**Symbology**
The 16 different colors that are used in our proposed 2-D barcode system, along with their corresponding RGB value and nibble pattern are shown in table-3. For each nibble, 3 pixels is filled with a specific pattern and thus each character is encoded by using 6 pixels using two patterns of symbologies shown in figure-4.

**Format of Barcode Image**
The proposed system encodes the information in a barcode image of rectangular shape having size of 102 pixels in length and 100 pixels in height. The format of encoded image is structure is shown in figure-5.

**Error Detection and Correction**
In any color barcode system, the barcode images are subject to color fading due to various environmental conditions that can cause stored information be decoded incorrectly. For ensuring reliability in retrieving information under such condition our proposed barcode symbology applies a probabilistic error detection and correction mechanism In the proposed error correction and detection method, the first two pixels of a pattern are compared at first. If two pixels colors match then it is assumed that there is no error due to color fading in the image, otherwise occurrence of error is detected. The third and last pixel’s color matches with that of first one then first...
pixel color is treated as actual color otherwise, second pixel’s color is treated as actual color. For simplicity we just include this error correction method into the decoding algorithm given section 3.6, instead of writing separately.

**Encoding and Decoding Algorithms**

We propose two algorithms for information encoding into and decoding from barcode image respectively. The details of these two algorithm are included in appendix-1 due to maintain orientation format of this paper. The novelties of our proposed encoding and decoding algorithms are their simplicity and computational efficiency. Moreover, our proposed decoding algorithm can be implemented into the micro-devices such as mobile phone easily so that proposed color barcode system can be embedded into hand-held devices for popular consumer based applications where no specialized barcode reader is needed. The `GetColor` method (please refer to the appendix-1 for details) is used during encoding process for selecting the colors of pixels for characters in the text to be encoded, whereas this method is used by the decoding algorithm for identifying colors of pixels taking color fading in consideration for error detection and correction.

**CONCLUSION AND FUTURE WORKS**

We propose a noble 2-D color barcode system that can store more information than conventional linear and 2-D barcode systems. In our proposed method only an insignificant percentage of pixels (around 0.01%) are allocated for header, and rest of the pixels (around 99.99%) are made available for encoding actual information. We use only 16 different colors to build barcode images as using more than 16 colors doesn’t help much in increasing information density, and it also requires extremely sensitive scanner to identify colors with near RGB values if more colors are used. However, it is still possible to decode more information into our proposed system and the error detection algorithm can also be more robust. In future we are going to investigate these two issues.

**REFERENCES**


Hahn, H. I. and J. K. Joung (2002). Implementation of Algorithm to Decode Two-Dimensional Barcode


**APPENDIX 1**

```plaintext
GetColor ( Pixel MainPixel,Pixel SupportingPixel1,Pixel SupportingPixel2 ) Return Color
1. If Color(MainPixel)=Color(SupportingPixel1)
   OR Color(MainPixel)=Color(SupportingPixel2)
      Then:
      return Color(MainPixel)
   Else
      If Color(SupportingPixel1)=Color(SupportingPixel2)
         Then:
         return Color(SupportingPixel1)
      Else:
         return Color(SupportingPixel2)
   [End of If Structure]
2. EXIT.
```

**CODE**

```plaintext
ENCODE( H<character,integer> , T<string,color> , Text)
Here H<character,integer> is the Hash Table which stores character as key and index as value.
```
Text is the message that to be encoded. Here Pixel is a 2-dimensional array which represents bar-code image.

1. **Initialize Segment Variables.**
   
   Set BarcodeWidth := 102, BarcodeHeight := 100, RowCounter := 0, ColumnCounter := 0, PixelsPerPattern := 3, BitsPerPattern := 4, ByteStream := null, ByteStreamLength := 0, Color := NULL, TotalTextLength := 0, IsLPattern := False, Index := 0, Text := Empty, StreamCounter := 0.

2. **Load Byte Stream From Text**
   
   Repeat Step 2.1 while Index < TotalTextLength.

   2.1. **Repeat Step 2.1 While ColumnCounter < BarcodeWidth**
      
      2.1.1. **Repeat Step 2.1.1** and 2.1.2 While RowCounter < BarcodeHeight
             
             **2.1.1.1. If SteamCounter < ByteStreamLength**
                 
                 If SteamCounter < ByteStreamLength Then:
                 
                 GET Color([BarcodeWidth*BarcodeHeight]/PixelsPerPattern) Then:
                 
                 GET Color(T<character,integer>, Color := NULL, TotalTextLength := 0, IsLPattern := False, Index := 0, Text := Empty, StreamCounter := 0.

             Else:
             
             2.1.2. **Retrieve character from each pattern**
                 
                 SET ColumnCounter := ColumnCounter + BitsPerPattern, SteamCounter := SteamCounter + BitsPerPattern.

             GetCharacter([BarcodeWidth*BarcodeHeight]/PixelsPerPattern) Then:
                 
                 GetCharacter(T<character,integer>, Color := NULL, TotalTextLength := 0, IsLPattern := False, Index := 0, Text := Empty, StreamCounter := 0.

             Else:
             
             2.1.2. **Retrieve Data from Color**
                 
                 2.1.2.1. If Header := 0 Then:
                 
                 GetBinaryStreamAsString(T<character,integer>, Color := NULL, TotalTextLength := 0, IsLPattern := False, Index := 0, Text := Empty, StreamCounter := 0.

             Else:
             
             2.1.2.2. **Retrieve character from each pattern**
                 
                 GET Character(Color) Then:
                 
                 GET Character(Color) Then:

             Else:
             
             End of If Structure of Step 2.1.1.1

2.2. **Set ByteStream := ByteStream + ConvertBinaryStreamToText(Color), ByteStream := null**

   End of If Structure of Step 2.1.2

3. **EXIT**

```
T<character,integer> is the Color Table which stores key and corresponding color as value.
T<color,string> is the Color Table which stores color and corresponding binary string as value.

IsLPattern := False, Index := 0,

If IsLPattern := True
ColumnCounter := ColumnCounter + 2,
Pixel[ColumnCounter,RowCounter+1] := Color,
Pixel[ColumnCounter+1,RowCounter] := Color,

Else:

If IsLPattern := True
Header := ConvertInto12BitBinaryString(TotalTextLength * 8), SteamCounter := 0.

2. [Load Byte Stream From Text]
Repeat Step 3 while Index < TotalTextLength.

3. SET ByteStreamLength := ByteStreamLength + ConvertBinaryString(GetIndexValue(H<character,integer>), CharacterAt(Text,index))) , Index := 0.

4. [Add Header with byte stream and calculate total Byte Stream length]
SET ByteStream := Header + ByteStream , ByteStreamLength := Length(ByteStream).

5. **Repeat Step 5.1.1 and 5.1.2 While RowCounter < BarcodeHeight**
   
   **5.1.1.** Repeat Step 5.1.1.1 and 5.1.1.2 While ColumnCounter < BarcodeWidth
               
               **5.1.1.1. If SteamCounter < ByteStreamLength**
                   
                   SET Color := GetColor(T<color,string>, SubString(ByteStream,SteamCounter,SteamCounter+BitsPerPattern)), Color := NULL.

               Else:
               
               SET Color := RandomColor()

               [End of If Structure of Step 5.1.1.1]

               **5.1.2.** If IsLPattern := True Then:
                   
                   [Draw L pattern]
                   
                   SET Pixel(ColumnCounter,RowCounter) := Color,
                   Pixel(ColumnCounter+1,RowCounter) := Color,
                   Color := ColumnCounter + 1, IsLPattern := False

               Else:
               
               [Draw reverse of L pattern]
               
               SET Pixel(ColumnCounter,RowCounter) := Color,
               Pixel(ColumnCounter+1,RowCounter) := Color,
               Color := ColumnCounter + 2, IsLPattern := True

               [End of If Structure of Step 5.1.1.2]

               [End of loop of step 5.1.1]

   **5.1.2.** SET RowCounter := RowCounter + 2
               
               [End of loop of step 5.1]

               [End of If Structure of step 5]

6. **EXIT**

DECODE(H<integer_character> : T<color,string>) Return Text As String
Here H<integer_character> is the Hash Table which stores index value as key and character as value.
T<color,string> is the Color Table which stores color as key and corresponding binary string as value.

1. If Initialize Segment Variables.

   Set BarcodeWidth := 102, BarcodeHeight := 100, RowCounter := 0, ColumnCounter := 0, PixelsPerPattern := 3, BitsPerPattern := 4, ByteStream := null, ByteStreamLength := 0, Color := NULL, TotalTextLength := 0, IsLPattern := False, Index := 0, Text := Empty, StreamCounter := 0.

2. Repeat Step 2.1 and 2.2 While RowCounter < BarcodeHeight

   2.1. **Retrieve color using Error detection and correction mechanism**
       
       If IsLPattern := True Then:
       
       [Read L pattern color]
       
       SET Color := GetColor(Pixel(ColumnCounter,RowCounter),
                   Pixel(ColumnCounter,RowCounter+1),
                   Pixel(ColumnCounter+1,RowCounter),
                   Pixel(ColumnCounter+1,RowCounter+1))

       Else:
       
       [Read reverse L pattern color]
       
       SET Color := GetColor(Pixel(ColumnCounter,RowCounter),
                   Pixel(ColumnCounter,RowCounter+1),
                   Pixel(ColumnCounter+1,RowCounter),
                   Pixel(ColumnCounter+1,RowCounter+1))

       [End of If Structure of Step 2.1.1]

   2.1.2 **Retrieve Data from Color**

       2.1.2.1. If Header := 0 Then:
       
       GetBinaryStreamAsString(T<color,string>, Color := NULL,
                   Header := Header + BitsPerPattern

       2.1.2.2.1. If Header := 12 Then:

       Set ByteStream := ByteStream + ConvertBinaryStreamToText(Color),
       ByteStream := null

       [End of inner if structure of 2.1.2.1.1]

       [End of If Structure of Step 2.1.2.2.1]

       2.2. **GetCharacter(Color) Then:**

       SET SteamCounter := SteamCounter + BitsPerPattern

       [End of inner loop of step 2.1.1]

   2.2. SET RowCounter := RowCounter + 2
               
               [End of loop of step 2]

3. **EXIT**