Samuel Steinberg

November 21st, 2019

ECE472

Final Project Report: QR Codes

Background and Storage

Developed in the 1990's for Japan's automotive industry, QR codes have taken off in popularity due to faster readability and greater storage capacity over standard 1D barcodes. This advantage lies in the QR code's structure: They are a 2D matrix, so they can store information in both horizontal and vertical directions. This allows them to hold much more information with multiple datatypes (See Table 1) and yet does not take as long to read as a 1D horizontal barcode, which can store a maximum of 43 characters (standard Code 39 barcode).

Table 1: Maximum QR Code Storage: Version 40-L

Data Type	Maximum Characters
Numeric	7,089
Alphanumeric	4,296
Byte/Binary	2,953
Kanji/kana	1,817

The flexibility of the 2D matrix structure allows for QR codes to have a large number of practical applications in the technological and business world, storing information such as: customer rewards programs, WIFI network logins, website logins, payments and URLS's among others. They are also used on mobile operating systems (iOS 11+), augmented reality systems, and inventory tracking.

Components

- Finder Pattern: The finder pattern consists of three identical large square black structures positioned in three of the corners of the QR code. This pattern allows the decoder to recognize the code and orient itself accordingly. They are surrounded by a single-pixel width separator to improve recognizability of the pattern from the actual data.
- Timing pattern: Pattern used as an offset marker and enables the decoder to determine the width of a single module.
- Version Information: Identifies the QR code version number to the decoder.
- Quiet Zone: A clear area around the QR code where nothing is printed, is used as a buffer to separate it from any adjacent patterns, images, text, etc. This will usually be the size of four times the width of a single data square.
- Alignment Pattern: Compensates for mild-moderate image distortions supporting the decoder. This also helps correct any skew in the image.

- Data: Data to be encoded is converted into a bit stream and stored in “codewords” (8-bit parts) in the data section.
- Format Information: Located next to the Finder Pattern separators, this section is 15 bits that stores information about the error correction level in the QR code in addition to its masking pattern.

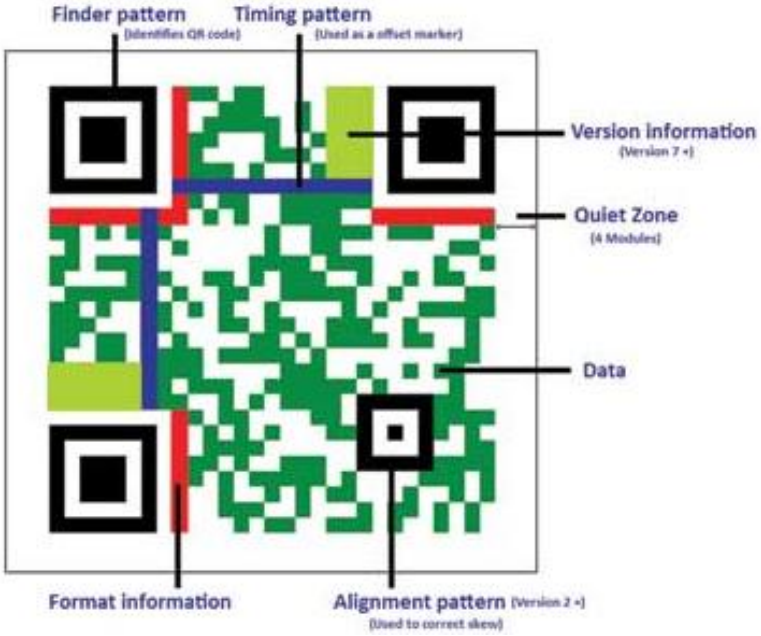

Figure 1: QR Code Component Diagram

Error Correction

Codewords use the Reed-Solomon error correction algorithm, which is block-based. Error correction level can be adjusted, though with higher error correction level there is less storage capacity. This makes it possible to make more aesthetically pleasing codes that still work, thus making them more desirable for marketing, incorporating logos, etc. Error correction levels are shown in Figure 2:

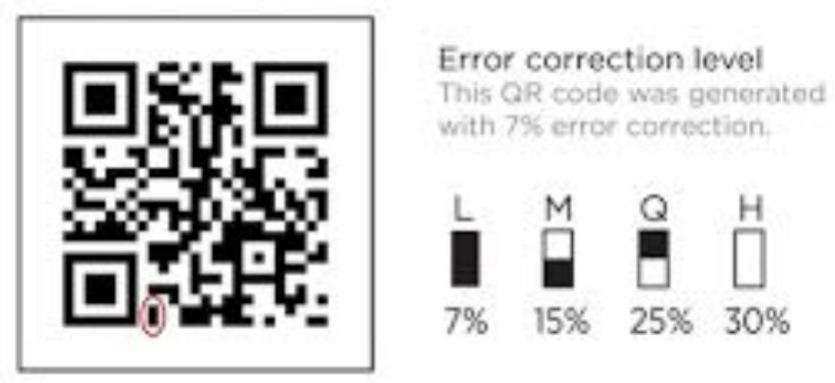

Figure 2: Error Correction Rate from Format Information

The encoder works by taking a data block and adding redundant bits. Events such as noise can cause errors during storage, and the decoder will process each block and attempt to correct these errors and recover data.

Encoding and Decoding

Encoding uses a Reed-Solomon code of RS(n, k) with s-bit symbols. The encoder will take k data symbols of s bits to make an n symbol codeword.

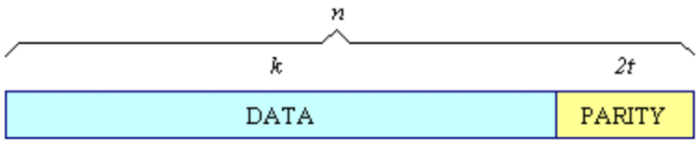

Figure 3: Codeword architecture in a QR code using Reed-Solomon.

The extra parity bits in the codeword are $2t$ in length, or n-k symbols. This will give the decoder room to find and fix errors. The decoder can correct t errors and up to $2t$ erasures (when location of blundered symbol is known). Each time a codeword is decoded, there are three potential outcomes:

1. Transmitted codeword is recovered if $2s + r < 2t$, were s is the number of errors and r is the number of erasures.
2. Decoder detects it cannot recover the code and sends this outcome back.
3. Decoder will mis-decode and recover incorrect code with no indication of this error.

Project Encode/Decode Implementation Examples

The implementation of QR code encoding an decoding was put into practice in this project. The code was written in Python and took advantage of libraries using the aforementioned methods. Two programs were written: One for encoding an image and one for decoding. Both were built to run on the command line. Below are a few examples from each, where input and output noted:

Encode Examples:

1. INPUT DATA: www.eecs.utk.edu
   OUTPUT: Encoded QR code shown below:

2. INPUT DATA: VCARD File
   OUTPUT: Encoded QR Code containing contact information:

Decode Examples

3. INPUT: Chick-Fil-A rewards card

   OUTPUT: Member ID number and bookkeeping information to link account with purchase.
4. INPUT: Noisy Chick-Fil-A rewards card

   OUTPUT: Same as example 3, note that due to not needing much space for the data error is allowed to be up to 30% (See Figure 2). Also, note that the orientation is flipped.