HiQ: Robust and Fast Decoding of High-Capacity Color QR Codes

YANG, Zhibo

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Philosophy in Information Engineering

The Chinese University of Hong Kong
July 2016
Abstract of thesis entitled:

HiQ: Robust and Fast Decoding of High-Capacity Color QR Codes

Submitted by YANG, Zhibo

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in July 2016

Color brings extra data capacity for QR codes, but it also brings tremendous challenges for the decoding because of color interference and illumination variation, especially for high-density QR codes. In this work, we put forth a framework for high-capacity QR codes, HiQ, which on one hand optimizes the decoding algorithm for high-density QR codes to achieve robust and fast decoding by mobile devices, and on the other hand leverages color to increase the data capacity of traditional QR codes. HiQ constructs a color QR code by combining multiple monochrome QR codes together in a layered manner and uses a set of highly discriminative colors to represent different combinations of binary data modules from the monochrome QR codes. To achieve robustness in decoding high-density QR codes, we propose a learning-based approach for color recovery and a robust geometric transformation algorithm to correct the geometric distortion.

In addition, we provide a challenging large-scale color QR code dataset, CUHK-CQRC, which consists of 5390 high-density color QR code samples captured by different phone models under different lighting conditions. We compare HiQ with the baseline method [3] on CUHK-CQRC. Experimental results show that HiQ outperforms [3] by 286% in decoding success rate and 60% in bit error rate. We also implement HiQ in different mobile platforms and demonstrate the effec-
tiveness of our framework in real-world application across different phone models. In particular, by using 3-layer\(^1\) color QR codes as an example, we show that HiQ can robustly decode QR codes of 2600 bytes, 4500 bytes and 7700 bytes of user data with printout size as small as \(24 \times 24 \text{ mm}^2\), \(34 \times 34 \text{ mm}^2\) and \(50 \times 50 \text{ mm}^2\), respectively, within 5 seconds.

\(^1\)HiQ is capable of more layers
彩色QR碼雖然增大了傳統單色QR碼的信息容量，但是因為顏色通道之間的互相干涉和光照對顏色影響，顏色的引入也極大地增大了掃描器解碼的難度，尤其是高密度的QR碼。我們在此論文中提出了一個名為HiQ的高容量QR碼的框架。一方面，HiQ針對性地優化了對高密度QR碼的解碼算法以提高用手機掃碼的速度和魯棒性；另一方面，HiQ使用顏色信息來增大傳統QR碼的信息容量。在HiQ中，我們用分層的方式把多個單色QR碼組合到一起，並且用一組容易分辨的顏色分別編碼這多個單色QR碼中不同二值模塊的組合。為提高高密度的QR碼的魯棒性，我們提出了一種基於學習的顏色還原的方法和魯棒性強的幾何變換算法來矯正QR碼的幾何畸變。

此外，我們還創造了一個極具挑戰性的大型彩色QR碼數據集，CUHK-CQRC，其中包括在不同的光照條件下，使用不同型號的手機拍攝的5390張高密度的彩色QR碼圖像。我們比較了HiQ與基準方法[3]在數據集CUHK-CQRC上表現，實驗結果表明，對HiQ比[3]在解碼成功率上高出286%，在比特錯誤率上減少了60%。我們也在不同的移動平臺實現了HiQ框架，並且使用了不同手機型號展示了HiQ框架在現實世界的應用程序的有效性。我們使用3層顏色QR碼\(^2\)作為一個例子，在手機上的測試結果顯示HiQ能夠在5秒內快速有效地解碼存儲了2600字節，4500字節和7700字節的用戶數據的QR碼，其最小打印面積分別能達到24×24 mm\(^2\), 34×34 mm\(^2\)和50×50 mm\(^2\)。

\(^2\)HiQ能夠產生更多層數的QR碼
Acknowledgement

First, I would like to thank my supervisor, Prof. Wing Cheong Lau, and co-supervisor, Prof. Chen Change Loy, who have been providing me the patient guidance, encouragement and advice throughout my postgraduate study. Moreover, I am very grateful that Prof. Lau has always been supportive of my exploration of the research field that I am interested in and provides many valuable advices that shape my research career.

Secondly, I would also like to thank all the lab members, Chak Man Li, Huanle Xu, Zhiyi Cheng, Guanchen Li and Ronghai Yang for their support and help, as well as the staff at The Chinese University of Hong Kong who helped me during my life in Hong Kong.

Lastly, I wish to thank my parents, my brother and my girlfriend, Jianyuan Deng, for their support and love throughout my whole life.
This work is dedicated to my family, who have been a ceaseless source of support, encouragement and love to me throughout my entire life.
Contents

Acknowledgement iv

1 Introduction 1

2 Background Study 5

2.1 Basics of QR codes .................................................. 6

2.1.1 Data encoding of QR codes .................................. 6

2.1.2 Decoding of QR codes ......................................... 8

2.2 Color barcodes ....................................................... 11

3 HiQ: A Layered Framework for High-Capacity QR Codes 13

3.1 Framework overview ............................................... 14

3.2 Data encoding ....................................................... 15

3.2.1 Encoding scheme ............................................... 15

3.2.2 Color selection .................................................. 17

3.2.3 Pattern coloring ................................................ 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 Challenges of Decoding Color QR Codes</td>
<td>22</td>
</tr>
<tr>
<td>3.3.1 High density</td>
<td>23</td>
</tr>
<tr>
<td>3.3.2 Chromatic distortion</td>
<td>24</td>
</tr>
<tr>
<td>3.4 Data decoding</td>
<td>26</td>
</tr>
<tr>
<td>3.4.1 Color QR code detection</td>
<td>26</td>
</tr>
<tr>
<td>3.4.2 Robust geometric transformation</td>
<td>28</td>
</tr>
<tr>
<td>3.4.3 Color recovery</td>
<td>31</td>
</tr>
<tr>
<td>4 Robust Color Recovery for Color QR Codes</td>
<td>33</td>
</tr>
<tr>
<td>4.1 Learning-based color recovery</td>
<td>34</td>
</tr>
<tr>
<td>4.2 Layered strategy</td>
<td>36</td>
</tr>
<tr>
<td>4.2.1 Methodology</td>
<td>36</td>
</tr>
<tr>
<td>4.2.2 Preliminary evaluation</td>
<td>38</td>
</tr>
<tr>
<td>4.2.3 Comparison between QDA and layered SVM</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Classifier training</td>
<td>42</td>
</tr>
<tr>
<td>5 Implementation Details</td>
<td>44</td>
</tr>
<tr>
<td>5.1 Identification of monochrome and color QR codes</td>
<td>45</td>
</tr>
<tr>
<td>5.2 Spatial randomization and block accumulation</td>
<td>46</td>
</tr>
<tr>
<td>5.3 Model selection for color QR codes</td>
<td>48</td>
</tr>
<tr>
<td>6 Experimental Evaluation</td>
<td>49</td>
</tr>
</tbody>
</table>
6.1 Performance metrics .................................................. 50
6.2 CUHK-CQRC: a large-scale color QR code dataset .................. 51
6.3 Evaluations on CUHK-CQRC dataset ................................... 53
   6.3.1 Evaluation of HiQ on CUHK-CQRC ............................... 53
   6.3.2 Evaluation of model selection ..................................... 56
6.4 Evaluations on mobile devices .......................................... 57
   6.4.1 Evaluation of HiQ on mobile application ......................... 57
   6.4.2 Pipeline analysis .................................................. 60
   6.4.3 Analysis of the layer-density trade-off ......................... 62
   6.4.4 Evaluation of randomization ..................................... 63

7 Related Works ............................................................... 66
   7.1 Monochrome QR code ................................................. 66
   7.2 Color 2D barcodes .................................................. 67

8 Conclusion and Future Work .............................................. 70

Bibliography .................................................................. 76
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Examples of different types of 2D barcodes.</td>
<td>1</td>
</tr>
<tr>
<td>2.1</td>
<td>Structure of QR code symbol.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Decoding process of a monochrome QR code using mobile phone.</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of different types of color 2D barcode.</td>
<td>11</td>
</tr>
<tr>
<td>3.1</td>
<td>An overview of the encoding and decoding of a 3-layer color QR code in HiQ.</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>HiQ first encodes the user data into 3 monochrome QR codes with the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dimension, followed by color encoding where a predefined color codebook is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>used to combine the 3 monochrome QR codes into one color QR code, and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>painting patterns into special colors. For decoding, after correcting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>geometric distortion and performing color recovery, HiQ uses the same</td>
<td></td>
</tr>
<tr>
<td></td>
<td>color codebook during encoding to decompose a color QR code into 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>monochrome QR codes, and extracts the embedded data using existing method.</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Examples of color QR codes of different layers with the same content size.</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>All layers are protected with low level of error correction.</td>
<td></td>
</tr>
</tbody>
</table>
3.3 RGB color space. ......................................................... 18
3.4 Pattern coloring: color the function patterns of a 3-layer color QR code. .............. 21
3.5 Cross-channel interference: comparison between ground truth and the all layers
directly extracted from a 3-layer color QR code after capturing. ...................... 24
3.6 Chromatic distortion of 3-layer color QR codes. ........................................ 25
3.7 Decoding process of a color QR code using mobile phone. Decoding of a color QR
code differs from decoding a monochrome QR code in the following components:
binarization, geometric transformation and color recovery, which are highlighted
in the yellow blocks. .......................................................... 27
3.8 Using existing method to binarize an image of a color QR code. ...................... 28
3.9 The estimation of the geometric transformation matrix. In the left is the effect of
adding more labeled points in estimating the geometric transformation, and in the
right is the effect of shifting one labeled point (17 labeled points in total) from the
correct location. The module size of the color QR code under testing is approxi-
mately 17 pixels. ............................................................. 29

4.1 The visualization of color normalization. In the first row are two images of the
same color QR code but captured under different lighting conditions, fluorescent
(left) and incandescent (right); and the corresponding images in the second row are
the images after color normalization. ........................................ 35
5.1 color distributions sampled from images of two monochrome QR codes (black and green) and a 3-layer color QR code captured by the same device and under the same lighting condition. 45

5.2 Spatial Randomization of Data Bits in Each Block. The same color bits belongs to the same single error-correction block. 47

6.1 The phases of decoding process the each performance metric measures. 51

6.2 Samples from CUHK-CQRC captured under different lighting conditions. 52

6.3 Decoding results of HiQ-C, PCCC and PCCC with RGT on CUHK-CQRC. 54

6.4 Decoding results of HiQ on CUHK-CQRC with different estimated module sizes. 55

6.5 Decoding results of HiQ-C, PCCC and PCCC with RGT on CUHK-CQRC. 56

6.6 Performance of HiQ on iPhone 6 Plus and Nexus 5 across various content sizes, printout sizes and error correction levels, measured by the 90th percentile of the scanning time of 30 successful scans. Since we cannot generate a QR code beyond its maximal capacity, we set its scanning time to infinity. The first row and the second row are the results of iPhone 6 Plus and Nexus 5 respectively. 57

6.7 Scanning performance of HiQ-C across various content sizes, printout sizes and error correction levels. We collect the scanning time (in seconds) of 30 successful trials using iPhone 6 Plus and use the 90th percentile, 10th percentile and median to represent the upper bound, lower bound of the scanning time and the overall performance. 59
6.8 Comparison of scanning performance of randomized and original (non-randomized) color QR codes on mobile device. We collect the scanning time (in seconds) of 10 successful trails using Nexus 5, and use the average running time to represent the performance. When the print-size of 6100-bytes samples become bigger, it is difficult to capture the images under uniform illumination, which is the reason that 54mm and 56mm-size samples of 6100-bytes QR code have longer running time than smaller samples.

6.9 Comparison of scanning performance with respect of the block-accumulation behavior. The selected samples are those cannot be decoded without randomization. The accumulated blocks are the percentage of successfully decoded blocks accumulated by each frame.
## List of Tables

4.1  Color Prediction under Fluorescent Light Accuracy of Different Methods  . . . . . 39

4.2  Scanning Performance of QDA and Layered SVM on Mobile Applications . . . . . 41

6.1  Types of Smartphones Used in Collecting Database . . . . . . . . . . . . . . . . . 52

6.2  Execution Time of Basic Blocks in Pipeline of HiQ . . . . . . . . . . . . . . . . . 61

6.3  Comparison between 2-layer and 3-layer color QR codes . . . . . . . . . . . . . . 63
Chapter 1

Introduction

Figure 1.1: Examples of different types of 2D barcodes.

A barcode is a type of optical machine-readable data representation invented in late 1960s, and has been widely used for many decades. The barcode is originally designed for enabling machines to read the product information automatically, and thus has very small data capacity. After years of improvements, there have evolved many kinds of barcodes with attractive features, e.g., Aztec Code, Quick Respond (QR) Code and High Capacity Color Barcode [13], see Fig. 1.1 for examples. Compared with others, QR codes have gained the most popularity because of its quick response
CHAPTER 1. INTRODUCTION

to scanning, higher capacity in data encoding, robustness to damage, and readability from any directions.

However, the capacity of the QR codes limits its applicability, even though QR codes can store much more data than other barcodes do. Currently, QR codes are widely used for advertisement, where a Uniform Resource Locator (URL) is usually encoded. By one simple scan using the cellphone, the application immediately directs the user to an on-line website or service. Indeed, this way of using QR codes exempts users from the bother of typing the URL using the cellphone, but it also brings severe problems:

- **expensive**, it requires an on-line web service and a dataset deployed on a remote server.

- **inconvenient**, it requires the Internet access for those devices.

- **insecure**, since the data is transmitted behind the scene, it may expose users to malicious attacks and cause private information leakage without users’ awareness [15].

To address these problems, we propose to increase the data capacity in QR codes. With higher capacity, one can add authentication mechanism such as digital certificates to prevent information leakage. Besides, storing all the data in the QR code instead of in a web page URL also solves the aforementioned problems. Moreover, high-capacity QR codes have great application prospect and commercial value, and there have emerged many applications that require high-capacity QR codes, e.g., AuthPaper [17] and face recognition by QR codes [22]. Towards this end, we propose a layered framework for high-capacity QR codes, HiQ, where color is leveraged to increase the data capacity of traditional QR codes.
CHAPTER 1. INTRODUCTION

However, achieving both decoding robustness and high data capacity in QR codes is non-trivial. Although a traditional QR code can store 2933 bytes of data at maximum, previous experimental results indicate most of the decoders easily fail to decode the high-density QR codes. Specifically, to reliably decode a monochrome QR code with around 1800 bytes, the printout size\(^1\) should at least be 75 mm [17] which is too big to be practical. Besides, decoding color QR codes is a great challenge because of the chromatic distortion, especially for high-density ones. The chromatic distortion occurs due to the following facts: 1) cross-channel interference: colorants in each channel tend to interfere with each other [3]; 2) cross-module interference: for a high-density QR code, the colorants in the neighboring data modules (spatial regions) may spill over and distort the color; 3) illumination variation: color varies dramatically under different lighting conditions [8].

In order to overcome these challenges, firstly, we improve the decoding algorithm so as to robustly decode high-density QR codes by mobile devices. Secondly, in order to robustly decode color QR codes, we utilize a learning-based approach to model the lighting variation and interferences of colors. In particular, HiQ constructs a color QR code by combining multiple monochrome QR codes together in a layered manner, which preserves the structure of traditional QR codes and the strength of their design. Thirdly, a robust geometric transformation method is proposed to correct the geometric distortion for high-density QR codes. To decode a color QR code image, HiQ uses a traditional QR code decoder to extract data from each layer (monochrome QR code) after colors are recovered using a pre-trained color classifier.

The technical contributions of this work are summarized as follows:

\(^1\)In this work, we use the side-length of a square to represent the area of QR code footprint, i.e., the printout size.
1. **High-capacity QR code framework.** We propose a high-capacity QR code framework, HiQ, which not only increases the capacity of traditional QR codes, but also improves the robustness of high-density QR code decoding. The experimental results show the advantages of HiQ over the baselines. We also demonstrate the effectiveness of HiQ on mobile applications over a wide range of smartphones and mobile operating systems. In particular, HiQ can encode 2600 bytes, 4500 bytes and 7700 bytes of data in areas of $24 \times 24 \text{ mm}^2$, $34 \times 34 \text{ mm}^2$ and $50 \times 50 \text{ mm}^2$, respectively, and still guarantee robust decoding within 5 seconds.

2. **Large-scale color QR code dataset.** Previously, no public high-density color QR code dataset is available for benchmarking color QR code decoding algorithms. In this work, we create a challenging color QR code dataset, CUHK-CQRC\(^2\), which consists of 5390 samples of color QR codes captured by different mobile phones under different lighting conditions (see section 6.2 for detailed description). We believe many researches and applications can benefit from this dataset.

3. **Real working applications.** Our framework has been implemented and integrated into a practical mobile application, which has been released to the public via the Apple App Store\(^3\) and Google Play\(^4\).

\(^2\)available at [https://authpaper.net/colorDatabase.html](https://authpaper.net/colorDatabase.html).


Chapter 2

Background Study

Summary

This chapter presents the background of QR codes (in Sec. 2.1) and color 2D barcodes (in Sec. 2.2) which will be visited in the later chapters from time to time. In particular, we show in Sec. 2.1.1 how existing system encodes the user data into a monochrome QR code (black/white by default) and the main structure of a QR code. In Sec. 2.1.2 we show how the decoding algorithms work to decode an image of a QR code. Moreover, in Sec. 2.2 we introduce several existing color 2D barcodes and discuss what motivates the development of color QR codes.
2.1 Basics of QR codes

2.1.1 Data encoding of QR codes

Fig. 2.1: Structure of QR code symbol.

Fig. 2.1 and Fig.1.1(b) give the structure and an example of a traditional QR code, respectively. Unlike 1D barcodes, QR codes encode data bits using small squares arranged in a square grid, and thus are often referred as 2D barcodes. In a QR code, the small monochrome square is referred as a data module, and the number of data modules along one side of the square grid is called the dimension of the QR code, which ranges from 21 to 177. Different dimension corresponds to different version, and there are 40 versions of the QR code in total.

The larger the dimension/ version is, the more data the QR code can carry. Each binary data bit is represented by one module of the QR code, which usually has two easily distinguishable colors. Black and white are naturally the first choice because they are easy to distinguish and cheap to
print, but other colors are often used to increase the visual attractiveness of the QR code, see Fig. 1.1(c) for illustration.

During encoding, user data are distributed into a set of data blocks. Within each data block, a Reed-Solomon error correction is used to add redundant bits to protect the user data in the data block. Existing QR standards support four different levels of Reed-Solomon error correction, namely \textit{Low} (L), \textit{Medium} (M), \textit{Quartile} (Q), and \textit{High} (H), which can be selected by users during QR code generation. Note that a higher error correction level enables the QR code to tolerate more errors during decoding, but it also reduces the amount of data the QR code can carry. For example, a QR code with the L level of error correction can carry data up to 2953 bytes of user data, but with the H level of error correction, it can only carry 1273 bytes at most.

Besides the user data, inside a QR code there are several groups of important components shown in 2.1:

- Function patterns, which contain location information. During generation, QR codes form four specially designed spatial patterns in the four corners for quick localization during decoding. One is located in the bottom-right corner denoted as \textit{alignment pattern} and the other three are located in the other three corners denoted as \textit{finder patterns}. The reason we need an alignment pattern, which is different from a finder pattern, is for determining the rotation direction of the QR codes. When the dimension of the QR code gets larger, a QR code will form more alignment patterns in its internals, which are referred as \textit{internal alignment patterns};
• Timing pattern and version pattern, which contains version/dimension information. As is shown in 2.1, the timing pattern is composed of two lines of modules with pattern “··· – \textit{black} – \textit{white} – \textit{black} – ···”, intersecting the whole code vertically and horizontally;

• Quite zone, which separates the QR code with the background. Quite zone is mainly used to prevent spatial pattern of the finder pattern being changed because if the background color happened to be close to the finder pattern color the QR code will be unable to detect;

• Format pattern which contains the level of error correction applied in the QR code.

2.1.2 Decoding of QR codes

Fig. 2.2 gives an overview of the QR code decoding processing on a mobile device. In standard QR code decoding algorithm\(^1\), given a captured image of a QR code, the decoding procedure mainly consists of the following steps:

1. Binarization. The decoder firstly converts the input image to a gray image and binarizes the it into a black/white (binary) image using a local thresholding method. Namely, the gray image is divided into multiple small size-fixed patches (e.g., 8 × 8), and in each patch the algorithm computes the average intensity over all pixels in the patch which is used as a local threshold to binarize the each pixel in this patch (i.e., a pixel is classified as white is its intensity exceeds the threshold, black otherwise);

\(^1\)In our work, we refer the implementation of ZXing [19], an open-source library of QR code generator and decoder, as standard
2. Detection. The decoder locates the QR code in the binarized image by finding the function patterns (described in Sec. 2.1.1) in the four corners. The detection algorithm leverages the special spatial pattern in those function patterns to locate their positions. Notice that because of the special design of the function patterns each finder pattern (alignment pattern) has a unique property: no matter how the QR code rotates, draw a horizontal or vertical line segment across the finder pattern (alignment pattern) through its center, the color of the pixels on the line follows a special length ratio 1:1:3:1:1 (1:1:1:1:1). Therefore, by detecting these special length ratios, the decoder can locate the finder patterns and alignment patterns. If multiple function patterns are detected which may contain false positives, then we choose the four points that form a larger square than every other group of four points. Besides, in this stage, the dimension is jointly determined by reading the timing pattern and the version pattern. Particularly, the algorithm firstly estimates the dimension from the timing pattern, which may be wrong but close to the true answer, and
CHAPTER 2. BACKGROUND STUDY

calculate a geometric transformation matrix based on the estimated dimension. Secondly, using
the geometric transformation matrix the algorithm then reads the version pattern and verifies the
correctness of the previously estimated dimension, if wrong, then the geometric transformation
matrix will be re-calculated using the dimension obtained from the version pattern.

3. Geometric transformation. Afterwards, by using the positions of the detected finder patterns and
alignment patterns, a geometric transformation matrix is constructed to correct the geometric
distortion and reconstruct the original QR code. To be specific and mathematical, given the
coordinates of four points (positions of the function patterns) denoted as \((x_i, y_i)\) where \(i \in \{1, 2, 3, 4\}\), and the corresponding points \((x'_i, y'_i)\) in the bit matrix whose positions are known
if provided the dimension of the QR code (e.g., the top-left finder pattern is \((3.5, 3.5)\)), we can
formulate two equations using one pair of corresponding points as follows:

\[
\begin{align*}
-h_4x_i - h_5y_i - h_6 + h_7y'_i x_i + h_8y'_i y_i + y'_i &= 0 \\
h_1x_i + h_2y_i + h_3 - h_7x'_i x_i - h_8x'_i y_i + x'_i &= 0
\end{align*}
\]

(2.1)

Therefore, we obtain the transformation matrix \(H = \{h_1, h_2, \cdots, h_8\}\) by solving a linear system
with eight unknowns \((h_1, h_2, \cdots, h_8)\) and eight equations. Then the bit matrix is reconstructed
by mapping each point to the binary image by solving Eq. (2.1).

4. Error correction. Having the reconstructed bit matrix, the data then are extracted from each data
block and parsed into the original message after applying the error correction using the built-in
Reed Solomon code whose level of error correction is read from the format pattern. Note that the decoding process succeeds if and only if in all data block the data are recovered from errors.

### 2.2 Color barcodes

![Color barcode examples](image)

Figure 2.3: Examples of different types of color 2D barcode.

As we introduced in Sec. 2.1, traditional QR code, as a data-carrying media, can only store very limited amount of data. In particular, using low level of error correction a QR code can 2,953 bytes of data at maximum which is relatively low for many applications, e.g. in protecting the authenticity of a QR code [17] and in supporting face recognition using QR codes [22].

To increase the data capacity of QR codes, leveraging color is probably the most natural and inexpensive approach because color printers have been widely used and easy to access, and adding color in the barcodes rarely affects the functioning of other components of the existing barcodes, therefore it is easy to adopt for existing applications. Color 2D barcodes are also referred in some literature as 3D barcodes where color provides another data dimension. In recent years, there have been many studies on using color to increase the capacity of traditional 2D barcodes, e.g., Microsoft
High Capacity Color Barcode (HCCB) [21], High capacity color 2D barcode (HCC2D) [9], per-colorant-channel color (PCCC) [3] and Paper-Memory Code (PMC), see Fig. 2.3 for illustration.

We will review and discuss these color 2D barcodes in Sec. 7.2.

☐ End of chapter.
In this chapter, we describe the proposed framework for high-capacity color QR codes, HiQ, which consists of two major parts, encoding and decoding. Firstly, in Sec. 3.1 we start with an overview of the encoding and decoding framework of color QR codes in HiQ, as shown in Fig. 3.1. Secondly, we present the data encoding scheme, the challenges of robustly decoding a high-density color QR code and the decoding algorithm in detail in Sec. 3.2, Sec. 3.3 and Sec. 3.4, respectively.
3.1 Framework overview

Figure 3.1: An overview of the encoding and decoding of a 3-layer color QR code in HiQ. HiQ first encodes the user data into 3 monochrome QR codes with the same dimension, followed by color encoding where a predefined color codebook is used to combine the 3 monochrome QR codes into one color QR code, and painting patterns into special colors. For decoding, after correcting geometric distortion and performing color recovery, HiQ uses the same color codebook during encoding to decompose a color QR code into 3 monochrome QR codes, and extracts the embedded data using existing method.

In this thesis, we propose a layered framework for high-capacity QR code, HiQ, where our contribution in increasing the data-carrying rate of QR code is twofold. Firstly, built upon traditional 2D QR code, HiQ improves the decoding algorithm so as to robustly decode high-density QR code by mobile devices while also preserving the merits of traditional QR code. Secondly, HiQ goes beyond 2D QR code, using color as the third data dimension which turns multiple monochrome QR codes into one color QR code (also known as 3D QR code). Fig. 3.1 gives an overview of encoding and decoding processing of a color QR code under the framework of HiQ.

In the encoding part of HiQ, a color QR code is constructed by combining multiple monochrome QR codes together in a layered manner and different color is used to represent different combina-
tion of overlapping binary data modules. To decode a color QR code in the captured image, HiQ uses traditional QR code decoder to extract data from each layer (monochrome QR code) before which the color recovery is applied. Particularly, as a single-layer color QR code with the color set to be \{black, white\}, traditional monochrome QR code is a special case under the framework of HiQ.

### 3.2 Data encoding

#### 3.2.1 Encoding scheme

In traditional QR codes, one module represents one bit of data because each module can only be one of the two colors, black or white (by default), representing 1 or 0 in the data bit. Therefore, the more the number of color each module can be of the more the number of data bits each module can carry. Based on this, we use more colors in constructing the color QR codes instead of just two colors in monochrome QR codes to increase the capacity.

In order to exploit and reuse existing QR code standards and tools, we maintain the structure of the traditional QR code in our color QR code by stacking multiple (traditional) monochrome QR codes together and using a highly discriminable set of colors to combine them into one color QR code. In particular, HiQ partitions the user data into multiple small pieces and encodes them into different monochrome (black/white) QR codes independently, after which HiQ uses different colors that are easily distinguishable to represent different sets of overlapping black/white modules. In the sequel, we introduce the notation of layer to denote each monochrome QR code in HiQ. Note
Figure 3.2: Examples of color QR codes of different layers with the same content size. All layers are protected with low level of error correction.

that different layers of monochrome QR codes can have different levels of error correction, but they have to be of the same dimension in order to maintain the structure of traditional QR code in the HiQ codes.

Fig. 3.1 illustrates the construction of a 3-layer color QR code. Given $n$ monochrome QR codes, $\{M_i\}$, where $i = 1, 2, \cdots, n$, each $M_i$ is composed of the same number of small squares (referred as modules) with two different colors (e.g., white and black) which represent one data bit, 0 and 1. We denote the $j$th module of $M_i$ by $m^i_j$, where $m^i_j = 0$ or 1. In order to achieve layer independence and separability in the color QR code, HiQ constructs an $n$-layer color QR code $C_n$ by concatenating all $M_i$ together so that the $j$th module of $C_n$, $c^n_j = \{m^n_1, m^n_2, \cdots, m^n_j\}$. HiQ then maps each $c^n_j$ into a particular color using a predefined color codebook $\mathbb{B}$, where $|\mathbb{B}| = 2^n$ as $m^n_j$ is binary.

Using $2^n$ different colors, HiQ constructs a $n$-layer color QR code whose capacity is increased by $n - 1$ times compared with the traditional monochrome QR code of the same dimension. Alternatively, given the same amount of data to carry (within the capacity of a monochrome QR
code), HiQ consumes much less substrate footprint in the print media than the traditional QR code does, if printed in the same density. Particularly, when $n = 1$, our color QR code degrades to the monochrome QR code, and each module carries one bit of (binary) data. Moreover, PCCC [3] is also a special case under our framework with $n = 3$ and a particular color codebook whose colors are black, white, red, green, blue, cyan, magenta, and yellow. In fact, HiQ is a framework that provides users and developers with more flexibilities in generating the QR code in terms of data capacity, embedded error correction level and appearance (color), etc. Fig. 3.2 gives examples of HiQ color QR codes of different layers ranging from 1 to 4. One can observe that given the same amount of user data the color QR code with more layers is less denser than the ones with less layers. However, note that the use of more layers also increases the number of colors in the color QR codes and thus increases the difficulties in decoding (mainly, in color recovery). Therefore, given that same amount of user data, there exists a trade-off between the number of layers and density which will be investigated in Sec. 6.4.3.

### 3.2.2 Color selection

To make the color QR codes easier to decode, it is critical to select a set of colors that can be easily differentiated from each other. For monochrome (single-layer) QR codes, black and white are naturally the best choice because they are cheap to print and easy and robust to distinguish under various lighting conditions. However, since the number of color $m$ grows exponentially with the number of layers $n$, it becomes extremely difficult to choose a suitable set of colors which are easy and robust to recognize when $n > 3$ (i.e., $m = 16, 32, \cdots$). In this work, we focus on the
cases when \( n \leq 3 \).

Suppose we represent the colors in the normalized RGB color space where each color is represented by a 3-dim RGB intensities vector whose elements are within \([0, 1]\) (see Fig. 3.3 for illustration), our color selection method is to choose a set such that the minimal Euclidean distance between any two colors among the color set is maximized. Let \( C_n \) denote a color set for \( n \)-layer color QR codes where \( |C_n| = 2^n \), \( S \) be the RGB space, \( C_n^* \) be the optimal color set, and \( c_i \) be the \( i \)th color vector of \( C_n \), then we have

\[
C_n^* = \arg \max_{C_n \subseteq S} \min_{c_i, c_j \in C_n, i \neq j} D(c_i, c_j),
\]

(3.1)

where \( D(x, y) \) denotes the Euclidean distance between vector \( x \) and \( y \).
CHAPTER 3. HIQ: A LAYERED FRAMEWORK FOR HIGH-CAPACITY QR CODES

For the cases where \( n = 2 \) or 3, where we need to select 4 or 8 colors from the RGB cube, we have the following theorem:

**Theorem 1.** When \( n = 2 \), the optimal color set is \( C_2^* = \{\text{white, blue, red, green}\} \), where the distance of every two colors is \( \sqrt{2} \); and when \( n = 3 \), the optimal color set is \( C_3^* = \{\text{white, blue, red, green, black, cyan, magenta, yellow}\} \), where the minimal distance between two colors is 1.

**Proof.** Let \((x_i, y_i, z_i)\) denote the coordinates of the \( i \)th color a color set, where \( x_i, y_i, z_i \in [0, 1] \) and \( i = \{1, 2, \cdots, N\} \) \((N = 2^n)\). Given a fixed color set \( C_n \), we have

\[
\min_{c_i, c_j \in C_n, i \neq j} D(c_i, c_j) \leq \frac{1}{\binom{N}{2}} \sum_{c_i, c_j \in C_n, i \neq j} D(c_i, c_j) \leq \frac{1}{\binom{N}{2}} \sum_{c_i, c_j \in C_n, i \neq j} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \leq \frac{1}{\binom{N}{2}} \sqrt{\sum_{c_i, c_j \in C_n, i \neq j} (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}.
\]

(3.4)

Note that each coordinate \( x_i, y_i, z_i \) can be treated equally in Eq. (3.4), in order to simplify the analysis, we only consider \( x_i \) at this stage. Without loss of generality, we let \( 0 = x_1 \leq x_2 \leq \cdots \leq x_N = 1 \). When \( n = 2 \), we have \( N = 4 \) and

\[
\sum_{c_i, c_j \in C_n, i \neq j} (x_i - x_j)^2 = (x_2^2 + (x_2 - 1)^2) + (x_3^2 + (x_3 - 1)^2) + (x_2 - x_3)^2 + 1 \leq 1 + 1 + 1 + 1 = 4
\]

(3.5)

Similarly, we can show that \( \sum_{c_i, c_j \in C_n, i \neq j} (y_i - y_j)^2 \leq 4 \) and \( \sum_{c_i, c_j \in C_n, i \neq j} (z_i - z_j)^2 \leq 4 \). Together
with Eq. (3.4), we show that 
\[
\min_{c_i, c_j \in C_2, i \neq j} D(c_i, c_j) \leq \sqrt{2},
\]
where the equality can be achieved if the color set \( C_2 = \{(x_i, y_i, z_i)\}_1^4 \) satisfies the following property:

**Property 1.** Half of the \( x \) coordinates, i.e., \( x_i \), equal to 0 and the other half equal to 1, so are the \( y \) and \( z \) coordinates, for example, \( \{(1, 1, 1), (1, 0, 0), (0, 0, 1), (0, 1, 0)\} \) when \( n = 2 \).

Therefore, only two sets satisfy this condition, which are \{white, blue, red, green\} and \{black, cyan, magenta, yellow\}. However, white color is the default color of the paper substrate, so the former one is the only optimal color set when \( n = 2 \).

When \( n = 3 \), it is straightforward that the optimal set is \{white, blue, red, green, black, cyan, magenta, yellow\}, whose minimal pairwise distance is 1, because the eight points form the largest polyhedron in the RGB cube. This completes the proof.

After the color set \( C_n \) is fixed, HiQ uses a pre-defined color codebook (Fig. 3.1 gives an example of the color codebook for \( n \)-layer color QR codes) is used to map each color code in \( C_n \) to a specific color during the generation of the color QR codes.

**Remark 1.** The presented color selection seems to work in RGB color space, one catch is if we use another color space, the so-called optimal set may not be optimal any more. To solve this problem, actually we can form a similar optimization problem as Eq. 3.1 in the new color space and find the solution using the similar techniques.
Remark 2. Another problem is, we found that the color distribution of a captured color QR code image was severely distorted compared to the one before printing (see Fig. 3.6(a) for illustration), so even if we found a optimal color set in a given color space, this optimal color set may not be optimal in the color space after printing and capturing (the color space where decoding algorithm operates). However, we do not know how color transforms during printing and capturing, therefore, we are not able to apply the above techniques directly in this case. Nonetheless, one possible solution is to approximate and learn the transformation function $f(c_i)$ from real-operating data, and solve $\max_{C_n \subseteq S} \min_{c_i, c_j \in C_n, i \neq j} D(f(c_i), f(c_j))$. We are not going to investigate it in this work and leave it as future extension.

3.2.3 Pattern coloring

At this stage, the encoding of the user data in a color QR code is completed. However, we further paint the special patterns with specific colors which not only function as the reference color (whose colors are known beforehand) to facilitate the color recovery process or carry useful
CHAPTER 3. HIQ: A LAYERED FRAMEWORK FOR HIGH-CAPACITY QR CODES

information, such as the number of layers of the color QR code, which will be useful in the decoding phase. Fig. 3.4 gives an illustration of pattern coloring of a 3-layer color QR code.

In our design, for a 2-layer QR code, we paint each finder pattern with a different color, thus together the white color contained in each pattern, it is enough to provide four reference colors. While in a 3-layer QR code, each finder pattern contains two different colors (as is shown in Fig. 3.3.2) so as to provide reference data for all eight colors (white colors in the white region of each pattern and black in the bottom-right alignment pattern). Moreover, In both 2- and 3-layer QR codes, if internal alignment patterns exist, we sequentially and recursively paint all of them with all colors except for white.

3.3 Challenges of Decoding Color QR Codes

Before diving into the decoding process of color QR codes, we first discuss the challenges in decoding a color QR code. From our preliminary experimental results, we found that the difficulties of increasing the capacity/footprint ratio come from two aspects:

- **high density**, which makes the decoding extremely sensitive to geometric transformation errors;
- **chromatic distortion**, which brings huge challenges for color recovery.

In the following, we discuss these two major challenges one by one in detail.
3.3.1 High density

Even for monochrome QR codes, although various QR code decoding algorithms have been used in the real-world applications for many years, robust and fast decoding for high-density QR codes on mobile phones is still an open problem.

The first issue is the inaccurate geometric transformation. The existing decoding algorithm corrects the geometric distortion using perspective projection [10] estimated from 4 detected special patterns in the 4 corners of the QR code. In practice, inaccurate corner detection and shift in the estimated position are inevitable. While a shift of position of a few pixels may be tolerable during the decoding of low-density QR codes, minor errors (more than 5 pixels) in estimating the position of the special patterns can be amplified in data matrix reconstruction of high-density QR codes. This is because high-density QR codes usually have much smaller module size (number of pixels for each square module) in the captured image when compared to the lower density ones.

The second issue is the difficulty in correctly detecting alignment patterns. The small module size of high-density QR codes also decreases the success rate of detecting the alignment pattern which is critical for localization. When taking images using mobile devices, camera position may be shifted during the shoot, which may cause mild motion blur or out-of-focus blur in the captured images. Such mild blurring causes mis-detection of the alignment pattern, leading to failure in QR code decoding. Moreover, for high-capacity QR codes, data modules have a higher chance to be similar to alignment patterns, which often leads to faulty detection.
3.3.2 Chromatic distortion

Color brings extra data capacity for QR codes, but it also brings tremendous challenges to the decoding because of color interference and illumination variation, especially for high-density QR codes. Moreover, the more the number of colors used in the color QR code, the harder the decoding will be.

Fig. 3.5 and Fig. 3.6 give illustrations of the three types of chromatic distortion of the color QR codes (mentioned in Sec. 1) where eight colors, *black, white, red, green, blue, cyan, magenta, and yellow*, are used. Fig. 3.6(b) depicts the variation of distribution of the color values in the RGB space under different lighting conditions. This shows that the same color differs a lot under different lighting conditions, which brings tremendous challenge to the decoding of color QR codes.
In addition, Fig. 3.6(a) shows that for high-density color QR codes the colorants in each module spill over to its adjacent modules contaminating the colors of each other, especially in the edges of each module. Furthermore, existing decoding algorithms estimate the transformation matrix by detecting four special patterns in the corners of a QR code as described in Sec. 2.1.2. However, in practice, the detection is inevitably inaccurate, and the cross-module interference makes the decoding more sensitive to transformation errors caused by the inexact detection, especially for a high-density QR code. Consequently, the decoding of high-density color QR code fails easily.

Figure 3.6: Chromatic distortion of 3-layer color QR codes.
3.4 Data decoding

The goal of the decoding process is to extract user data embedded in the QR codes. As an example, an overview of decoding process of a 3-layer color QR code is given in Fig. 3.7. Typically, given an image of a monochrome/color QR code captured by a cellphone camera, HiQ first binarizes it for detecting the special patterns which are needed for determining the boundary and orientation of the QR code. The patterns also play a critical role in correcting geometric distortion of the QR code image taken by the camera. Before decoding, HiQ will read the number of layers embedded in the patterns during creation. If the QR code is multiple-layer (color), HiQ will apply color recovery and use a predefined color codebook to map the color QR code into multiple monochrome QR codes. Finally, HiQ uses the existing monochrome QR code decoding algorithm to extract the data layer by layer using the built-in error correction mechanism.

3.4.1 Color QR code detection

To locate a color QR code in a captured image, directly apply the detection method of traditional QR code (see Sec. 2.1.2) is not feasible. As existing monochrome QR code decoders usually use the image luminance, e.g., the Y channel of the YUV color space, to binarize the QR code, directly applying it on color ones is problematic because some colors (e.g., yellow) have much lower luminance than other colors, which results in some patterns being undetectable (see Fig. 3.8 for illustration). To solve this problem, we use a simple but effective method, local binarization, to binarize the color QR code, after which we can apply the existing detection algorithm to locate the
function patterns.

**Local binarization:** Let $I$ denotes an image of the color QR code formatted in the RGB color space. We first equally divide it into $8 \times 8$ blocks. In each block, a threshold is computed for each channel of the RGB space as follows:

$$T_i = \frac{\max(I_i) + \min(I_i)}{2}$$

where $i \in \{R, G, B\}$ and $I_i$ is the $i$th channel of image $I$. A pixel denoted by a triplet $(P_R, P_G, P_B)$ is assigned 1 (black) if $P_i < T_i$ for any $i \in \{R, G, B\}$, 0 (white) otherwise.
3.4.2 Robust geometric transformation

The existing decoding algorithm corrects the geometric distortion using perspective projection [10] estimated from four detected special patterns in the four corners of the QR code. In practice, inaccurate corner detection and shift in the estimated position are inevitable. While small shift of position of a few pixels may be tolerable during the decoding of low-density QR codes, minor errors (more than 5 pixels) in estimating the position of the special patterns can be amplified in data matrix reconstruction of high-density QR codes. This is because high density QR codes usually have much smaller module size (number of pixels for each square module) in the captured image when compared to the lower density ones. Moreover, the cross-module interference (see Fig. 3.6(a) for illustration) makes the decoding more sensitive to transformation errors caused by the inexact detection.

To solve these problems, we look into the QR code standards which provide more internal alignment patterns. We find that using more points to calculate the geometric transformation can reconstruct a more accurate QR code matrix, see Fig. 3.9(a) for experimental results. Therefore,
Figure 3.9: The estimation of the geometric transformation matrix. In the left is the effect of adding more labeled points in estimating the geometric transformation, and in the right is the effect of shifting one labeled point (17 labeled points in total) from the correct location. The module size of the color QR code under testing is approximately 17 pixels.

Instead of developing a more complex detection algorithm which increases processing latency, we circumvent this problem by proposing a robust geometric transformation (RGT) algorithm, which accurately samples for each module a pixel within the central region where the color interference is less severe than that along the edges of a module. Unlike standard methods, RGT harnesses all special patterns, including the internal ones, and solves a weighted over-determined linear system to estimate transformation matrix.

Given \( n \) tuples each consists of a pair of 2D data-points, namely, \( \{ < x_i, x'_i >, i = 1, 2, \cdots, n \} \), where \( x_i \) is the position of a detected pattern, and \( x'_i \) is the corresponding point in the data matrix to be reconstructed. In perspective projection [10], \( x_i \) is the homogeneous coordinate representation \((x_i, y_i, z_i)\) where we empirically choose \( z_i = 1 \), and each pair of corresponding points gives two
linear equations:

\[ A_i H = 0, \quad (3.7) \]

where \( H \) is the transformation matrix to be estimated and

\[
A_i = \begin{bmatrix}
0^T & -x_i^T & y'_i x_i^T \\
x_i^T & 0^T & -x'_i x_i^T
\end{bmatrix}.
\]

Note that although \( H \) has 9 entries, \( h_1, h_2, \cdots, h_9 \), since in 2D homography \( H \) is defined up to scale, and thus has eight degrees of freedom and one may choose \( h_9 = 1 \). Therefore, four point coordinates give eight independent linear equations as Eq.(3.7) which are enough for estimating \( H \).

However, since the estimated positions of the special patterns often contain noise, which implies \( A_i H \neq 0 \), RGT regards the norm \( \|A_i H\|_2 \) as the transformation error and minimizes a weighted error sum to obtain the estimation of \( H \):

\[
\minimize_{H} \sum_{i=1}^{n} w_i \|A_i H\|_2 \\
\text{subject to } \|H\|_2 = 1.
\]

(3.8)

where \( w_i \) is the weighting factor of each input point \( x_i \). Instead of arbitrarily fixing one \( h_i \), we add constraint \( \|H\|_2 = 1 \) to avoid \( H = 0 \). Based on our finding that the estimated positions of finder patterns are often more accurate than that of alignment patterns, we assign higher weights to detected positions of finder patterns and lower weights to alignment patterns. Empirically, we set
$w_i = 0.6$ if $x_i$ is from the finder pattern, $w_i = 0.4$ otherwise. Note that solving (3.8) is equivalent to solve the following unconstrained optimization problem:

$$\min_H \frac{\|AH\|_2}{\|H\|_2} \quad (3.9)$$

where $A$ is a matrix built from $\{w_iA_i | i = 1, \ldots, n\}$, and each $w_iA_i$ contributes two matrix rows to $A$. Fortunately, the solution to Eq. (3.9) is just the corresponding singular vector of the smallest singular value [10]. Singular value decomposition (SVD) can be used to solve this problem efficiently.

RGT is robust to minor shift in the detected positions, but not false positives. Fig. 3.9(b) shows that one false positive of the pattern detection, which is several module sizes away from its real location, can destroy the whole geometric transformation. In order to reduce the false positive rate of alignment pattern detection for the color QR codes, we leverage the color property by adding color constraints for pattern detection. For each pattern, we paint it using a specific color in the encoding phase (see Sec. 3.2) and during the detection, we filter out the false detections by checking whether the color of it is correct or not. This approach significantly reduces the false positive rate of the alignment pattern detection.

### 3.4.3 Color recovery

As we discussed in Sec. 3.3.2, due to the cross-channel color interference, directly applying traditional QR code decoding method on all layers (monochrome QR codes) from the color QR codes,
see Fig. 3.5 for illustration. Therefore, it is necessary to cancel the color interference and recover the color before we apply traditional QR code decoding on each layer.

For a monochrome QR code, a simple thresholding method is enough for the color recovery since after all there are only two colors between which the chromatic contrast is often high. However, color recovery for color QR codes which may consist of 4, 8, 16 or even 32 colors, becomes non-trivial. In our approach, we collect real-world operating data of color QR codes scanning for training a color classifier off-line, and apply it for color recovery during each the decoding process. After color recovery, we mapping a $n$-layer color QR code into $n$ monochrome QR codes using the predefined color codebook used in encoding (see Fig. 3.1 for example), and rely on the embedded error correction code to further recover the errors and decode the user data. We will discuss about the color recovery method in detail in Sec. 4.
Chapter 4

Robust Color Recovery for Color QR Codes

Summary

This chapter describes how we handle the three types of chromatic distortion in color QR code decoding. In Sec. 4.1, we introduce the proposed leaning-based color recovery method, and in Sec 4.2 a novel way of Support Vector Machine training scheme, layered strategy, is proposed to learn multiple-class SVMs for color QR codes. Then in Sec. 4.3 we describe how we collect the training data and train the color classifier.
4.1 Learning-based color recovery

Fig. 3.6(b) depicts that the variation of distribution of the color values in the RGB space under different lighting conditions. Such variations give rise to the so-called color constancy [6] problem which has been (and continued to be) an active area of computer vision research. However, most existing algorithms for achieving color constancy tend to be computation intensive and thus not viable for our application of color QR code decoding using off-the-shelf smartphones. To tackle this problem, we first normalize the color values of each sampled pixel by leveraging the structure of QR codes to make the color less illumination-sensitive. We then use a color classifier trained off-line to model the chromatic distortion using the normalized color values as inputs.

**Color Normalization:** Fig 4.1 visualizes the color normalization effects. Given a captured image of a n-layer color QR code, we first estimate the white RGB values, \( W \), of the captured image from the fixed white regions in the QR codes (e.g., along the boundaries, i.e., quiet zones, and within the special patterns). We denote by \((x, y)\) a pixel sampled by RGT, where \(x\) is the color feature and \(y = 1, 2, \cdots, n\) is the color label. Instead of directly using RGB values, denoted by \(I = \{I_R, I_G, I_B\}\), of each pixel as the color feature for color recovery, we normalize \(I\) by \(W\):

\[
x_j = \frac{I_j}{W_j}, \quad j \in \{R, G, B\}.
\]

For classification tool, we evaluate a few off-the-shelf machine learning techniques (see Table 4.1 for details) and adopt *quadratic discriminant analysis (QDA)* [28] as the color classifier for benchmark, and a more customized method is discussed in Sec. 4.2. QDA is an effective multi-
class classifier and has been well studied [28]. The decision rule of QDA is given as $G(x) = \arg\max_k \delta_k(x)$, where $x$ is a feature vector, $k$ is the class index and

$$\delta_k(x) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k) + \log \pi_k$$  (4.1)

In Eq.(4.1), $\Sigma_k$, $\mu_k$ and $\pi_k$ are parameters which can be learned from the training data [11].
4.2 Layered strategy

4.2.1 Methodology

Intuitively, color recovery can be treated as a traditional multi-class classification problem, where one color represents one class. And as the most commonly used classification tool in many computer vision tasks, SVM can be used as a color classifier. To train a multi-class SVM, one-vs-one and one-vs-all [12] are the most commonly used strategies. However, this approach has two major drawbacks:

- **Large processing latency as the number of layers increases.** For a QR code scanning application running on a mobile device where computational resources are very limited and tens of thousands of predictions need to be made within hundreds of milliseconds, even the prediction cost of the one-vs-all strategy is too expensive. Taking a $n$-layer HiQ code as example, there are $2^n$ colors, i.e., the color recovery is a $2^n$-class classification problem. Consequently, using one-vs-all strategy one has to train $2^n$ binary SVMs, and $2^{2n}$ binary SVMs if using one-vs-one strategy, so when $n$ becomes larger the scanning latency increases dramatically.

- **Unfavorable equality between classes.** Treating color recovery as a traditional multi-class classification task means each color is treated equally, namely, the penalties of misclassifying any two classes are the same, which is unfortunately unfavorable in HiQ codes decoding. For a 3-layer HiQ code, there are eight colors in total of which each represents a unique combination of three binary bits in three layers, for example, color $1 = (0, 0, 0)$, color $2$
= (1, 0, 0), color 3 = (0, 1, 0), color 4 = (1, 0, 1), etc. If we incorrectly classify color 1 to color 2, we lost one bit of data; while if we misclassify color 2 to color 3 and color 3 to color 4, we lost two and three bits of data, respectively. Therefore, we prefer the predictions that result in less bits loss.

To tackle these problems, we propose the \textit{layered strategy} whose prediction cost scales \textit{linearly} with the number of layers. In a nutshell, we train a binary SVM for each layer independently to predict the bit in the corresponding layer. In the sequel, we refer it as \textit{layered SVM} in this work. For \textit{n}-layer color QR codes the training data are denoted as \{\mathcal{X}, \mathcal{Y}\}, where \mathcal{X} and \mathcal{Y} are sets of normalized RGB values and binary \textit{n}-tuples (e.g., \(|\{1, 0, \cdots, 0\}| = n\)), respectively. Different from traditional SVM classifier which treats \mathcal{Y} just as color indicators and uses the one-vs-all strategy to train \(2^n\) binary SVMs on \{\mathcal{X}, \mathcal{Y}\} as there are \(2^n\) colors, we form \(\mathcal{Y}_1, \mathcal{Y}_2, \cdots, \mathcal{Y}_n\) by separating each element in \(\mathcal{Y}\) into \textit{n} binary indicators, and train \textit{n} different SVMs by using \{\mathcal{X}, \mathcal{Y}_1\}, \{\mathcal{X}, \mathcal{Y}_2\}, \cdots, \{\mathcal{X}, \mathcal{Y}_n\} as training data by solving \textit{n} optimization problems as follows:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|\omega_j\|^2 + C \sum_{i=1}^{N} \xi_i \\
\text{s.t.} & \quad \omega_j^T x_i + b_j \geq 1 - \xi_i \quad \{x_i, y_i = 1\} \in \{\mathcal{X}, \mathcal{Y}_j\}, \\
& \quad \omega_j^T x_i + b_j \leq -1 + \xi_i \quad \{x_i, y_i = 0\} \in \{\mathcal{X}, \mathcal{Y}_j\}, \\
& \quad \xi_i \geq 0, \quad 1 \leq i \leq N
\end{align*}
\]

where \(1 \leq j \leq n\) and \(N\) is the number training data. So given a testing sample \(x\), we use all \textit{n}
SVMs to output the predicted $n$-tuple, $\hat{y} = \{\hat{y}_1, \cdots, \hat{y}_n\}$, where

$$\hat{y}_j = \text{sign}(\omega_j^T x + b_j).$$  \hfill (4.5)

Layered SVM has two major advantages over methods in color QR code decoding:

- **Low processing latency.** By taking advantages of the structure in the feature and label, layered SVM only need $n$ binary SVM classifiers for $n$-layer color QR codes which compared with the one-vs-all strategy requiring $2^n$ binary SVM classifiers is a huge improvement regarding processing latency. Specifically, layered SVM reduces the time needed to perform color recovery by $1 - \frac{n}{2^n}$ (e.g., 67.5 for 3-layer color QR codes) compared to one-vs-all SVM.

- **Layer separability.** In the layered strategy, the classifications of all layers are completely independent. Thus, we can apply explicit mapping only on the weak layers to improve the performance without much sacrifices in the processing speed. And in a sequence of scanning, once a layer is decoded it won’t need to be processed anymore, which further saves unnecessary computations. In contrast, this cannot be achieved in one-vs-all SVM as all layers are coupled together and each prediction is jointly made by all $2^n$ binary SVMs.

### 4.2.2 Preliminary evaluation

We conduct experiments on real-operating color data (one million pixel samples) to compare our layered SVM with several off-the-shelf classification methods which include one-vs-all SVM,
Table 4.1: Color Prediction under Fluorescent Light Accuracy of Different Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Average</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layered SVM (linear kernel)</td>
<td>99.65%</td>
<td>99.34%</td>
<td>95.93%</td>
<td>98.31%</td>
<td>1</td>
</tr>
<tr>
<td>One-vs-all SVM (linear kernel)</td>
<td>98.28%</td>
<td>99.29%</td>
<td>96.84%</td>
<td>98.14%</td>
<td>2.7</td>
</tr>
<tr>
<td>Layered SVM (RBF kernel)</td>
<td>99.71%</td>
<td>99.44%</td>
<td>98.15%</td>
<td>99.10%</td>
<td>inf</td>
</tr>
<tr>
<td>One-vs-all SVM (RBF kernel)</td>
<td>99.62%</td>
<td>98.32%</td>
<td>97.99%</td>
<td>98.98%</td>
<td>inf</td>
</tr>
<tr>
<td>QDA</td>
<td>99.68%</td>
<td>99.40%</td>
<td>98.14%</td>
<td>99.07%</td>
<td>10.7</td>
</tr>
<tr>
<td>Decision Forest</td>
<td>99.45%</td>
<td>98.53%</td>
<td>96.93%</td>
<td>98.30%</td>
<td>inf</td>
</tr>
<tr>
<td>Layered SVM (Polynomial kernel)</td>
<td>99.72%</td>
<td>99.43%</td>
<td>98.00%</td>
<td>99.05%</td>
<td>6.7</td>
</tr>
<tr>
<td>Layered SVM (Quadratic kernel)</td>
<td>99.68%</td>
<td>99.41%</td>
<td>97.40%</td>
<td>98.83%</td>
<td>3.3</td>
</tr>
</tbody>
</table>

“inf” means not applicable in mobile applications.

quadratic discriminant analysis (QDA) and decision forest, and the preliminary experimental results are listed in Table 4.1. Linear, polynomial (with degree 2 and 3) and RBF kernels are tried in SVM implementation. For random forests, we use depth-9 trees and train 100 of them by using 5 random splits when training each weak learner. The results show that our layered strategy outperforms the one-vs-all SVM in both accuracy and latency, and layer 3 yields the worst performance across all methods, while layer 1 and 2 are better. Taken both speed and accuracy into account, QDA and layered SVM with polynomial kernel are better methods for color recovery, where QDA is of higher accuracy while layered SVM has lower processing latency. In the later section, we compare the performance of QDA and layered SVM in real mobile applications.

Layered SVM with RBF kernel seems to be the best choice for our application considering
accuracy, but using kernel techniques in SVM on a mobile application is too time-consuming. To solve this problem, we can either approximate the RBF kernel via explicit feature mapping [24] [29], or map the original feature into a slightly higher dimensional space using an as-good kernel\(^1\) such as a low-degree polynomial kernel [4]. Then, we can use the augmented feature after mapping to train a linear SVM which is fast for prediction. For simplicity, we use a low degree polynomial kernel, e.g., 2 (quadratic) or 3 (polynomial), to explicitly map the feature \(x = \{x_1, x_2, x_3\}\) from 3-dim to 10-dim \(\phi_2(x)\) or 20-dim \(\phi_3(x)\) by using:

\[
\phi_2(x) = [1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3]^T, \tag{4.6}
\]

and

\[
\phi_3(x) = [1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3, x_1^3, x_2^3, x_3^3, x_1^2x_2, x_1^2x_3, x_2^2x_3, x_1x_2^2, x_1x_3^2, x_2x_3^2, x_1x_2x_3]^T. \tag{4.7}
\]

Moreover, based on our finding that linear kernel has comparable performance with other kernels in the first two layers while only suffers in the third layer, we train a linear SVM using the mapped feature only for the weak layer(s) (e.g., the third layer). In this way, we further reduce the computational cost during color recovery.
### Table 4.2: Scanning Performance of QDA and Layered SVM on Mobile Applications

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Method</th>
<th>QDA</th>
<th>Layered SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of frames</td>
<td>1.44</td>
<td>1.86</td>
</tr>
<tr>
<td>117</td>
<td>Overall latency (ms)</td>
<td>375.06</td>
<td>372.83</td>
</tr>
<tr>
<td></td>
<td>Time per frame (ms)</td>
<td>260.91</td>
<td>200.22</td>
</tr>
<tr>
<td>137</td>
<td># of frames</td>
<td>1.40</td>
<td>1.44</td>
</tr>
<tr>
<td></td>
<td>Overall latency (ms)</td>
<td>435.32</td>
<td>308.36</td>
</tr>
<tr>
<td></td>
<td>Time per frame (ms)</td>
<td>310.94</td>
<td>214.14</td>
</tr>
<tr>
<td>157</td>
<td># of frames</td>
<td>3.65</td>
<td>5.51</td>
</tr>
<tr>
<td></td>
<td>Overall latency (ms)</td>
<td>1323.77</td>
<td>1295.17</td>
</tr>
<tr>
<td></td>
<td>Time per frame (ms)</td>
<td>362.30</td>
<td>234.75</td>
</tr>
<tr>
<td>177</td>
<td># of frames</td>
<td>5.30</td>
<td>6.67</td>
</tr>
<tr>
<td></td>
<td>Overall latency (ms)</td>
<td>2150.59</td>
<td>1704.21</td>
</tr>
<tr>
<td></td>
<td>Time per frame (ms)</td>
<td>406.06</td>
<td>255.63</td>
</tr>
</tbody>
</table>

### 4.2.3 Comparison between QDA and layered SVM

In this section, we evaluate the performance of QDA and layered SVM in real mobile application to study this speed-accuracy tradeoff by using iPhone 6 Plus as the representative mobile device. Note that in our implementation of layered SVM we use linear kernel in the first two layers and polynomial kernel of degree 3 in the third layer. In this experiment, we choose 4 challenging samples of color QR codes in different versions and printout sizes, namely, version 25, 30, 35 and 40 in size 3.5\,mm, 4\,mm, 5\,mm and 5\,mm, respectively. We train the QDA and layered SVM using \footnote{RBF kernel is not feasible in this case because RBF kernel explicitly maps to an infinite feature space.}
the same training data set and using the same device, iPhone 6 Plus, to scan each sample for 30 times. Table 4.2 lists the performance of the two methods. It is shown that compared with QDA layered SVM takes more frames to complete decoding a color QR code but consumes less time in processing each frame. In general, layered SVM outperforms QDA because layered SVM takes less time (on average) to successfully decode a color QR code, and we expect there being further improvements as the number of layers increases.

4.3 Classifier training

Other frameworks such as [3] and [9] train the classifier on-line for each captured image by extracting labeled data from reference colors. Different from these on-line methods, HiQ learns the parameters of the classifier off-line using color data collected exhaustively from real settings of QR codes scanning. This avoids training bias and unnecessary computations on mobile devices.

To collect the color data from the color QR code images, we use a human-assisted labeling approach. To be more specific, given a captured image of a color QR code, instead of manually labeling the color pixel by pixel, we only manually input the positions of the markers of the color QR code and apply the existing geometric transformation algorithm to sample the color data from each square module of which we actually know the ground-truth color. In this way, we can substantially cut down the manual labeling effort while managing to collect millions of labeled color-data modules under a wide range of real-world operating/ lighting conditions. Such a rich set of labeled data plays an important role in boosting color prediction/ classification performance for our
learning-based decoding algorithm.

Moreover, due to the fact that the estimation of white color (i.e., \( W \)) is often noisy, we augment the training data by deliberately injecting noise to \( W \) to enhance the robustness of our color recovery algorithm to the noisy estimation of white color. For a \( n \)-layer color QR code, the trained classifier takes the normalized color as the input and outputs the color label which is then mapped to \( n \) binary bits for decoding using the same color codebook in encoding process (see Sec. 3.2).

End of chapter.
In this chapter, we discuss some practical issues we encounter in the implementation of our color QR codes scanner for mobile applications. In particular, we develop some heuristics and approaches to reduce the scanning latency and improve performance of our mobile application, which include monochrome and color QR codes identification (see Sec. 5.1), model selection (see Sec. 5.3), spatial data bits randomization and block accumulation (see Sec. 5.2).
5.1 Identification of monochrome and color QR codes

In order to make our framework compatible to decode traditional monochrome QR codes, we propose a fast and effective method to differentiate monochrome QR codes and color QR codes. Note that monochrome QR codes can be any combination of two differentiable colors, not just black and white, see Fig. 1.1(b) and Fig. 1.1(c) for illustration. Therefore, instead of making assumption of the color of the monochrome QR code which facilitates the identification of the type of the QR code but harms the generality of the method, we solve this problem by analyzing the difference between the color distributions of monochrome and color QR codes.

Fig 5.1 gives the color distributions of the two monochrome QR codes (black/white and red/white) and a 3-layer color QR code, from which we can observe that the color distributions of two monochrome QR codes are “thinner” than that of the color QR code. Based on this finding, we propose to use PCA to model the “thickness” of a color distribution. In particular, we define a
CHAPTER 5. IMPLEMENTATION DETAILS

*eigenratio* $\gamma$, where

$$
\gamma = \frac{\text{largest eigenvalue}}{\text{smallest eigenvalue}}.
$$

One can see the eigen-ratio of a monochrome QR code is much larger than that of a color QR code. Therefore, for each captured frame, we first sample a small patch of the QR code and calculate the eigenvalues using SVD which is fast (only cost about 15 milliseconds using Nexus 5), then we define a threshold $t$ (which is set to 25 in our implementation), if $\gamma > t$, then the QR code is monochrome, otherwise color. Lastly, we can use corresponding algorithms to decode the captured QR code image.

### 5.2 Spatial randomization and block accumulation

Through our experiences of scanning QR codes using mobile devices, we noticed one strange fact that a small amount of errors in the local area, which is definitely within the error correction capability, somehow fails the whole decoding. After examining the error correction mechanism, we surprisingly found that QR code decoder performs the error correction block by block, and in each data block the data is shuffled byte by byte in the QR code. However, for high-dimensional QR code, data bits from the same block do not spread out uniformly, instead they tend to assemble in the local areas (see Fig. 5.2 for illustration). Consequently, the concentration of the data bits in each block will easily lead to error-correction failure because external causes like local overexposure and QR code damage often lead to a large error percentage in one block. What’s worse, if there exists block decoding failure—even one single block failure—then decoder will initiate a new round of
scanning while discarding all information obtained so far. In addition, we find that in most failure decoding cases errors in each captured image always assemble in several blocks instead of affecting all the blocks.

To tackle these problems, to improve the performance of HiQ framework, and to reduce the latency of real-time scanning, we make the following adaptations:

- **Spatial randomization of data bits in each block**: To avoid data block decoding failure caused by local error, we propose to shuffle the data of each block bit-by-bit into the whole matrix uniformly, which we named as *spatial bits randomization*, to improve the probability of successful correction of each block as shown in Fig. 5.2.

![Figure 5.2: Spatial Randomization of Data Bits in Each Block](image)

Figure 5.2: Spatial *Randomization* of Data Bits in Each Block. The same color bits belongs to the same single error-correction block.

- **Data block accumulation**: Moreover, in order to prevent the failure in one single block making the efforts in other blocks in vain and reduce the scanning latency, we propose *Data Block Accumulation*, which accumulates the successfully decoded data blocks in previous scans until all the data blocks are successfully decoded.
By using these heuristics, we manage to cut down the scanning latency by a large margin, see Sec. 6.4.4 for experimental results.

5.3 Model selection for color QR codes

Furthermore, we propose to use real-time *model selection* to deal with wide illumination variation which may be caused by different lighting conditions and different imaging devices. We separately train different classifiers using data collected under several major different lighting conditions, e.g., fluorescent and incandescent. During decoding, we evaluate each classifier using data sampled from the function patterns (finder patterns and alignment patterns) whose colors we already known, and choose the one of the highest accuracy or score to be the classifier in the color recovery for the current frame. If there are multiple classifiers with the highest accuracy, we then compare their accumulative utilities/scores of all data and choose the one with the highest score.

☐ End of chapter.
Chapter 6

Experimental Evaluation

Summary

This chapter presents experimental evaluation and analysis of the proposed framework, HiQ. The evaluation is divided into two major parts: evaluation using our collected dataset, CUHK-CQRC, which is collected under various lighting conditions using different mobile devices (see Sec. 6.3); and the evaluation on real mobile applications using smartphones including Nexus 5 and iPhone 6 plus (see Sec. 6.4). We start this chapter with the definitions of the performance metrics in Sec. 6.1, which is followed by a detailed introduction about the evaluation dataset CUHK-CQRC in Sec. 6.2.
In this section, we provide extensive analysis of the experimental results. To evaluate the performance of HiQ, we conduct two different sets of experiments, one on a challenging color QR code dataset, CUHK-CQRC and one on real-world mobile devices. To be unambiguous, we use HiQ-C to denote color QR codes under HiQ framework. We collect a large-scale color QR code dataset, CUHK-CQRC, to evaluate the performance of HiQ-C compared with PCCC [3], which proposes two different approaches: PB for the color QR code with embedded reference color and EM for those without the reference color. We use PB as the baseline approach for color QR codes in this work because PB performs better than EM [3]. We also show the effectiveness of HiQ-C by evaluating its performance on real mobile application.

6.1 Performance metrics

We use the following four metrics to quantify the performance of each approach: 1) decoding success rate (DSR), 2) bit error rate (BER), 3) localization success rate (LSR), 4) scanning time. DSR, BER and LSR are used for the evaluation on CUHK-CQRC which is conducted on desktop, while scanning time is for testing on mobile devices.

DSR is the percentage of QR codes which can be successfully decoded with the error correction mechanism applied over those that can be successfully localized. It is used to measure the overall performance of the decoding method, including geometric transformation and color recovery; BER denotes the percentage of wrongly recovered bits before applying error correction compared with the embedded bit stream of the originally generated QR code. Compared with DSR, BER is a rela-
Figure 6.1: The phases of decoding process the each performance metric measures.

*only for color QR code

...metric which directly measures the error of the color recovery and the geometric transformation. LSR is the percentage of QR code that can be successfully localized by the QR scanner, and it measures the effectiveness of the binarization and localization. Scanning time is the interval between the time when the camera takes the first frame and the time when the decoding is finished. It measures the overall performance of the decoding approaches on mobile devices.

6.2 CUHK-CQRC: a large-scale color QR code dataset

In this work, we create a challenging color QR code dataset, CUHK-CQRC, which consists of 1,506 photos and 3,884 camera previews (video frames) of high-density 3-layer color QR codes captured by different phone models under different lighting conditions. Fig. 6.2 presents some samples of CUHK-CQRC. Different from [3], we also include previews in our dataset because of the two reasons. Firstly, photos are different from previews. When users take a photo using the on-board camera of a mobile phone, many embedded systems implicitly process (e.g., lightweight deblurring and color correction) the output image in order to make it more attractive in appearance, while preview will not go through this process. Secondly, compared with capturing photos, it is
Table 6.1: Types of Smartphones Used in Collecting Database

<table>
<thead>
<tr>
<th>ID</th>
<th>Model Name</th>
<th>Megapixels (MP)</th>
<th>Image Stabilization</th>
<th>AutoFocus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iphone 6 plus</td>
<td>8.0</td>
<td>✓ (optical)</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>iphone 6</td>
<td>8.0</td>
<td>✓ (digital)</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Nexus 4</td>
<td>8.0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Meizu MX2</td>
<td>8.0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Oneplus 1</td>
<td>13.0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Galaxy Nexus 3</td>
<td>5.0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>Sony Xperia M2</td>
<td>8.0</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>Nexus 5</td>
<td>8.0</td>
<td>✓ (optical)</td>
<td>✓</td>
</tr>
</tbody>
</table>

much faster and more cost-effective for a cellphone camera to generate previews, and hence most mobile applications use camera previews as the input of the decoder.

We implement the color QR code generator based on an open-source barcode processing library, ZXing, and we also deploy the HiQ generator on-line at https://authpaper.net/.

For fair comparison between HiQ-C and PCCC in which the color QR code is inherently 3-layer, so we generate 5 high-capacity 3-layer color QR codes with different data capacities (excluding redundancies from error correction mechanism) which are 2787 bytes, 3819 bytes, 5196 bytes, 6909 bytes, 9120 bytes.
bytes and 8859 bytes (maximum for a 3-layer color QR code). In order to test the limit of each approach, all color QR codes are embedded with low level of error correction in each layer. By using a common color printer (Ricoh Aficio MP C5501A), we print each generated color QR code on ordinary white paper substrate in different printout sizes, 30 mm, 40 mm, 50 mm and 60 mm (for simplicity, we use the length of one side of the square to represent the printout size), and two different printout resolutions, 600 dpi and 1200 dpi. To simulate the normal scanning scenario, the samples are captured by different users under several typical lighting conditions: indoor, outdoor (under different types of weather and time duration of a day), fluorescent, incandescent, and shadowed (both uniform and nonuniform cases are considered). Moreover, we capture the images using eight types of popular smartphones (see Table 6.1 for details).

6.3 Evaluations on CUHK-CQRC dataset

6.3.1 Evaluation of HiQ on CUHK-CQRC

In this set of experiments, we evaluate the performance of HiQ-C by comparing it with the baseline method, PCCC from [3] using CUHK-CQRC. Note that although the QR codes in CUHK-CQRC are generated using the proposed encoder (Section 3.2), the color QR codes are also compatible with the PCCC decoder. Both HiQ-C and PCCC are implemented using the decoder part of a popular traditional QR code implementation, ZXing [19], as the code base.

Since the parameters of HiQ-C are learned off-line, we train the color classifier of HiQ-C using data sampled from CUHK-CQRC prior to conducting experiments. We randomly sample 65 images
of color QR codes which cover different lighting conditions, phone models, print resolutions and formats (i.e., photo and preview) for training and use the rest for testing.

We use LSR, BER and DSR (defined in Section 6.1) to measure the performance of each approach.

As we apply the same localization algorithms for both HiQ and PCCC, they achieve the same LSR of 68.87%. Fig. 6.3 shows that HiQ-C outperforms PCCC in both DSR and BER. When comparing with PCCC, HiQ increases DSR (the higher the better) by 286%, while reducing BER (the lower the better) by 60%. We also apply RGT on PCCC (labeled as PCCC with RGT in Fig. 6.3). RGT improves the DSR and BER of PCCC by 159% and 17.5% respectively. Across all of the 3 schemes under test, the third layer yields the worst performance, which is also consistent with the finding in [3]. This is because the color classifier cannot well distinguish between yellow and white which are encoded as \(\{0, 0, 1\}\) and \(\{0, 0, 0\}\) respectively in the codebook (see Fig. 3.1), especially under strong light. Likewise, the classifier performs poorly in distinguishing blue (\(\{1, 1, 0\}\) and
Figure 6.4: Decoding results of HiQ on CUHK-CQRC with different estimated module sizes.

black (\{1, 1, 1\}) under dim light. The combined effect is that the third layer often cannot be decoded reliably during more extreme lighting conditions. Fortunately, it is possible to apply a higher level of error correction on the third layer to compensate the higher error rate in color classification.

Notice that as the distance between the camera and the QR code under test varies, the estimated module size for the QR code also changes. By examining the estimated module size during the scanning process, we find that the decoding performance (in both DSR and BER) for all three layers consistently improves as the estimated module size of the QR code increases (see Fig. 6.4). This observation suggests that:

1. when scanning a color QR code, it is helpful to put the camera as close to the QR code as possible as long as it is able to capture a well-focused image.

2. a higher-resolution image can enhance decoding performance before the onset of color-smearing effects due to the limited output resolution of color printers.
6.3.2 Evaluation of model selection

In this section, we evaluate the effectiveness of model selection described in Sec. 3.4 by comparing classifiers trained with and without model selection scheme. In particular, we train layered SVMs (see 4.2) using data collected under three typical lighting conditions, fluorescent, incandescent and outdoor, and evaluate the classifiers on CUHK-CQRC. With model selection, we train three different layered SVM using data from different lighting conditions separately, while without model selection, only one layered SVM will be trained using data from all three lighting conditions. The experimental results are listed in Fig. 6.5. It is shown that in general using model selection can improve the scanning performance in both DSR and BER compared with using no model selection.

Figure 6.5: Decoding results of HiQ-C, PCCC and PCCC with RGT on CUHK-CQRC.
(a) LLL (≤ 8900 bytes). (b) LLM (≤ 8200 bytes). (c) LLQ (≤ 7600 bytes). (d) MMM (≤ 7000 bytes).

(e) LLL (≤ 8900 bytes). (f) LLM (≤ 8200 bytes). (g) LLQ (≤ 7600 bytes). (h) MMM (≤ 7000 bytes).

Figure 6.6: Performance of HiQ on iPhone 6 Plus and Nexus 5 across various content sizes, printout sizes and error correction levels, measured by the 90th percentile of the scanning time of 30 successful scans. Since we cannot generate a QR code beyond its maximal capacity, we set its scanning time to infinity. The first row and the second row are the results of iPhone 6 Plus and Nexus 5 respectively.

### 6.4 Evaluations on mobile devices

#### 6.4.1 Evaluation of HiQ on mobile application

We demonstrate the effectiveness of HiQ-C under two popular mobile platforms, Android and iOS, using two representative mobile phones, Nexus 5 and iPhone 6 Plus. In this experiment, by using 3-layer color QR codes, we study the performance of the decoding algorithm given different content sizes (excluding error correction redundancies) and printout sizes, and we also evaluate the impact of four possible error correction levels. In particular, we choose 6 different content sizes, 2000 bytes, 2900 bytes, 4500 bytes, 6100 bytes, 7700 bytes and 8900 bytes (largest capacity for a 3-layer color QR code). For each content size, we generate color QR codes using 4 different levels of
error correction which are denoted by 4 triplets, $LLL$, $LLM$, $LLQ$ and $MMM$. Each symbol of the triplet represents the level of error correction applied for the corresponding layer (see Section 2.1.1). For example, the $M$ in $LLM$ represents the Medium (M) level of error correction on the third layer of the color QR code. We try different error correction levels in the third layer because it has shown to be the most error-prone layer during decoding in our previous experiments (see Sec. 6.3.1 for details). As data capacity of a QR code will be reduced if higher error correction level is used, so we cannot apply 4 different error corrections for all content sizes. For instance, with a content size of 8900 bytes, we can only use $LLL$, and for content size of 7700 bytes, only $LLL$ and $LLM$ are feasible. All color QR codes are printed in different printout sizes ranging from 22 mm to 70 mm using a Ricoh Aficio MP C5501A office color printer.

We collect the scanning time of 30 successful scans (i.e., trials where the HiQ-C code is successfully decoded) for each printout HiQ-C code and the experiments are conducted under fluorescent lighting. A trial is unsuccessful if the code cannot be decoded within 60 seconds, and a HiQ-C code is regarded as undecodable if three consecutive unsuccessful trials occur.

Fig. 6.7 gives the performance of HiQ-C on color QR codes with different error correction level and different content sizes. The results demonstrate the effectiveness and robustness of HiQ-C when implemented in off-the-shelf smartphones. Observe from the figure that, the HiQ-C decoder can, in most of the cases, successfully decode the color QR code within 5 seconds with small variance. Fig. 6.6 conveniently shows the smallest printout sizes of the color QR codes with different content sizes that can be decoded in a reliable and rapid manner. And we observe that the performance differs between the two mobile devices: iPhone 6 Plus is better and faster regarding
Figure 6.7: Scanning performance of HiQ-C across various content sizes, printout sizes and error correction levels. We collect the scanning time (in seconds) of 30 successful trials using iPhone 6 Plus and use the 90th percentile, 10th percentile and median to represent the upper bound, lower bound of the scanning time and the overall performance.

It is clear that for larger content size, the smallest decodable printout size will be larger. For example, given a content size of 4500 bytes and 7700 bytes using LLL error correction level, the overall performance, but Nexus 5 can scan the QR codes with smaller printout size (comparing Fig. 6.6(c) and Fig. 6.6(g)). This is because iPhone 6 Plus has a more powerful CPU, while Nexus 5 has smaller minimal in-focus distance, which means when the QR code and the camera are too close iPhone 6 Plus cannot focus while Nexus 5 can. Consequently, the QR code captured by Nexus 5 has larger module size, and thus better performance as we discussed in Sec. 6.3.1.

It is clear that for larger content size, the smallest decodable printout size will be larger. For example, given a content size of 4500 bytes and 7700 bytes using LLL error correction level, the
smallest decodable QR code sizes are $34 \text{ mm}$ and $50 \text{ mm}$, respectively. Comparing the performance of different error correction levels, we can see that, in most cases, $LLL$ and $LLM$ outperform $LLQ$ and $MMM$ in terms of the smallest decodable printout size given the same content size. This suggests that it is not helpful to apply error correction level higher than $M$ on even the weak layer. This is due to the fact that, while applying higher level of error correction can increase the error-tolerant ability, it also increases the data density by add more redundancies. Fig. 6.8(b)-6.7(f) show that, given the same content size, QR codes with larger printout size can sometime be worse (in terms of scanning performance) than QR codes with smaller printout size. This is due to the fact that, when the size of the printout gets too large (e.g., $\geq 58, \text{ mm}$), the printout becomes more susceptible to overexposure and non-uniform lighting which makes decoding more challenging.

6.4.2 Pipeline analysis

In this section, we examine the execution time of sequential flows (basic components) of instructions in the pipeline of our proposed HiQ method on mobile device so as to find possible improvement. The experiments are conducted on several QR codes of different dimensions by using Nexus 5 as the representative phone model. Specifically, we choose 3 different dimensions, 137, 157 and 177 (largest dimension of QR code standard). For each dimension, we evaluate the execution time of each block by averaging over ten scans.

Note that we implement our framework in real-world applications on both Android and iOS. However, there is a difference between our Android implementation and iOS implementation, which is that under Android platform (before version 6.0) we cannot directly obtain RGB inten-
<table>
<thead>
<tr>
<th>Dim</th>
<th>Type</th>
<th>YUV-2- RGB</th>
<th>Binari- zation</th>
<th>Patterns</th>
<th>Transformation</th>
<th>QDA Sample</th>
<th>Randomization</th>
<th>Correc- tion</th>
<th>Error</th>
<th>Time per Frame</th>
<th>Number of Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>137</td>
<td>B/W</td>
<td>NA</td>
<td>112ms (34%)</td>
<td>23ms (7%)</td>
<td>16ms (5%)</td>
<td>20ms (6%)</td>
<td>41ms (13%)</td>
<td>0ms</td>
<td></td>
<td>100ms</td>
<td>3</td>
</tr>
<tr>
<td>157</td>
<td>Color</td>
<td>400ms (27%)</td>
<td>204ms (14%)</td>
<td>153ms (10%)</td>
<td>14ms (1%)</td>
<td>500ms (34%)</td>
<td>18ms (5%)</td>
<td>45ms (3%)</td>
<td>37ms</td>
<td>150ms</td>
<td>4.5</td>
</tr>
<tr>
<td>177</td>
<td>B/W</td>
<td>NA</td>
<td>104ms (32%)</td>
<td>123ms (27%)</td>
<td>37ms (10%)</td>
<td>38ms (10%)</td>
<td>60ms (16%)</td>
<td>60ms (10%)</td>
<td>380ms</td>
<td>600ms</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>386ms (24%)</td>
<td>200ms (12%)</td>
<td>150ms (9%)</td>
<td>20ms (1%)</td>
<td>650ms (40%)</td>
<td>38ms (10%)</td>
<td>69ms (4%)</td>
<td>60ms</td>
<td>200ms</td>
<td>5.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Color</td>
<td>400ms (20%)</td>
<td>193ms (10%)</td>
<td>213ms (11%)</td>
<td>25ms (2%)</td>
<td>881ms (44%)</td>
<td>21ms (5%)</td>
<td>50ms (11%)</td>
<td>97ms</td>
<td>200ms</td>
<td>9</td>
</tr>
</tbody>
</table>
cies from the sensor, while iOS can. To solve this problem, we use YUV color format which is supported in Android and convert YUV data to RGB intensities\(^1\) which are then processed the same as in iOS.

Table 6.2 gives the pipeline analysis of HiQ-C on Nexus 5 with different dimensions. Observe from the table that the most time-consuming parts for HiQ-C is color recovery using QDA and the YUV-to-RGB conversion, taking up around 40% and 20% respectively. YUV-to-RGB conversion is necessary for Android mobile system to convert the captured images format to RGB intensities, which can be processed as we mentioned in Sec. 4. Fortunately, the YUV-to-RGB conversion is not necessary after Android 6.0, which saves much computational power. This analysis also motivates us to develop the layered SVM (see Sec. 4.2) to accelerate the color recovery process. Furthermore, the randomization part for both monochrome and color QR code takes up no more than 12% of total time, which is acceptable for practical usage.

### 6.4.3 Analysis of the layer-density trade-off

As we discussed in section 3.2, there exists a trade-off between layer and density, namely, given the same amount of user data, a color QR code with more layers has less module density but it also has more difficulties in color recovery. In this section, we study this layer-density trade-off by comparing the performance of 2-layer and 3-layer color QR codes.

We conduct this experiment under fluorescent lighting conditions and using Nexus 5 as the representative mobile device. We print 2-layer and 3-layer color QR codes of 2900, 4500, 5800

---

\(^1\)This color space conversion from YUV to RGB can lead to some information loss and latency increase.
Table 6.3: Comparison between 2-layer and 3-layer color QR codes

<table>
<thead>
<tr>
<th></th>
<th>2900 bytes</th>
<th>4500 bytes</th>
<th>5800 bytes</th>
<th>8900 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2-layer</td>
<td>3-layer</td>
<td>2-layer</td>
<td>3-layer</td>
</tr>
<tr>
<td>dimension</td>
<td>125</td>
<td>105</td>
<td>157</td>
<td>125</td>
</tr>
<tr>
<td>limit size</td>
<td>2.5cm</td>
<td>2.6cm</td>
<td>3.5cm</td>
<td>3.4cm</td>
</tr>
<tr>
<td>number of module</td>
<td>15625</td>
<td>11025</td>
<td>24649</td>
<td>15625</td>
</tr>
<tr>
<td>predictions per frame</td>
<td>62500</td>
<td>88200</td>
<td>98596</td>
<td>125000</td>
</tr>
</tbody>
</table>

and 8900 bytes amount of user data using a common office color printer, Ricoh Aficio MP C5501A. In this experiment, we use QDA as the color classifier both for 2-layer (4 classes) and 3-layer (8 classes) color QR codes. Table 6.3 lists the experimental results. It is shown that given the same content sizes, 2-layer and 3-layer color QR codes have similar limit printout sizes\(^2\), and decoding a 2-layer color QR code consumes less time in each frame. Besides, it is also shown that given the same versions 2-layer color QR codes can decode higher-density QR codes (smaller limit size), which indicates the difficulties brought by adding one layer (4 classes).

### 6.4.4 Evaluation of randomization

We evaluate the performance of randomization, which is discussed in Sec. 5, for HiQ-C on mobile device. We use Nexus 5 as the representative phone model, which is more challenging model for high-density color QR codes compared with the iPhone 6 Plus. We compare the scanning

\(^2\)The smallest printout size of the QR codes that can be decoded.
Figure 6.8: Comparison of scanning performance of randomized and original (non-randomized) color QR codes on mobile device. We collect the scanning time (in seconds) of 10 successful trails using Nexus 5, and use the average running time to represent the performance. When the print-size of 6100-bytes samples become bigger, it is difficult to capture the images under uniform illumination, which is the reason that 54mm and 56mm-size samples of 6100-bytes QR code have longer running time than smaller samples.

Performance on randomized and non-randomized QR code samples, given different content sizes (4500-bytes, 6100-bytes and 7700-bytes) and printout sizes.

Fig. 6.8 gives the performance comparison with respect of scanning latency. Observed from the figure that, the randomization has no significant improvement for low-density (4500-bytes color QR code) compared with the original samples. For 6600-bytes color QR code, randomization can push the limit of printout size from 46mm to 42mm, and randomized samples has shorter scanning time before they reach the limit size. When the content size of the color QR code reach 7700 bytes, the original samples can hardly be decoded by Nexus 5, while the randomized samples can still be decoded and the decoder reach the limit printout size until 50 mm. Therefore, we conclude that Randomization can help to improve the scanning performance of high-density color QR codes.

Moreover, Fig. 6.9 gives the performance with respect of block-accumulation behavior. It indicates that the randomized samples have higher starting point of successful-blocks percentage, and have
Figure 6.9: Comparison of scanning performance with respect of the block-accumulation behavior. The selected samples are those cannot be decoded without randomization. The accumulated blocks are the percentage of successfully decoded blocks accumulated by each frame.

higher rising speed, while the original samples got stuck and cannot decoded.

End of chapter.
Chapter 7

Related Works

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>In this chapter, we review the related works regarding monochrome QR codes in Sec 7.1 and color 2D barcodes in Sec. 7.2.</td>
</tr>
</tbody>
</table>

7.1 Monochrome QR code

Binarization, barcode localization and geometric distortion correction are three key issues for robustly decoding the monochrome QR code. [16] provides a full analysis of the quick response of the QR codes in the wild. G. Soros et. al. [26] propose to localize the QR code by detecting the area with plenty of lines and corners, relying on the structure feature of QR code. However, compared with localization using finder patterns, this method is not robust to the QR code with complex back-
CHAPTER 7. RELATED WORKS

ground and not sensitive to the blurriness, which can help us to discard the low-quality frames and retake a new image for decoding. H. Yang et al [32] propose to binarize each pixel in the captured image by construct the dynamic window with adaptive size and calculate the threshold according to the statistics of its local window. They are devoted to build the local window where the numbers of black and white pixels are in balance. However, this method is time consuming, which takes 4.9 s for a 320-by-240 image on desktop processing.

For the geometric distortion correction, most of the QR decoders use only the location of four patterns in the corners to do transformation [18] [7], which are not robust and accurate enough to perspective distortion. Besides, W. Xu et al [31], G. Soros et al [27] and C. Chu et al [7] propose to remove the motion blur in captured images by assuming the motion blur is linear and constant-velocity. However, they take over 10s to process a single frame, which is not applicable in real-time processing. Besides, the assumption of linear motion blur is limited, since the blur due to handshake is always nonuniform and much more complicated. Besides, C. Chen [5] propose to measure the quality of the captured barcode image before feeding it to the decoder, which can reject most of the blurred and underexposed images that cannot be decoded in advance and hence reduce the process latency.

7.2 Color 2D barcodes

In recent years, there have been numerous attempts in the literature using color to increase the capacity of traditional 2D barcodes [3] [20] [14] [9] [23] [21] [1]. Microsoft High Capacity Color
Barcode (HCCB) [21], as illustrated in Fig. 2.3(a), encodes data using color triangles with a predefined color palette. The number of colors in the palette can be 2, 4, or 8, depending upon the amount of user data. However, [9] reports the fragility in the localization and alignment of HCCB, and by far the most common (probably the only) mobile application, Microsoft Tag, requires the Internet accessibility during decoding. Moreover, the on-line creator of HCCB only allows users to encode no more than 1000 bytes of data in HCCB.

The color QR code framework, HCC2D, proposed in [9] also encodes multiple data bits in each color symbol and both 4- and 8-color palette are used to provide different data capacities. Moreover, HCC2D adds extra color symbols which serve as the reference color in decoding process. However, our preliminary testing shows that using reference color can easily introduce bias in the color recovery process. It is because dirt and damage on the reference symbols, or even nonuniform lighting can easily make color QR codes impossible to decode. Lastly, no usability study on mobile devices was provided for HCC2D.

H. Blasinski et al. [3] propose the per-colorant-channel color (PCCC) framework for 2D barcodes where data are encoded in three independent monochrome QR codes which represent the three channels in the CMY color space during printing. The authors then propose a color interference cancellation algorithm to perform color recovery. However, in PCCC, for each captured image, parameters for color interference cancellation should be learned before decoding, which brings unnecessary computational burden to mobile devices. Moreover, the color interference cancellation algorithm implicitly assumes every color QR code scanning involves printing process, and thus PCCC is inapplicable to the scenario where we want to decode a color QR code on the screen.
of a computer or a mobile device. Lastly, the experimental evaluations in [3] do not study the effectiveness of their proposed scheme on high-density color QR codes\(^1\). Neither do they investigate the limitations regarding smartphone-based implementations.

H. Bagherinia and R. Manduchi propose to model the color variation under various illuminations using a low-dimension subspace, e.g., principal component analysis, without requiring reference color patches [1]. Shimizu et. al. propose a 64-color 2D barcode and augment the RGB color space using seed colors which functions as references to facilitate color classification [25]. Their method uses 15-dimension or 27-dimension feature both in training and testing which may not be too computationally expensive for mobile devices in real application. In [2], to decode color barcodes from blurry images, H. Bagherinia and R. Manduchi propose a iterative method where they assume each measured patch color is a linear combination of the colors of the patch itself and its neighboring patches. However, their proposed method takes more than 7 seconds on a desktop with Intel i5-2520M CPU @ 2.50GHz to process one image, thus can hardly be used for mobile devices.

\(^{\text{1}}\)The color QR samples used by [1] only hold no more than 150 bytes within an area whose size ranges from 15 mm to 30 mm.
Chapter 8

Conclusion and Future Work

Summary

This chapter concludes this thesis, and summarizes the future works which potentially can further improve the proposed system.

This work presented HiQ, a layered framework for high-capacity color QR codes, which supports robust and rapid decoding using off-the-shelf smartphones. HiQ enables users and developers to create generalized QR codes with flexible and broader range of choices of data capacity, error correction and color, etc. Moreover, we have also collected a large-scale color QR code dataset, CUHK-CQRC, which will be made available to the community. Experimental results show substantial advantages of the HiQ over other baseline approaches. Our implementation of HiQ using off-the-shelf smartphones has demonstrated its usability and effectiveness in mobile applications.
As future work, we plan to generalize the framework to support different number of color layers and study the trade-offs between the number of layers, the choice of encoding colors and the corresponding decoding performance. Another possible direction is to leverage the emerging deep-learning techniques to further enhance the color learning/recovery process.

In addition, under current design the error codes are added in each layer independently, and thus the error correction is performed layer by layer. However, based on our experiences, there exists weak layer in which the errors occur more often than in the other layers. In weak layer the error often go beyond the amount that the error codes can repair, while in other layers there are redundant error-correction capacities. Therefore, it will be helpful if we can share the correction capacity across all layers by constructing error codes and performing correction for contents in all layers as a whole rather than layer by layer.

We also developed a prototype for 4-layer color QR codes, but we found that the bottleneck is to select a color set of 16 colors of which every two colors can be well differentiated. However, it is hard to find such large number of colors under a color space which is distorted during the printing and capturing process (see Fig. 3.6 for illustration). Although it is difficult, we have a few possible directions can be explored: 1) a greedy-type approach, where we collect color data of many possible candidate colors under various lighting conditions and choose the best group of colors; 2) we can build mathematical model like sphere packing [30] to geometrically analyze the choice of colors.

☐ End of chapter.
Bibliography


