2015 IEEE 35th International Conference on Distributed Computing Systems

# RainBar: Robust Application-driven Visual Communication using Color Barcodes

Qian Wang†, Man Zhou†, Kui Ren‡, Tao Lei†, Jikun Li† and Zhibo Wang†*
†School of Computer Science, Wuhan University, P. R. China Email: qianwang@whu.edu.cn
‡Dept. of CSE, The State University of New York at Buffalo, USA Email: kuiren@buffalo.edu

***Abstract*—Color barcode-based visible light communication (VLC) over screen-camera links has attracted great research interest in recent years due to its many desirable properties, including free of charge, free of interference, free of complex network configuration and well-controlled communication security. To achieve high-throughput barcode streaming, previous systems separately address design challenges such as image blur, imperfect frame synchronization and error correction etc., without being investigated as an interrelated whole. This does not fully exploit the capacity of color barcode streaming, and these solutions all have their own limitations from a practical perspective.**

**This paper proposes RainBar, a new and improved color barcode-based visual communication system, which features a carefully-designed high-capacity barcode layout design to allow flexible frame synchronization and accurate code extraction. A progressive code locator detection and localization scheme and a robust color recognition scheme are proposed to enhance system robustness and hence the decoding rate under various working conditions. An extensive experimental study is presented to demonstrate the effectiveness and flexibility of RainBar. Results on Android smartphones show that our system achieves higher average throughput than previous systems, under various working environments.**

***Index Terms*—Visible light communication, color barcode, screen-camera link, robustness, smartphones**

## I. Introduction

As a promising and attractive short-range communication technology, recently visible light communication (VLC) has received significant attention. Due to its inherent property of association by physical proximity and the high directionality of narrow light beams, VLC has found a wide range of applications in various scenarios. The VLC technology was first explored to combine LEDs with techniques from other research domains such as physical layer radio technologies (*e.g.*, OFDM and MIMO [1]), complex computer vision techniques or even additional DSP hardwares [2], [3]. However, most of solutions along this track are not suitable for off-the-shelf devices due to high computational overhead.

Meantime, with the popularity of smartphones, a special form of VLC, *i.e.*, barcode-based VLC, has been developed by leveraging smartphone screen-camera links for data transmission. In our daily life, black and white barcodes have been widely used to identify items in advertisements, social networking, retail and logistics [4]–[6]. Compared to conventional RF wireless connectivity, such links are free of charge and interference. And most importantly, the directionality and extremely short visible range can guarantee well-controlled communication security without troublesome link setup or authentication. Such advantages enable wireless users to automatically initiate and set up communication channels, enjoying quick and convenient information sharing.

Motivated by the idea of detecting colors for high-rate information transfer, researchers began to investigate specially-designed 2D color barcodes [7]–[9], based on which much more information (other than only web links or contact information) can be transmitted in a video of barcodes over screen-camera links. To achieve high throughput barcode streaming, previous approaches aimed at increasing the image quality to enhance the per-frame capacity, or exploiting error correction codes to improve transmission reliability, or ameliorating frame synchronization to increase the frame rate, making the barcode-based VLC systems more and more practical.

However, previous solutions all have their own and limitations. In [7], COBRA set the screen's display rate to be exact half of the camera's capture rate, which under-utilizes the transmitter's capability and therefore limits the communication throughput. Meanwhile, the design of timing reference blocks cannot accurately localize blocks in code area since the captured images may suffer from severe perspective distortion, which affects the decoding accuracy. In [9], the authors proposed a Tri-level error correction scheme to recover lost blocks and frames to further enhance the transmission reliability, which however at the cost of excessive use of codings in all circumstances and therefore greatly reduces the communication throughput/goodput on average. Orthogonal to per-frame improvement, LightSync [8] achieves high throughput by increasing the display rate of barcodes. It addressed the frame synchronization problem as long as the camera's capture rate is at least half of the screen's display rate. LightSync, however, has only been shown to work efficiently for black and white barcodes. Using color barcodes could obviously increase the encoding capacity of a frame than using black and white barcodes, but it is also more difficult to solve the frame synchronization problem. Another limitation with the existing systems is that they generally focused solely on the barcode transmission itself but ignored the inherent applications behind screen-camera links. In fact, it is practical applications that drive new designs and protocols for barcode-based VLC. Therefore, color barcode-based VLC systems should be carefully designed and tailored to meet the particular

*Corresponding author.

1063-6927/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDCS.2015.61

requirements of various applications.

In this paper, we design and implement a new application-driven VLC system called RainBar: Robust Application-driven vIsual commuNication using color BARcodes. The system is designed from two orthogonal perspectives: optimizing the encoding capacity of each color barcode frame and fully utilizing the transmission capacity of the sender, to achieve high throughput over screen-camera links. To this end, we design a new layout for color barcodes to improve encoding capacity and propose new techniques to solve the frame synchronization problem as well as code extraction from blurred images. The contributions of this work are summarized as follows:

- We design and implement RainBar, a new application-driven VLC system that features high decoding rate and thus throughput in dynamic environments compared to previous solutions.
- We design a new layout for color barcodes which features in-frame code locators to accurately localize blocks in code area even for severely distorted and blurred captured images, and in-frame tracking bars to achieve fast and flexible frame synchronization when the display rate is high.
- We propose a progressive code locator detection and localization scheme together with a HSV-based robust code extraction scheme to realize fast and accurate code extraction.
- We implement RainBar on off-the-shelf devices and conduct extensive experiments to evaluate its performance. Experimental results show that RainBar outperforms existing color barcodes systems. We finally discuss how to adapt RainBar to specific applications to meet particular user requirements.

The rest of this paper is organized as follows. Section II discusses the challenges and problems of color barcode design. We present the detailed design of RainBar in Section III and evaluate its performance in Section IV. A discussion of how to adapt RainBar for text file transfer is given in Section V. Finally, Section VI concludes our work.

## II. Challenges of Color Barcode Streaming

The challenges for realizing color barcode streaming lie in both subjective conditions and objective operations, which are listed as follows.

- *Dynamic environments*: The environments cannot be the same when users capture color barcodes at different places. Even for one stream of color barcodes, the working conditions may change as the color barcodes display. Different degrees of illumination and shade could introduce blur to the captured images, which significantly affects the decoding accuracy.
- *Lens distortion*: Lens distortions of cameras are radial distortions that straight lines in a captured image become distorted, which may make some color barcodes become arc-shaped and then affect the localization accuracy of the center point of each block.

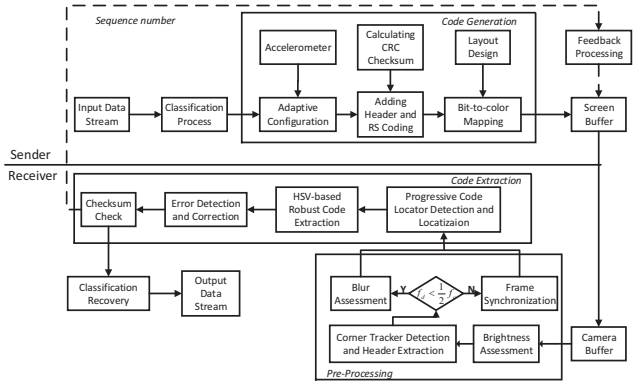



Fig. 1: System architecture of RainBar

- *High display rate*: A higher display rate helps to increase the throughput, but also introduces more blur and noise to the captured images. Even worse, a relatively high display rate makes the frame mixing vary on a per-line basis due to *rolling shutter*.
- *Angle and distance* between the camera and the screen: It is difficult to maintain the distance and angle between two smartphones during the transmission of color barcode frames. The captured images may be distorted or shrinked when the angle is not zero. Also, it is not always possible to maintain the distance so that the whole color barcode falls into the camera's sampling box. All of them increase the difficulty of decoding.

Other challenges include the shake of hands and smartphone differences, etc. Moreover, the requirement of realtime barcode streaming and the limited computational resource of smartphones render the computation-expensive image processing techniques useless. Keeping the above challenges in mind, we are highly motivated to design a robust, flexible and lightweight barcode-based VLC system that fully exploits the capacity and functionalities of off-the-shelf smartphones.

## III. RainBar Design

RainBar aims to achieve robust and high-throughput communication over screen-camera links in dynamic environments. To this end, the system is designed from two orthogonal perspectives: optimizing the encoding capacity of each color frame, and fully utilizing the transmission capacity of the sender smartphone. In this section, we first present the high-level overview of the design of RainBar and then describe how it works in detail.

### A. System Overview

Figure 1 illustrates the system architecture of RainBar that consists of two sides: the sender and the receiver (*e.g.*, two on-the-shelf smartphones). Roughly speaking, the sender encodes data (*e.g.*, a text file) into color barcodes and displays the stream of color barcodes on its screen. The receiver uses its camera to capture images of color barcodes and decodes the captured images to obtain the original data.

**The sender**: Data are encoded into a stream of color barcodes which are displayed on the screen of the sender. A *classification* component pre-processes the data based on its specific application types before data encoding to guarantee the communication efficiency. The *code generation* component is responsible for mapping the pre-processed data into a stream of color barcodes.

In *code generation*, we adopt the *accelerometer* and *adaptive configuration* components from [7] to estimate the level of mobility and adaptively adjust the size of blocks. However, different from [7], we argue that these procedures should be finished before data mapping so that the optimal block size can be determined. Otherwise, we cannot decide how much data should be put in each color barcode frame. Besides, CRC checksum and Reed-Solomon (RS) code [10] are embedded in the data, which can be used for error detection and correction at the receiver. A header contains a lot of important attributes of each color barcode, such as sequence number and display rate etc. The pre-processed data together with the header and RS coding redundancy are mapped into a stream of color barcode frames, where each 2 bits is mapped into a color block. Finally, the stream of color barcodes is displayed on the screen of the sender.

**The receiver**: The receiver captures images of color barcodes and tries to decode the data correctly and quickly. The receiver side mainly has three components: *pre-processing*, *code extraction* and *classification recovery*.

The *pre-processing* component is responsible for degree of blur and brightness estimation, corner tracker detection and header extraction, and frame synchronization. In the *code extraction* component, we propose a *progressive code locator detection and localization* scheme to accurately localize blocks at the code area for blurred and distorted images, and a *HSV-based robust code extraction* scheme to accurately recognize the color of each block. Error correction codes correct errors in the decoded data and CRC checksum checks whether the whole decoded frame has errors or not. If errors cannot be completely corrected, the receiver sends a feedback to the sender to ask for retransmission of this frame. Finally, the *classification recovery* component recovers the data based on the application type.

### *B. RainBar Layout Design*

Figure 2 shows the layout of a color barcode frame in RainBar, which consists of four areas: *corner trackers* (CTs), *code locators*, *tracking bars* and *code area*.

**Corner trackers**: The *corner trackers* are used to quickly and accurately locate the corners of the color barcode. COBRA [7] uses four corner trackers (CTs) to locate four corners of its designed color barcode. Based on extensive experiments, we found that actually two corner trackers and in-frame code locators suffice to quickly and accurately detect four corners. Inspired by our findings, as shown in Figure 2, our color barcode design puts only two CTs at the top-left corner and the top-right corner, respectively. Similar to COBRA, a corner

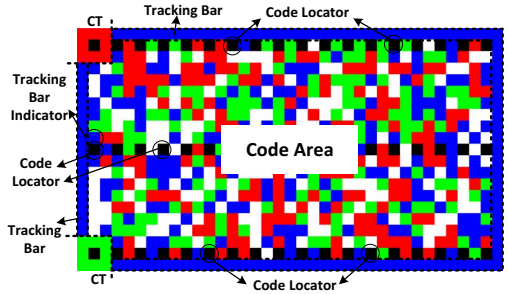



Fig. 2: The layout design of RainBar

tracker is represented by $3\times3$ blocks, with a center black block surrounded by blocks of the same color (*e.g.*, red or green). In our design, the top-left CT is represented by a center black block surrounded by 8 green blocks, and the top-right CT is represented by a center black block surrounded by 8 red blocks. Different from COBRA, our bottom corners can be detected with the help of code locators (We will show how to detect corners in Sections III-C and III-E in detail). Our experiment results show that this design not only reduces the time for corner detection, but also saves more space that can be used for data encoding.

**Code locators**: To successfully decode the encoded information in each block, both the location and the color of each block should be correctly identified. COBRA uses *timing reference blocks* (TRBs) at four sides to localize each block at the code area. In particular, a block is localized as the intersection between the line connecting its left and right TRBs and the line connecting its top and bottom TRBs. However, this localization approach highly relies on the starting and the ending locators in each row and column. It is highly possible that blocks may deviate from their original locations due to image distortion. Figure 3 illustrates the location error of a block in COBRA. We can see the estimated location (the yellow point at the intersection of two lines) of the green block is at the boundary of two blocks, which significantly deviates from the center of the block (the red point) and makes it difficult to determine the color of the block (*e.g.*, the color of the green block can be misjudged as red).

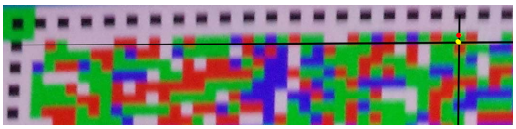
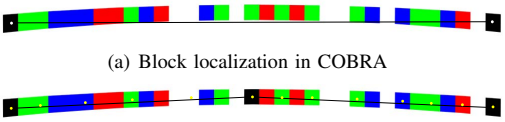


Fig. 3: Localization errors of COBRA

Our findings from the experiments reveal that local distortion is much weaker than global distortion. Inspired by this observation, we can add columns of TRBs in the code area. It can be imagined that the localization accuracy increases as the number of columns of TRBs increases but at the cost of decreased encoding capacity. Therefore, it is necessary to balance the localization accuracy and the encoding capacity. Through extensive experiments, we found that TRBs at two

sides and one more column of TRBs at the middle of the code area can satisfy the localization accuracy requirement even if the image distortion is very bad. In RainBar, we call the TRBs as *code locators* to differentiate them from COBRA. As shown in Figure 2, we add three columns of code locators represented by black blocks into the code area. Two adjacent code locators are separated by a colored block in white, red, green or blue. Figure 4 shows an example of block localization using COBRA and RainBar, respectively. We can easily observe that the use of one middle code locator could significantly improve the localization accuracy.



(a) Block localization in COBRA

(b) Block localization in RainBar

Fig. 4: A comparison between COBRA and RainBar for block localization

**Code area**: The code area is the main body of the color barcode frame, which is composed of blocks with encoded information. We use four colors (*e.g.*, white, red, green and blue) to encode data blocks in the code area. In RainBar, it is worth noting that blocks between two adjacent code locators are also belonging to the code area and contain encoded information.

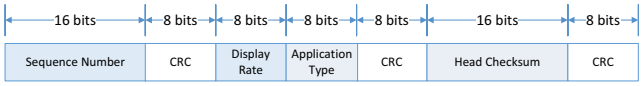



Fig. 5: Data structure of header

The code area of each color barcode frame contains header and data encoding information. The header stores labels and authentication information of a color barcode frame. Figure 5 shows the data structure of the header in the first color barcode frame. It contains multiple fields, including the sequence number of the frame, the display rate and the head checksum of the frame and CRCs (due to the importance of header information, we adopt a 8-bit CRC for every 16-bit data). A sequence number, which uniquely identifies a color frame, is used for reordering correctly-decoded frames during data recovery and/or locating the failed ones for retransmission. The most significant bit (MSB) of the sequence number is used to denote if the current frame is the last one of a complete data file. The remaining 15 bits can be used to represent $2^{15}$ different frames, which suffice for most applications.

The display rate ($f_d$) is compared to the capture rate ($f_c$) at the receiver. If $f_d \leq \frac{1}{2} f_c$, it means that each transmitted frame is captured at least twice, then we utilize blur assessment [7] to choose the best one and discard the others; If $f_d > \frac{1}{2} f_c$, it means that blocks in a transmitted frame may be captured in different frames at the receiver due to *rolling shutter*, then we keep all captured frames to reconstruct the original frames. The head checksum is used to check the integrity of the whole

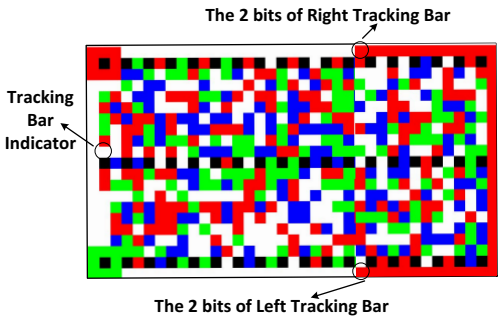



Fig. 6: Illustration of a captured frame composed of two consecutive frames

frame. Note that the headers of remaining frames do not have to include the application type and the display rate fields. Similar to RDCode, we use the RS codes [10] for in-frame error correction. Based on a finite field with 256 elements (one element has size of 1 byte), an RS$(n, k)$ code has the ability to correct up to $(n-k)/2$ error bytes and detect any combinations of up to $n-k$ error bytes, and all the encoded bytes in a block are called an RS message.

**Tracking bar**: The tracking bars are designed to address unsynchronized communication over screen-camera links. In practice, CMOS sensors are commonly used in cameras of off-the-shelf smartphones, and they use a method called *rolling shutter* to capture an image. Instead of taking a snapshot of the entire scene at one instant, *rolling shutter* scans the screen line by line to synthesize the complete image. In this case, when the display rate of color barcode frames on the screen is larger than half of the capture rate of the camera, a captured image may be composed of two frames, as shown in Figure 6. Hu et al. [8] made the first step to study the so-called *synchronization problem*, *i.e.*, how to correctly decode imperfect and mixed frames to obtain the original data information. The limitation of their work is that they only use two colors, namely white and black, to encode information, which severely limits the encoding capacity of a frame and thus the throughput.

In RainBar, we try to realize high throughput by increasing the display rate and solve the frame synchronization problem for color barcode frames. Note that as long as $f_d < f_c$, at most two partial frames will appear in a captured image at the receiver. In this paper, we design tracking bars in each color barcode frame to distinguish between two partial frames within each captured image and realize frame synchronization for color barcode frames.

Our experimental study shows that the decoding accuracy for blocks on the borders of a color barcode frame is unsatisfactory, however, it will also be wasteful if the border areas cannot be fully exploited. In RainBar, as shown in Figure 2, we put 4 tracking bars on the 4 borders of a color barcode frame to solve the synchronization problem. The tracking bars within one color barcode frame is composed of blocks in the same color. We use the last 2 bits of the sequence number to control the color of tracking bars used in a color barcode

frames, which is called the tracking bar indicator. The 2 bits can represent 4 colors (*i.e.*, white: 00, red: 01, green: 10, blue: 11) and thus tracking bars of any 4 consecutive color barcode frames are in different colors. We will describe how to use tracking bars to realize frame synchronization later.

**Encoding capacity analysis**: In RainBar, the blocks between code locators in each column also carry encoded information, which is not the case in COBRA. For the same size of screen, we can calculate that the size of code area in each frame of Rainbar has 2.5 more columns and 4 more rows of blocks than that of COBRA. Let's take a specific smartphone as an example. The resolution of a 5-inch SAMSUNG GALAXY S4 screen is 1920 $\times$ 1080, which can be divided into 147 $\times$ 83 blocks and each block is 13 $\times$ 13 pixels. The code areas in each frame of COBRA and RainBar have $(147-6)\times(83-6) = 10857$ blocks and 11520 blocks, respectively. That is, RainBar has an additional of 663 blocks that can carry 166 more bytes of data information than COBAR in each frame.

RDCode [9] divides the screen into $h \times h$ square blocks. This configuration, however, limits the adaption of frames on different sizes of screens, which would waste a lot of space on the screen. For example, given a 5-inch SAMSUNG GALAXY S4 smartphone with 147 $\times$ 83 blocks, RDCode divides the screen into $12 \times 6$ squares and each square contains $12 \times 12$ blocks. In this case, the code area only has $(12 \times 6 - 1) \times (12 \times 12 - 6) = 10508$ blocks, which is smaller than that of RainBar. Even worse, its effective code area is even smaller since a lot of blocks in RDCode are redundancy blocks used for error correction and cannot carry data information.

In summary, we can see that RainBar has the largest effective code area compared to existing color barcode systems. In the following subsections, we will present how to quickly and accurately decode data from captured images.

### C. Brightness Assessment and Corner Tracker Detection

After the frames are captured, RainBar first assesses the brightness of each frame. Because the brightness of each frame maybe different under different working conditions, *e.g.*, the brightness of the sender's screen, outdoor environments or indoor environments. To improve the decoding rate, we propose a HSV-based robust color recognition scheme, we need to calculate a threshold $T_v$ according to the brightness of each frame. In Section III-F, we will discuss how to calculate $T_v$ in detail.

Then, RainBar detects corner trackers of each frame and decodes blocks at the code area. We use the same fast corner detection method as in COBRA except that we only detect the top-left corner and the top-right corner (Please refer to Section 4.5 in [7] for the details). The header information, which is encoded in the first row between the two CTs, will be extracted accordingly. So, we can know the sequence number, the display rate, and other properties of each frame. For the bottom corners, they can be detected simultaneously when code locators are located.

### D. Blur Assessment and Frame Synchronization

The captured images cannot be decoded directly due to severe blur effects, image asynchronization, perspective distortion, and so on. All of these problems could significantly affect the decoding accuracy. Thus, the receiver should first pre-process the captured image by blur assessment or frame synchronization, and then decode data from the pre-processed images.

**Blur assessment**: As mentioned in COBRA [7], when the display rate of color barcode frames on the screen is smaller than half of the capture rate of the camera, the receiver is likely to capture more than one images from the same frame with different degrees of blur. It is wasteful of time and resource to process captured images of the same frame. Therefore, we adopt the same blur assessment method proposed in COBRA and select the one with the best quality from the images of the same frame for data decoding. The blur assessment method could improve the decoding accuracy and saves computation resources. Please refer to Section 4.4 in [7] for more details of blur assessment.

**Frame Synchronization**: In RainBar, the header of each captured frame is extracted before frame synchronization, and thus the tracking bar indicator of the frame can be identified. we use 00, 01, 10, and 11 for consecutive color barcode frames, respectively. Since a frame with 11 tracking bars is followed immediately by a a frame with 00 tracking bars, we assume that the difference between 11 and 00 is 1, but the difference between 00 and 11 is still 3. Therefore, the difference of tracking bars between any two consecutive color barcodes is 1. For a captured frame, we can identify the color of the left and the right tracking bars in each row. Let $d_t$ denote the difference between the tracking bar in this row and the tracking bar indicator of the frame. We have:

- When $d_t = 0$, this row belongs to this frame.
- When $d_t = 1$, this row and below belong to the next frame.
- When $d_t \geq 2$, which is actually rare, we simply drop this frame and resort to retransmission.

Figure 6 shows an example of two consecutive frames captured in one single image at the receiver. By checking the block of tracking bars in each row with the last 2 bits of the sequence number, we can correctly determine which frame that a row of color blocks belong to. Finally, all captured frames can be arranged in the correct sequence.

### E. Progressive Code Locator Detection and Localization

We design code locators to help the localization of blocks in code area. Therefore, accurate location of locators is critical to color barcode decoding. Note that we need to find the center location of each block rather than locations at the edge. This is because the locations at the edge are easily blurred on the captured images, which greatly affect color recognition of each block. Therefore, the location of a block refers to the block's center location in this paper.

The captured images may suffer from severe perspective distortion, especially when the screen and the camera are not in parallel. It makes the localization problem very challenging for code locators. To solve the problem, we propose a *progressive localization* algorithm that progressively localizes each locator based on its upper locator and offsets the localization errors to obtain its accurate location.

Figure 7 illustrates how the *progressive localization* algorithm works for locator localization. Take the code locators at the left column as an example. We can consider the center black block in the corner tracker as the first code locator, the location of which is already known in the *corner tracker detection* phase. The *progressive localization* algorithm works as follows.

- Step 1: Calculate the location of the next code locator based on the first code locator's location;
- Step 2: Recalculate the code locator's location by using the *location correction* algorithm, and then calculate a new code locator's location based on the current code locator's corrected location;
- Step 3: Repeat step 2 until the last code locator in this column is localized.

We describe how to calculate a code locator's location (say $i+1$) based on the location of its upper code locator (say $i$). Let $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ denote the locations of the $i$-th and the $(i+1)$-th code locators, respectively. As shown in Figure 7, let $BST$ denote the vertical length of the $i$th code locator. Since two adjacent code locators are separated by one colored block, we can get $(x_{i+1}, y_{i+1}) = (x_i, y_i + 2 \cdot BST)$.

However, due to perspective distortion, the calculated location may not be the center location of the code locator. To obtain its accurate location, we propose a *location correction* algorithm to reduce the localization error caused by perspective distortion. Our algorithm is based on the observation that the top and bottom edges, the left and right edges of a distorted block are still in parallel, so the center location of a block will not change after being distorted. Inspired by the *K-means* algorithm [11], our *location correction* algorithm is run as follows.

- Step 1: Find the pixels within a rectangular centered at $(x_{i+1}, y_{i+1})$ with the edge of length $BST$;
- Step 2: Recalculate $(x_{i+1}, y_{i+1})$ as the mean of locations of pixels in black color within the rectangular area;
- Step 3: Repeat Steps 1 and 2 until $(x_{i+1}, y_{i+1})$ converges to a stable value.

The convergence of the *location correction* algorithm can be easily proved, and the accurate location of the code locator will be finally obtained. By progressively locating code locators one by one, all the locations of code locators in the left column can be calculated. Similarly, the locations of code locators in the right column can be determined.

We next describe how to calculate the locations of code locators in the middle column. The first code locator in the middle column should be calculated first. It is supposed to be at the midpoint of the center locations of the left and right corner trackers. However, this is not true for captured images with distortions. Our basic idea is to first find the midpoint of the center locations of the left and right corner trackers, and then search around the midpoint to find the first code locator.

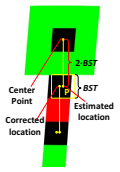



Fig. 7: Illustration of the progressive localization algorithm

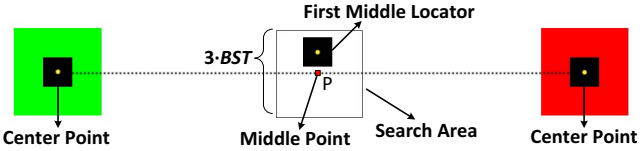



Fig. 8: Finding the first code locator in the middle column

Let $P$ denote the midpoint of the center locations of the left and right corner trackers. As shown in Figure 8, we scan pixels line by line starting from the top-left pixel in the rectangular centered at $P$ with the edge of length $3 \cdot BST$, where $BST$ is the average of the vertical length of the first code locators in the left and right columns. In RainBar, we choose $3 \cdot BST$ as the edge length of the rectangular, which is a compromise of localization accuracy and resource consumption. A larger edge can increase the accuracy of finding the first code locator but at the cost of more computation time and resources. Our experimental results also validate that $3 \cdot BST$ is a good choice for the edge length of the rectangular. We start scanning from the top-left pixel. If it's not black, we keep scanning the next pixel; otherwise, we search pixels in four directions from the current pixel. Note that the current pixel may be just a noise point, which can be discarded and recovered by using its surrounding pixels. In each direction, the search terminates until it is found that the color of a pixel is not black.

Let $b_u$, $b_l$, $b_d$ and $b_r$ denote the number of pixels that the searches have moved in the upward, left, downward and right directions, respectively. The searched pixels in four directions is a square. In RainBar, the sender can adaptively change the size of a block. Let $B_{min}$ and $B_{max}$ denote the minimum and maximum sizes of blocks, respectively. Then, if $B_{min} \leq b_u + b_d \leq B_{max}$ and $B_{min} \leq b_l + b_r \leq B_{max}$, we can conclude that the searched black pixels belong to the first code locator; otherwise, they are just noisy points. Once the first code locator is found, we use our *location correction* algorithm to find its center location, and then the other code locators in the middle column can be progressively localized by using the *progressive localization* method.

### *F. HSV-based Robust Code Extraction*

After obtaining the locations of code locators, the next step is to decode the encoded information in the code area. This phase consists of two steps: *localization* of each block and *color recognition*.

**Block localization**: In RainBar, we design three columns of

code locators to improve the localization accuracy of blocks on a captured image. Basically, instead of localizing all blocks in the code area using the code locators in the left and right columns, RainBar uses the code locators in the left and middle columns to localize blocks in the left-half code area, and uses the code locators in the middle and right columns to localize blocks in the right-half code area.

Take a row in the left-half code area as an example and let $(x_l, y_l)$ and $(x_m, y_m)$ denote the locations of left and middle code locators in this row, respectively. Assume there are a total of $n-1$ blocks between the left and middle code locators in this half-row. The location of the $i$-th block starting from the left, denoted by $(x_i, y_i)$, is calculated as follows

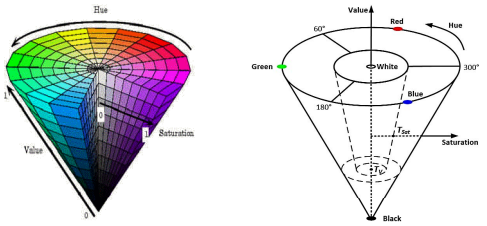
$$x_i = (1 - \frac{i}{n})x_l + \frac{i}{n}x_m \quad \text{and} \quad y_i = (1 - \frac{i}{n})y_l + \frac{i}{n}y_m. \quad (1)$$

Similarly, we can calculate the locations of all blocks in the right-half area. Note that code locators in each column are placed separately, so there are rows that do not have code locators. For these rows, we first calculate the locations of the left-most block, the middle block and the right-most block, and then use them to calculate the locations of other blocks using Equation (1). After extensive testings over many captured images, we found that although an image may be globally severely distorted, its local regions are not. In other words, each block itself is not severely distorted, and the centers of adjacent blocks are almost on a line. Based on this observation, we can estimate the location of the left-most block of a row as the average of the locations of its adjacent code locators. The locations of the middle block and the right-most block of this row can be estimated in the same way. Finally, the locations of all blocks in this row can be obtained.

**HSV-based robust color recognition**: Recall that the encoded data are mapped to blocks with different colors in a color barcode frame. To decode data correctly, the color of each block should be determined. In practice, it is challenging to recognize the colors of blocks since captured images may suffer from severe blur and noise. We perform two operations: *block denoising* and *HSV-based robust color recognition*, to realize robust and accurate code extraction.

In RainBar, we adopt the simple and lightweight *mean filter* for block denoising. Note that recognizing the color of a block is actually recognizing the color of the pixel at the center location of the block, so a block denoising is equivalent to a pixel denosing. The basic idea of *mean filter* is to simply replace a pixel value with the mean value of its neighbors including itself, which is very suitable for denoising purpose for our system. This is because a center pixel of a block and its neighbor pixels are supposed to have the same value, so the noise effects on the center pixel can be reduced after averaging of neighbor pixels. Moreover, *mean filter* is simple and easy to be implemented, which consumes little computation resources. In our implementation, we use a $3 \times 3$ square kernel for the *mean filter*.

We then recognize the color of the denoised pixel at the center of each block. The designed color barcode frame only has five colors, so we only need to design a robust *color classifier* that can well differentiate between these five colors. After we carefully compared several color models, such as Lab, HSV and RGB, and found that HSV model is a perfect model for our five-color differentiation.




(a) HSV model (b) Color Recognition

Fig. 9: Illustration of the color space of HSV model and our color recognition

Figure 9 (a) shows the color space of HSV model where *hue* represents the color, *saturation* refers to the dominance of *hue* in the color, and *value* indicates the lightness of the color. An important property of HSV model is that it just uses *hue* to indicate different colors other than white and black, so we can easily differentiate between red, green and blue by using the value of *hue*. As for white and black, *saturation* and *value* can be used to differentiate between them.

As shown in Figure 9 (b), the color space of HSV can be divided into five subspaces and each subspace is corresponding to one color. We first convert the color of a pixel from RGB model to HSV model and normalize the values of *saturation* and *value*. Let $T_v$ and $T_{sat}$ denote the thresholds of *value* and *saturation*, respectively. When *value* is smaller than $T_v$, the pixel is in black; otherwise, if *saturation* is smaller than $T_{sat}$, the pixel is in white. If the pixel is not in black or white, we further check *hue*. If $60^o < hue < 180^o$, the pixel is in green; if $180^o < hue < 300^o$, the pixel is in blue; otherwise, the pixel is in red.

In our experiments, we found that HSV model has several good properties. First, for a pixel in the original color barcode, the values of the corresponding pixel in the RGB model on the captured images change a lot under different illuminance levels. However, when we convert the colors of the pixels from RGB model to HSV model, we found that only *value* changes a lot while *saturation* and *hue* almost remain the same. Second, $value$ for pixels in colors other than black, such as white, red, green and blue, are almost equivalent under the same illuminance level. Thus, we can conclude that illuminance levels could not affect the differentiation between white, red, green and blue, but only affect the differentiation between black and other colors. This is a very good property which implies that HSV model is robust to illuminance level changes, especially for the differentiation of red, green and blue colors. In order to carefully differentiate black from other color, $T_v$ should be carefully selected.

As shown in Figure 9 (b), the value of $T_v$ varies between the bottom point and the center of the top surface when

the illuminance level changes. Since pixels in white, red, greed and blue have the same *value*. The value of $T_v$ can be considered as a linear combination of the average *value* of black pixels and the average *value* of non-black pixels. To efficiently calculate $T_v$, we divide a captured frame into four equal regions: top-left, top-right, bottom-left and bottom-right ones. In each region, we randomly sample $N$ pixels and calculate the *value* of each pixel. For pixels with *value* smaller than 0.1, they are in black color and we calculate their average *value*, denoted by $V_b$. For the other pixels, we also calculate their average *value*, denoted by $V_o$. Then $T_v$ of the captured frame can be estimated as follows:

$$T_v = \mu V_b + (1 - \mu)V_o, \tag{2}$$

where $0 < \mu < 1$ is an coefficient that balances the threshold between the black color and other four colors. In our experiments, we found that $\mu = 0.55$ is a good choice. Since *saturation* changes little when the illuminance level changes, $T_{sat}$ can be seen as a constant value. In our experiments, $T_{sat}$ is set to be 0.41. We also conducted extensive experiments and found that using Equation (2) can find suitable value for $T_v$ and the resulting color recognition accuracy is very high.

COBRA [7] uses HSV enhancement that improves the quality of images to improve the accuracy of color recognition, which however is very time-consuming. For example, it takes 16ms for COBRA to decode a captured image while 12ms is used for HSV enhancement, which severely affects the processing performance for barcode decoding. In contrast, we do not need to perform HSV enhancement for image quality improvement, but just determine which subspace the HSV model of blocks belong to, and it achieves higher accuracy on color recognition with high decoding process performance. RDCode [9] uses color palettes to decide the colors of blocks to have a high accuracy of color recognition. However, 4 blocks in each $h \times h$ square of blocks are used for color palettes, which severely reduces the encoding capacity in each color barcode frame.

## IV. Implementation and Performance Evaluation

In this section, we conduct extensive experiments to evaluate the performance of RainBar and compare it with the first notable color barcode system COBRA.

We implement RainBar on Android smartphone platforms using Eclipse with the Android SDK and JAVA compiler. Two Samsung Galaxy S4 smartphones running Android 4.4.2 are used in our experiments. The resolution of each screen is $1920 * 1080$, and the camera is 13 MegaPixel. An application of Rainbar is developed for both the sender and the receiver, which requires 1MB memory on the phone after being installed and 6MB RAM allocation while running the application.

At the sender, the drawing procedure takes much more time than the encoding procedure. To make sure that the display rate can achieve around 30 fps, we use four threads running the drawing procedure simultaneously. Each thread takes charge of displaying one-fourth blocks of a frame in different areas. By using multi-thread, the average time to encode and draw a frame is about 31ms. In practice, different smartphones may have different camera capture rates. To maximize the communication throughput, RainBar offers an option that allows the sender to choose the phone type of the receiver such that an appropriate display rate for sending streams of color barcode frames can be chosen.

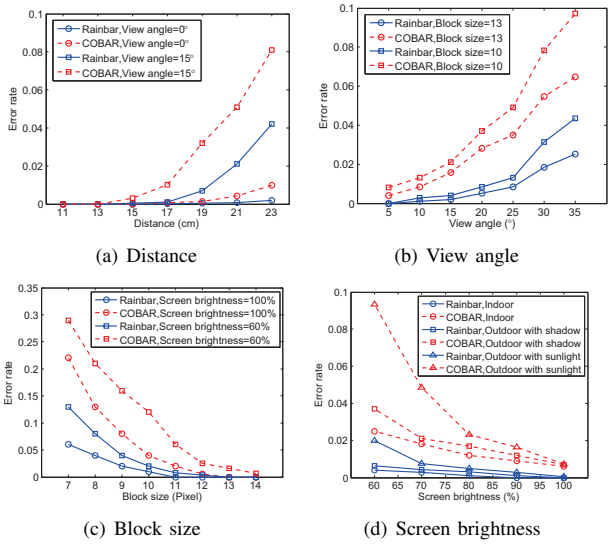
At the receiver, we provides two options: the real-time decoding mode and the buffered decoding mode. In the real-time decoding mode, a thread captures the barcodes from the sender continuously and at the same time, another thread decodes the captured frame. This means that the decode time must be smaller than the display time. Otherwise, the next frame maybe lost. In the buffered decoding mode, we save the captured barcodes in the smartphone in the form of video, then we read each frame of video for decoding. Although we design and implement light-weight algorithms for continuous image decoding and introduce multi-thread technology, it still cannot meet the requirements because of the limited computation power of our smartphones. For example, the total time to decode a frame is about 80ms, it means that the display rate cannot be more than 12 fps in the real-time decoding mode. Hence, as in [7], we will evaluate the decoding rate and the throughput of the system in the buffered decoding mode and the average decoding time in the real-time decoding mode.

Our evaluation is focused on the following metrics: 1) *decoding/error rate*: the percentage of correctly/wrongly decoded data in the total amount of data contained in a color frame; 2) *throughput*: the average amount of data successfully decoded per second in the received frames. Note that *error rate* equals to 1 minus the *decoding rate*. We evaluate these metrics on several important factors: the block size ($b_s$), the view angle ($v_a$) and distance ($d$) between the sender and the receiver, the screen brightness ($s_b$) and the display rate ($f_d$).

### A. Error Rate Comparison between RainBar and COBRA

We first evaluate the effects of parameters on the error rate and compare the performance between RainBar and COBRA. The default values for parameters are set as follows: $f_d = 10$fps, $b_s = 12 * 12$ pixels, $d = 12$cm, $v_a = 0^o$ and $s_b = 100\%$ in an indoor environment. When evaluating one parameter, the other parameters do not change.

As shown in Figure 10(a) and (b), we can observe that the error rate increases as the view angle or the distance increases, which is because the increase of the view angle or the distance will aggravate image distortion and blur, making data decoding more difficult. We also observe that the effect of view angle is more serious for a smaller block size. As shown in Figure 10(c) and (d), the error rate decreases when the block size or the screen brightness increases. This is because that data decoding is more difficult for smaller blocks and lower screen brightness. Note that the error rate is much higher when the images are taken at outdoor environments compared to indoor environments.




(a) Distance (b) View angle

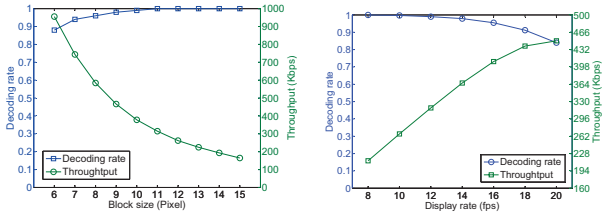
(c) Block size (d) Screen brightness

Fig. 10: The effects of different parameters on decoding error rate

In all experiments, we observe that the error rate of RainBar is smaller than that of COBRA, and COBRA is more sensitive to working condition changes than RainBar. Therefore, we can conclude that RainBar is more robust to changes of capturing conditions, such as view angle, distance and screen brightness.

### B. Decoding Rate and Throughput of RainBar

The decoding rate and the throughput mainly relate to block size and display rate, so we evaluate these two performance metrics on the two parameters. The other parameters are set as follows: $d = 12cm$, $v_a = 0^o$, and $s_b = 100\%$. The display rate is 10 fps when we evaluate the block size, and the block size is $12 * 12$ pixels when we evaluate the display rate.




(a) Block size (b) Display rate

Fig. 12: The effects of block size and display rate on throughput and decoding rate

Figure 12(a) shows the effects of block size on the throughput and the decoding rate of RainBar. We can see that the decoding rate increases when the block size increases and reaches 100% when the block size is $11 * 11$ pixels. This is because that the larger the block is, the less severe of the distortion and the easier of the decoding. When the block size is large enough, we can still decode each block correctly. In contrast, the throughput decreases when the block size increases. This is because that the number of blocks decreases when the block size increases, and therefore the encoded data in each color barcode frames becomes less.

Figure 12(b) shows the effects of display rate on the throughput and the decoding rate of RainBar. The throughput increases as the display rate increases since more number of frames are captured at the receiver per unit time. In contrast, the decoding rate decreases as the display increases since it is more difficult to decode data for a higher display rate. Note that RainBar has a high decoding rate as it is still above 91% when the display rate is as large as 18 fps.

### C. Comparison of Decoding Rate and Throughput between RainBar and COBRA

As shown in Figure 11(a) and (b), we can observe that the decoding rate and the throughput of RainBar are higher than that of COBRA. Although both the decoding rates of COBRA and RainBar decrease as the display rate increases, COBRA drops more quickly than RainBar. Note that the throughput of RainBar increases as the display rate increases, however, the throughput of COBRA first increases and then decreases as the display rate increases. This is because that RainBar solves the frame synchronization problem and works well even the display rate is larger than half of the display rate, while COBRA does not. As shown in Figure 11(c) and Table I, we can also observe that both the decoding rate and the throughput of RainBar are better than those of COBRA.

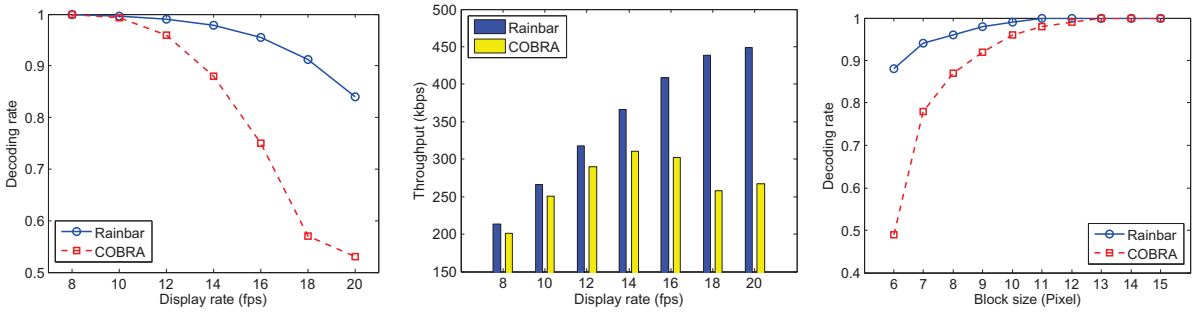
### D. The Average Decoding Time Per Frame

In this subsection, we evaluate the average decoding time per frame in the real-time decoding mode with single thread and multiple threads, respectively. Due to the space limitation, please refer to our full technical report [12] for the details.

## V. Adapting RainBar to Applications: A Discussion

Due to the space limitation, we only briefly discuss the application case of text file transfer in this section. More details and more cases (*e.g.*, image file and audio file transfers) can be found in our technical report [12].

Text file transfer is a typical application between smartphones. Without relying on WiFi and other cumbersome network configurations, text files can be encoded as color barcodes transmitted from one smartphone to another quickly and securely. Typically, most text files on smartphones have a small size (*e.g.*, around 1MB), so hundreds of color frames will be enough for encoding the whole file. However, text file transmission requires extremely high accuracy since even one-bit decoding error will lead to a wrong character, which further affects the readability of text files. RDcode [9] proposed a Tri-level (intra-block level, inter-block level and inter-frame level) error correction mechanism to guarantee the transmission reliability, which is at the cost of reduced throughput/goodput (a lot of redundance blocks occupy the code area). More seriously, the text file can never be recovered at the receiver when the corruptions exceed the error correcting ability of the codes. In RainBar, we use RS$(n, k)$ code for only intra-frame level error correction and retransmit failed frames including




(a) Display rate (b) Display rate (c) Block size

Fig. 11: The effects of block size and display rate on throughput and decoding rate

| **Block size (pixels)** | $6*6$ | $7*7$ | $8*8$ | $9*9$ | $10*10$ | $11*11$ | $12*12$ | $13*13$ | $14*14$ | $15*15$ |
|---|---|---|---|---|---|---|---|---|---|---|
| **RainBar's throughput (kbps)** | 955.68 | 743.85 | 584.32 | 466.48 | 378.18 | 314.67 | 262 | 224.67 | 193.33 | 164.67 |
| **COBRA's throughput (kbps)** | 518.09 | 597.48 | 510.40 | 420.13 | 350.08 | 290.73 | 245.52 | 212 | 181.33 | 153.33 |
| **Enhancement** | 84.46% | 24.5% | 14.48% | 11.03% | 8.03% | 8.23% | 6.71% | 5.97% | 6.62% | 7.39% |

TABLE I: The comparison of the throughput between RainBar and COBRA under different block sizes

frames that are not captured and cannot be corrected. We argue that retransmission is better than excessive coding for guaranteeing the transmission reliability of text files.

The key to retransmission is to establish a feedback channel over the screen-camera links, transmitting feedback information from the receiver to the sender and indicating which frame needs retransmission. Observe that smartphones are equipped with LEDs, we propose to use LED-Light sensor links as the feedback channel. Through the experiment, we found that the brightness and flicker frequency of LEDS in smartphone can be modulated, the brightness can vary from 0 to 20K Lux and the flicker frequency can reach around 10 Hz. In addition, the light sensor is also enough sensitive to identify 1 lux change and the sampling frequency can reach dozens of Hz. The calculating data indicate that the time to retransmit a sequence number can be smaller than 0.5s. Based on a careful analysis, it is shown that the efficiency of retransmission is higher than excessive coding while achieving universal transmission reliability. Our ongoing work is to fully implement this retransmission mechanism on off-the-shelf smartphones.

## VI. Conclusions

In this paper, we designed and implemented a robust high-throughput visual communication system using color barcodes. We optimized the layout for color barcodes to improve encoding capacity of each barcode and solved the frame synchronization problem to fully utilize the transmission capacity of the sender. A progressive locator localization algorithm and a HSV-based robust code extraction scheme were proposed to realize fast and accurate code extraction. Analysis and experimental results show that RainBar outperforms the existing color barcode system in terms of decoding rate and throughput.

## Acknowledgment

Kui's research is supported in part by US National Science Foundation under grants CNS-1421903, CNS-1318948 and CNS-1262275. Qian's research is supported in part by National Natural Science Foundation of China (Grant No. 61373167, 61373169), Natural Science Foundation of Hubei Province (Grant No. 2013CFB297) and Wuhan Science and Technology Bureau (Grant No. 2015010101010020). Zhibo's research is supported in part by Fundamental Research Funds for the Central Universities (No. 2042015kf0016).

## References

[1] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005.

[2] H. Elgala, R. Mesleh, and H. Haas, "Indoor broadcasting via white leds and ofdm," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1127–1134, 2009.

[3] A. Ashok, M. Gruteser, N. Mandayam, J. Silva, M. Varga, and K. Dana, "Challenge: Mobile optical networks through visual mimo," in *Proc. of MobiCom*, 2010, pp. 105–112.

[4] Barcode payment service, http://gigaom.com/2012/05/30/paypal-rolls-out-barcode-payments-in-the-uk/.

[5] S. L. Ghìron, S. Sposato, C. M. Medaglia, and A. Moroni, "NFC ticketing: A prototype and usability test of an NFC-based virtual ticketing application," in *Proc. of NFC*, 2009, pp. 45–50.

[6] B. Zhang, K. Ren, G. Xing, X. Fu, and C. Wang, "SBVLC: Secure barcode-based visible light communication for smartphones," in *Proc. of INFOCOM*, 2014.

[7] T. Hao, R. Zhou, and G. Xing, "COBRA: color barcode streaming for smartphone systems," in *Proc. of MobiSys*, 2012, pp. 85–98.

[8] W. Hu, H. Gu, and Q. Pu, "LightSync: unsynchronized visual communication over screen-camera links," in *Proc. of MobiCom*, 2013, pp. 15–26.

[9] A. Wang, S. Ma, C. Hu, J. Huai, C. Peng, and G. Shen, "Enhancing reliability to boost the throughput over screen-camera links," in *Proc. of MobiCom*, 2014, pp. 41–52.

[10] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*. Wiley, 1999.

[11] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 3, pp. 433–439, 1999.

[12] Q. Wang, M. Zhou, K. Ren, T. Lei, J. K. Li, and Z. B. Wang, Technical Report - RainBar: Robust Application-driven Visible Communication using Color Barcodes, http://web.eecs.utk.edu/~zwang32/publications/rainbar.pdf.