

# ChromaCode: A Fully Imperceptible Screen-Camera Communication System

Kai Zhang\*  
Tsinghua University  
nezharen1995@gmail.com

Chenshu Wu\*  
University of Maryland,  
College Park  
wucs32@gmail.com

Chaofan Yang  
Tsinghua University  
yangcf10@gmail.com

Yi Zhao  
Tsinghua University  
zhaoyi.yuan31@gmail.com

Kehong Huang  
Tsinghua University  
huangkh13@gmail.com

Chunyi Peng  
Purdue University  
chunyi@purdue.edu

Yunhao Liu  
Tsinghua University&MSU  
yunhao@greenorbs.com

Zheng Yang  
Tsinghua University  
hmilyyz@gmail.com

## ABSTRACT

Hidden screen-camera communication techniques emerge as a new paradigm that embeds data imperceptibly into regular videos while remaining unobtrusive to human viewers. Three key goals on imperceptible, high rate, and reliable communication are desirable but conflicting, and existing solutions usually made a trade-off among them. In this paper, we present the design and implementation of CHROMACODE, a screen-camera communication system that achieves all three goals simultaneously. In our design, we consider for the first time color space for perceptually uniform lightness modifications. On this basis, we design an outcome-based adaptive embedding scheme, which adapts to both pixel lightness and regional texture. Last, we propose a concatenated code scheme for robust coding and devise multiple techniques to overcome various screen-camera channel errors. Our prototype and experiments demonstrate that CHROMACODE achieves remarkable raw throughputs of >700 kbps, data goodputs of 120 kbps with BER of 0.05, and with fully imperceptible flicker for viewing proved by user study, which significantly outperforms previous works.

## CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Networks** → *Mobile networks*;

\*The first two authors are co-primary authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

*MobiCom '18, October 29-November 2, 2018, New Delhi, India*

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-5903-0/18/10...\$15.00

<https://doi.org/10.1145/3241539.3241543>

## KEYWORDS

Screen-camera communication; hidden visible communication; non-intrusive visible communication

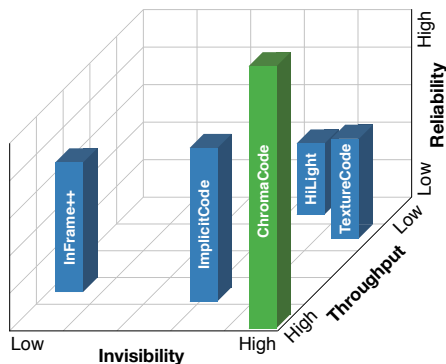
### ACM Reference Format:

Kai Zhang, Chenshu Wu, Chaofan Yang, Yi Zhao, Kehong Huang, Chunyi Peng, Yunhao Liu, and Zheng Yang. 2018. ChromaCode: A Fully Imperceptible Screen-Camera Communication System. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India*. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3241539.3241543>

## 1 INTRODUCTION

Every day, billions of videos are generated, broadcast, and watched over electronic visual displays such as phone or tablet screens, computer monitors, TVs, and electronic advertising boards. As digital marketing evolves, people continue to see trends favoring video, especially in the era of mobile Internet. Today such videos are mainly for viewing only, but we look forward to an emerging paradigm of *simultaneous viewing and communication* with great demands to allow movie audiences to refer to a website for more contents, TV programs to convey interactive information, and advertisers to provide extra details for an advertising video, all during watching.

If enabled, the new paradigm will reform various existing applications and foster new possibilities. For example, it may change the digital advertising industry. Currently, it is reported that 65 percent of people skip online video advertising, mainly due to the product placement's intrusiveness [27]. Delivering advertising content by side information without tampering primary video itself will not only guarantee viewing experience but also enhance user interaction, a key to attract consumers' attention according to a recent research [2]. Interactive TV programs and games will also be renovated, when users can perceive multi-dimensional contents for engagement with high throughput and low latency.



**Figure 1: A comparison of state-of-the-art works**

All these necessitate a fully-imperceptible, high-rate, and reliable screen-camera channel to deliver the new paradigm.

A common real-life example towards this paradigm is Quick Response (QR) code [13]. QR code spatially occupies a small area in the corner of a screen, or temporally appears as a large image in the end of the primary video contents. However, QR code is obtrusive for viewing and inefficient in terms of data rate.

Recent efforts [28, 35, 38, 42] attempt to enable hidden screen-camera communication by leveraging two facts: (1) Due to the flicker-fusion property of Human Vision System (HVS), human eyes cannot resolve light intensity fluctuations, but instead perceive the averaged luminance when it alternates beyond a certain frequency of about 40-50Hz in typical scenarios [14]. (2) Today, many off-the-shelf displays support 120Hz or higher refresh rates, and commodity smartphone cameras can support mega-pixel resolution images at a high capture rate. The above two facts result in an opportunity to embed data into high frame rate primary videos unobtrusively. Leveraging these properties, InFrame++ [42] achieves high throughput though with noticeable flicker, while TextureCode [35] and HiLight [28] both ensure invisibility yet at the expense of throughput.

Albeit inspiring, previous works achieve their results due to different trade-offs required to overcome the challenges in imperceptible communication and suffer from several limitations. First, perceptible flicker still remains in practice. Second, the throughput remains low. Third, they cannot recognize the code reliably under distorted screen-camera channel. Fig. 1 shows qualitative comparison of state-of-the-art works. None of previous studies achieve the three goals simultaneously. And among them, TextureCode[35] embeds in only partial frames while ImplicitCode[38] is applicable to only grayscale videos.

In this work, we aim to achieve all three goals on unobtrusive, high-rate, reliable screen-camera communication while retaining video watching experience to users. Our design and implementation of embedding data into video frames excel in three unique aspects. First, we consider for the first

time color space for lightness changes (Specifically, I in HSI color space, L in HSL color space, Y in YUV color space, or  $L^*$  in CIE 1976  $L^*a^*b^*$  color space, etc). The key to relax the tension between screen-to-camera data communication and screen-to-eye vision channel lies in the distribution of color space. Existing solutions, however, never explore along this direction but use common color spaces to embed bits. However, we tackle this problem fundamentally different from others. We explore the color space and make it right at the beginning with a uniform color space. As color is merely a subjective feeling of HVS, uniform color space allows better tolerance of complimentary lightness changes because, by definition, it is perceptually uniform. Particularly, we adopt CIE 1976  $L^*a^*b^*$  color space (a.k.a CIELAB or LAB) [10], along with CIEDE2000, the latest and most accurate color difference formula currently available [12].

The second is a full-frame adaptive embedding mechanism. Invisibility is usually guaranteed by using less space and applying smaller lightness changes. Yet to gain throughput, data should be embedded in as much space as possible with appropriate lightness changes for the full frame. Applying a fixed lightness change over all pixels, however, would frequently lead to too large perceivable color changes or too small undetectable color differences, due to a previously unnoticed fact that an identical lightness change to a pixel does not necessarily produce the same color differences, depending on the original lightness and surrounding texture. Different from previous works, we manipulate lightness adaptively for each pixel in an outcome color difference driven manner and account for both original lightness and texture, which reduces user-perceivable flicker to minimum while allows full-frame embedding for high rate transmission.

Third, we achieve reliable high-rate communication. Hidden screen-camera channel usually exhibits low signal-to-noise ratio and thus suffers from significant random and burst errors. Previous works mainly focus on invisible embedding or high-rate transmission, but pay little attention to reliability, which is, though, of great importance for practical applications. To ensure reliable communication, we design a concatenated error correction code scheme consisting of Reed-Solomon code and convolutional code, providing high error-correcting capability. We further devise interleaving technology to deal with burst errors and propose techniques to eliminate channel distortions including Moiré pattern, projection distortions, rolling shutter effect, etc, which together yield highly reliable and efficient transmission.

We have implemented CHROMACODE on commodity monitors and camera-equipped smartphones and conducted extensive evaluation including user studies. CHROMACODE guarantees imperceptible flicker under various conditions, confirmed by a user study of 20 users. In the meanwhile, CHROMACODE yields remarkable raw throughput of 777 kbps and

data goodput of 120 kbps, and a low bit error rate (BER) of 0.05, significantly outperforming previous works under the same conditions. We believe such achievements promisingly shape a practical technique for the emerging paradigm of simultaneous viewing and communication, which will become more appealing in the near future.

In summary, the main contributions are as follows:

- We introduce for the first time uniform color space for unobtrusive data embedding. Different from all early works, we embed bits to pixels using the most accurate color difference formula CIEDE2000 in a perceptually uniform color space CIELAB.
- We design a novel adaptive embedding scheme in an outcome-based philosophy, which accounts for both pixel lightness and frame texture and ensures flicker invisibility over the full frame.
- We achieve reliable data transmission by a concatenated code mechanism together with multiple techniques to handle screen-camera channel errors.
- We prototype a full system CHROMACODE and compare it with previous works via experimental evaluation. We demonstrate that CHROMACODE achieves remarkably higher throughputs and goodputs with significantly lower BER than previous arts, while guaranteeing better viewing experience.

In the following, we first provide an overview of CHROMACODE in §2. Then we introduce adaptive embedding in §3 and robust coding in §4. Implementation and experiments are presented in §5 and §6, respectively. We review the related works in §7 and conclude in §8.

## 2 CHROMACODE DESIGN

### 2.1 Design Goals

We aim to achieve the following goals in CHROMACODE.

1) **Fully Unobtrusive.** The screen-camera communication channel is visually hidden from users. Neither the embedded images monopolize any screen space (either spatially or temporally), nor the users perceive any flicker (over the full screen) when viewing the video.

2) **High Rate.** High data rate enables transmission of large volume data and results in low latency, which is critical to practical real-time interaction applications. A high data rate should be achieved regardless of the video content. In particular, we aim at up to 100 kbps, allowing delivery of e.g., a 1kb URL within 10ms.

3) **Reliable.** Previous works mainly focus on the compromise between unobtrusiveness and high rate transmission. In CHROMACODE, we further keep in mind robust communication with low bit error rates. A reliable communication channel is anticipated under various interferences such as

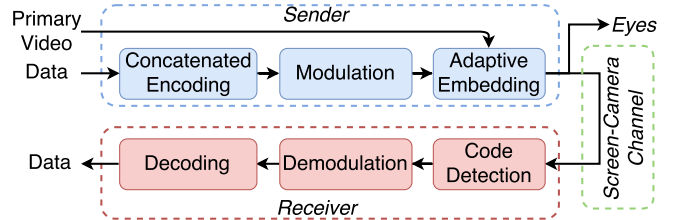


Figure 2: CHROMACODE overall architecture

rolling-shutter effects, Moiré pattern, projection distortions, screen-camera distances, etc.

All three goals enable side-channel to users and provide more possibility and flexibility for content providers. However, it is extremely challenging to simultaneously achieve the three conflicting goals. For example, reducing lightness changes for full unobtrusiveness may hurt reliable and high-rate communication. In contrast, applying larger lightness changes enhances reliability but may lead to flicker.

### 2.2 Design Overview

Fig. 2 illustrates the overall workflow of CHROMACODE. CHROMACODE employs commodity screens as transceiver and off-the-shelf smartphones as receiver. The sender takes primary video together with secondary side information (referred as video and data respectively hereafter) as inputs. Data are first encoded by the *concatenated error correction coding* scheme. The encoded data are then *modulated* as an imagery code, which is *adaptively embedded* into the primary video in a visually unobtrusive way, producing the ultimate output at the sender (displayed on the screen and streamed over the screen-camera channel).

On the receiver side, a video clip is captured by camera. The receiver first extracts the embedded imagery code from the captured video frames by *code detection*. The extracted code is then *demodulated* and *decoded*, recovering the embedded bits. During the entire receiving procedure, users can normally watch the video, without feeling affected by the screen-camera transmission.

The core of CHROMACODE is to address two tensions: (1) unobtrusive and high-rate bit embedding, and (2) high-rate and reliable communication. For the first one, we tackle it through spatially adaptive embedding in uniform color space (§3). For the second one, we address it by concatenated code and robust code detection (§4).

## 3 UNOBTRUSIVE BIT EMBEDDING

In this section, we discuss how to invisibly embed data frames, in the form of image codes as shown in Fig. 5, into normal videos without affecting the viewing experience. We first assume we already have the data frames (consisting 0 and 1 data bits as white and black cells) for embedding and leave its generation and modulation to next section.

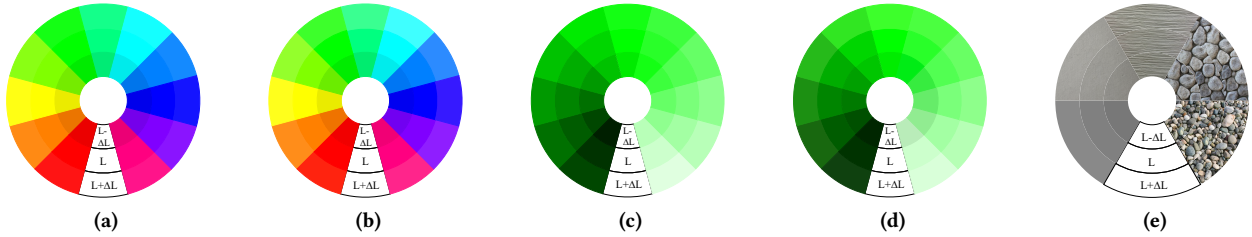


Figure 3: Perception diversity w.r.t. hue palettes in (a) HSL space and (b) LAB space w/ CIEDE2000, lightness palettes in (c) HSL space and (d) LAB space w/ CIEDE2000, and (e) texture complexity.<sup>1</sup>

### 3.1 Bits Embedding to Pixels

Inspired by previous works [42], we also embed data frames by altering the lightness of carrier video, exerting a series of time-variant high-frequency lightness changes that are imperceptible to human eyes yet detectable to cameras. Briefly, the carrier video is first multiplexed into a high frame rate of, e.g., 120 frames per second (fps). Each data frame is rendered by a pair of successive frames with contrary lightness changes  $\pm\Delta L$ , i.e., increasing the pixel lightness of the first frame by a certain value and decreasing that of the subsequent frame by an identical amount (or in reverse). This results in two complementary frames that visually cancel out each other’s lightness changes thanks to the flicker fusion property of HVS. Yet in the meanwhile, they still can be captured by cameras with high capture rate, allowing extraction of the carried data bits.

Despite the intuitive philosophy, in practice however, various factors may cause flicker perception, such as frame rate, primary video contents, color space, etc. In the following, we present our novel design that enables fully imperceptible embedding, without degrading the transmission capability. Overall, we consider perceptual flicker regarding three factors: color space, pixel lightness, and texture complexity.

### 3.2 Color Space Selection

All existing works directly use common color spaces like YUV for lightness modifications, without examining the potential gains in using different color space [28, 35, 38, 42]. As colors are relevant to human psycho-perception (In fact, colors are not physical existence but merely subjective feelings of HVS [37]), exploring a better color space can greatly relieve the tension between unobtrusiveness and high-rate communication.

There are many color spaces designed with a lightness dimension, such as HSI, HSL, CIELAB, etc. In CHROMACODE, we target at a space that well suits human vision system, especially in lightness dimension. Such a color space should ensure *perceptual uniformity*, which means that under its lightness definition, increasing or decreasing the lightness

by  $\Delta L$  should result in about the same visual color differences and thus similar perceptual changes. Only by such condition can the eye-perceived average lightness of continuous changes from  $L + \Delta L$  to  $L - \Delta L$  be exactly equal to  $L$ , thus enabling flicker-free communication.

According to prior psycho-visual studies [19, 26], uniform color spaces are the most promising towards our goal because, by definition, they are more perceptually uniform, which means a change of the same amount in a color dimension produces a change of about the same visual importance. We experimentally validate such benefits of uniform color space. Fig. 3a and 3b depict the perceptual uniformity under HSL, a non-uniform color space, and CIELAB, a uniform color space respectively. In both cases, the original colors (middle of the each sector) are of the same saturation (100%) and lightness (50%), but are different in hue ranging from  $0^\circ$  to  $300^\circ$ . For each original color, we increase and decrease the lightness (outer and inner sector) by the same amount ( $\Delta L = 4\%$ ). As seen, given the same lightness changes, the resulted color differences are considerably more uniform in CIELAB space, regardless of the hue values. Similar results are observed in Fig. 3c and 3d where we keep hue ( $120^\circ$ ) and saturation (100%) unchanged and alter lightness dimension from 10% to 90%.

Therefore in CHROMACODE, we select CIELAB color space for lightness modification, which describes perceivable colors with three dimensions  $L^*$  for lightness,  $a^*$  for red/green color opponent and  $b^*$  for yellow/blue color opponent. CIELAB is a representative uniform color space with the latest and most accurate color difference formula CIEDE2000 applicable [10, 12]. It can be converted easily from commonly used spaces like the RGB space. Concretely, we first convert each pixel in the original color space (e.g., RGB or YUV) into the CIELAB color space, modify its lightness, and then convert it back to the original color space for display. This conversion process is necessary because the required changing values in RGB color space cannot be decided without firstly counting for the lightness change in CIELAB color

<sup>1</sup>Figure size and PDF compression may influence the perception. Refer to online examples for better viewing: <https://walleve.github.io/ChromaCode/>

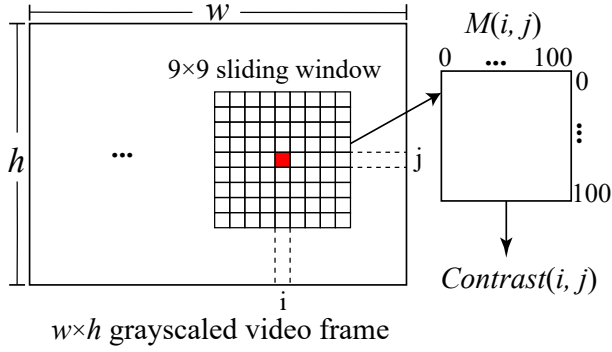


Figure 4:  $Contrast(i, j)$  calculation

space. Precision losses between color space conversions affect perception little and can be ignored.

### 3.3 Spatially Adaptive Embedding

Intuitively, we can simply apply a uniform lightness change value to a whole video frame as done in [35, 42]. However, noticeable flicker still frequently remains under such modifications due to two folds of reasons: First, depending on the specific original pixel lightness, an identical lightness change may produce different outcome color differences. Second, video contents upon which the data frame is embedded affect perceivable flicker. In particular, video regions with smooth texture are more sensitive to lightness changes. Thus applying a fixed lightness change value over a full video frame may lead to regional noticeable flicker.

These limitations motivate us to design an outcome-based spatially adaptive embedding scheme for lightness modifications, targeting at ultimate outcome of color difference and accounting for primary video contents.

**3.3.1 Outcome-based Lightness Change Derivation.** In contrast of lightness-change-driven principle adopted in previous works [28, 35, 42], we propose to design an outcome (i.e., color difference) based lightness modification scheme. Specifically, rather than applying a certain lightness change and anticipating certain color difference as output, we first specify the desired color difference and then calculate the required lightness changes for the color difference.

Denote  $\Delta E_{00}$  as the expected color difference. To gain the best uniformity in eye perception, we calculate the color difference  $\Delta E_{00}$  of a pair of color values in CIELAB space using the CIEDE2000 formula, the most accurate color difference formula currently available suggested by CIE [12]:

$$\Delta E_{00} = \left[ \left( \frac{\Delta L'}{k_L S_L} \right)^2 + \left( \frac{\Delta C'}{k_C S_C} \right)^2 + \left( \frac{\Delta H'}{k_H S_H} \right)^2 + \frac{R_T \Delta C' \Delta H'}{k_C S_C k_H S_H} \right]^{\frac{1}{2}}$$

where  $\Delta L'$ ,  $\Delta C'$  and  $\Delta H'$  are corresponding lightness, chroma and hue differences,  $S_L$ ,  $S_C$ ,  $S_H$  and  $k_L$ ,  $k_C$ ,  $k_H$  are respective weighting functions and parametric factors, and  $R_T$

is a rotation function. As we only modify  $L^*$  and keep  $a^*$  and  $b^*$  unchanged, both  $\Delta C'$  and  $\Delta H'$  here equal 0. The above formula can be simplified as:

$$\Delta E_{00} = \frac{\Delta L'}{k_L S_L}. \quad (1)$$

Then the lightness change  $\Delta L'$  required to achieve color difference  $\Delta E_{00}$  can be derived as  $\Delta L' = k_L S_L \Delta E_{00}$  with  $k_L = 1$  recommended by CIE [12] and

$$S_L = 1 + \frac{0.015(\bar{L}^* - 50)^2}{\sqrt{20 + (\bar{L}^* - 50)^2}}, \quad (2)$$

where  $\bar{L}^*$  denotes the average lightness value, which equals to the original lightness in CHROMACODE as we increase/decrease it by the same amount. Then for a specific pixel at position  $(i, j)$  of a video frame, the lightness change for a color difference of  $\Delta E_{00}$  is:

$$\Delta L_1^*(i, j) = k_L \left[ 1 + \frac{0.015(L^*(i, j) - 50)^2}{\sqrt{20 + (L^*(i, j) - 50)^2}} \right] \Delta E_{00}, \quad (3)$$

where  $L^*(i, j)$  is the original pixel lightness. As long as an appropriate color difference  $\Delta E_{00}$  is specified, we can have the required lightness change value  $\Delta L_1^*(i, j)$ .

Eqn. 3 indicates that the required lightness change value  $\Delta L_1^*$  for a specific color difference  $\Delta E_{00}$  depends on the original lightness  $L^*$  and that the same lightness change value based on different original lightness will output different color differences, demonstrating insufficiency of conventional lightness modification approaches without considering original pixel lightness. Previous researches [29] suggest that  $\Delta E_{00}$  should be below 6.0. We evaluate impact of  $\Delta E_{00}$  on flicker invisibility in §6.2 and give its recommended values.

**3.3.2 Texture-based Lightness Change Adaptation.** Prior research has demonstrated that video texture affects perceptual lightness changes [16]. In particular, as depicted in Fig. 3e, according to our experimental measurements, smooth regions usually expose noticeable flicker more easily while textured regions help hide flicker. Hence adjusting  $\Delta L_1^*(i, j)$  in Eqn. 3 by accounting for regional texture complexity will further relieve noticeable flicker especially in less textured regions. Yet different from TextureCode [35] that only selects some parts of a video frame for embedding and directly discards other regions, our method seeks for appropriate lightness changes over the whole frame, retaining its full capability for transmission.

We adopt gray level co-occurrence matrix [39], a typical method for image texture measures, for this purpose. For a co-occurrence matrix  $M$ , each element  $M(i, j)$  indicates the frequency within a set of image values (i.e., gray level) that value  $i$  co-occurs with value  $j$  in certain spatial relationship called displacement vector. Based on gray level

co-occurrence matrix, different metrics have been defined to reflect texture complexity more precisely, such as energy, entropy, contrast, etc [22]. Here we use and compute *contrast* as follows.

We apply a  $9 \times 9$  pixel sliding window to a video frame and generate regional gray level co-occurrence matrix, using  $L^*$  of pixels in this region to represent the gray level. As shown in Fig. 4, let  $Contrast(i, j)$  be the calculated contrast from the gray level co-occurrence matrix  $M(i, j)$  generated under a  $9 \times 9$  region centered at pixel  $(i, j)$ . Denote  $S(i, j)$  as the amount of valid pixels covered by the sliding window. For most circumstances  $S(i, j) = 9 \times 9 = 81$ , except when part of the sliding window moves out of the bound of a video frame so that  $S(i, j)$  will be smaller. Then we have the normalized texture complexity near pixel  $(i, j)$  as:

$$Texture(i, j) = \frac{Contrast(i, j)}{S(i, j)}. \quad (4)$$

A larger  $Texture(i, j)$  indicates more complicated texture near pixel  $(i, j)$ . Suppose the maximum  $Texture(i, j)$  over a whole video frame is  $TextureMax$ . We define a lightness scaling ratio  $\alpha$  as:

$$\alpha(i, j) = \frac{Texture(i, j)}{TextureMax} \times (1 - k) + k. \quad (5)$$

This maps  $Texture(i, j)$  to  $\alpha(i, j) \in [k, 1]$ , where  $k$  is the minimal scaling factor and can be set to a certain value. Then we refine the ultimate lightness modification amount  $\Delta L_2^*$  as:

$$\Delta L_2^*(i, j) = \Delta L_1^*(i, j) \times \alpha(i, j). \quad (6)$$

As seen,  $\Delta L_2^*$  is scaled down from  $\Delta L_1^*$  in smooth regions with smaller  $Texture(i, j)$ .

We then use  $\Delta L_2^*(i, j)$  as lightness modification amount for complementary pixels at position  $(i, j)$ . For one pair of pixels at position  $(i, j)$  in two complementary frames, we apply a respective lightness change of  $\pm \Delta L_2^*(i, j)$  to them to embed information bit '1' and apply  $\mp \Delta L_2^*(i, j)$  for bit '0'. By doing this, we successfully embed the data frames intended from streaming into the primary video frames, without impairing the viewing quality for display.

## 4 RELIABLE DATA TRANSMISSION

Data transmission is achieved by decoding the slight lightness changes of received video captured through the screen-to-camera channel. Besides the primary video contents, various factors may lead to channel errors, such as projection distortions, blurring, Moiré patterns, rolling shutter effects, etc. In this section, we present how to overcome these challenges and address the tension between high rate and reliable communication over the noisy screen-to-camera channel.

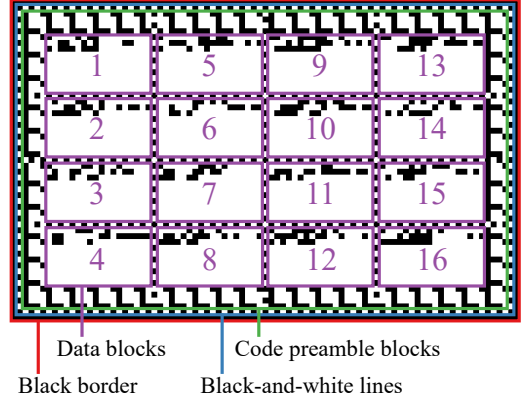


Figure 5: Data frame

### 4.1 Encoding and Modulation

**4.1.1 Data Segmentation.** A single data frame (image code as in Fig. 5) can hold only limited amount of data bits. Thus streaming data need to be segmented into different segments, each containing  $K$  bytes for one data frame. We design the segment header as simple as possible to reduce the amount of non-data bits. As shown in Fig. 6a, the segment header contains 3 bytes, the first two for segment sequence number and the last for checksum. The checksum is simply computed as the XOR sum of all data bytes and the two sequence number bytes.

Besides the three bytes, a payload length field is also needed to allow different code sizes. The payload length field is marked by a variable length of  $L = 1 \sim 4$  bytes, depending on segment length  $K$  (Fig. 6b). Here we use a variable length in purpose of remaining as more capacity for data bits as possible.

Supposing the capacity of a single data frame is  $C$ , we can derive the required  $L$  for the largest payload length and thus determine the largest group length  $K = C - L$ . The capacity is determined by data frame size. In this work, we define 40 different sizes of data frame, allowing for various requirements of code capacity. For each data frame size  $C_k$ , the data frame covers  $12k$  bits in width, and  $12k \times 9/16$  bits in height, where  $k \in [1, 40]$ .

**4.1.2 Concatenated Error Correction Coding.** To deal with random channel errors, a common error-correcting mechanism used in visible light communication is systematic code like BCH code, Reed-Solomon (RS) code, etc [42, 43]. Despite its stable error-correcting capability, RS code requires deterministic input for hard-decision decoding. Therefore, the performance may degrade due to channel distortions.

To deal with the noisy channel, we consider convolutional code commonly used in wireless communication, which performs maximum likelihood soft decision decoding. It doesn't require each bit's definite binary value, but instead only uses

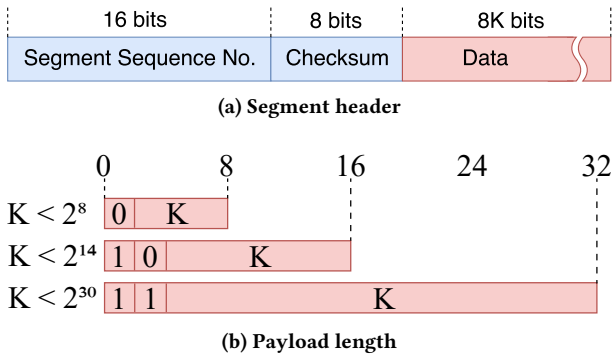


Figure 6: Data segment header structure

each bit’s distance to ‘0’ or ‘1’. This feature is especially beneficial to screen-camera channel where the received data (captured video) may be noisy due to channel distortions. The limitation, however, lies in its unstable error correcting capability related to input data bits, error position, etc.

Fortunately, we notice that RS code and convolutional code complement each other in advantages and drawbacks. To leverage the advantages of both codes and overcome each other’s shortcomings, we design a concatenated error correction code that employs RS code as outer code and convolutional code as inner code, which is commonly used in radio communications [20].

As shown in Fig. 7, to apply RS coding, the input data bits are divided into multiple units, whose length is determined by RS encoder’s parameter, i.e., length of data bits. For each unit, we apply RS coding (outer code) and add parity bits to the data. After that, we merge these units together and pass them through the convolutional encoder (inner code), resulting in encoded data bits. By concatenated encoding, the receiver can not only benefit soft decision decoding from convolutional code, but also enjoy high error-correcting capability of RS code to correct those data bits that are not correctly decoded by convolutional code.

Both the input/output ratios of RS code and convolutional code can be changed to achieve different error-correcting capability. In this work, we design 4 ratios for RS code and 2 for convolutional code, resulting in 8 different error-correcting levels for different scenarios. In this paper, we denote error correction level of convolutional code as  $(N, K, L)$ , and that of RS code as  $(N, K)$ , with  $N$  being output symbol rate,  $K$  being input data rate, and  $L$  being constraint length.

**4.1.3 Interleaving and Data Frame Assembling.** Finally the encoded data is rendered as an imagery code for embedding. Fig. 5 illustrates the overall structure of our imagery design for a data frame. It involves three types of basic elements, i.e., *cell*, the basic operation unit comprised by several pixels for one bit, *block* made from multiple neighboring cells, and

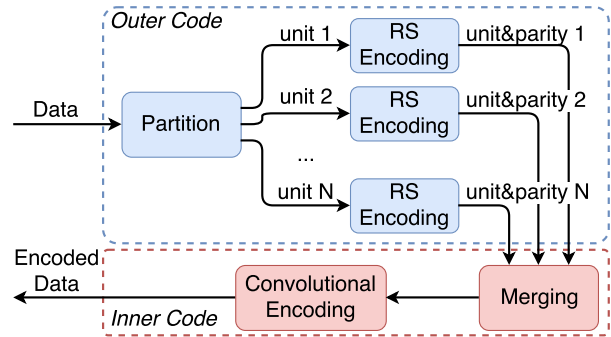


Figure 7: Concatenated Error Correction Code

*line*, a specially featured block formed by a column or row of cells.

(1) **Code Borders.** The outmost fringe of the data frame is comprised by a rectangle with black edges. Next to it is a rectangle consisting of four intermediate black-and-white featured lines with one cell width. Together with another 3 horizontal and 3 vertical such lines crossing the rectangle, they partition the whole frame into 16 blocks (i.e., data blocks). These featured code borders are intended for code detection and localization, as well as data block recognition. The specially designed black-and-white patterns are also preserved for handling rolling shutter effects (§4.2.1) and lightness normalization (§4.2.2).

(2) **Code Preamble Blocks.** Inside the black-and-white borders are filled by a round of repeated square blocks (4×4 cells) intended for code preamble information. Particularly, as shown in Fig. 8, code preamble information include 2 bits for segment sequence number (the last two bits of segment sequence number after segmentation, remained for handling rolling shutter effects as will be detailed below), 2 bits for RS code error correction level and 1 bit for convolutional code error correction level. We apply BCH coding to generate 10 parity bits for these 5 bit information, occupying 15 bits in total. The remaining 1 bit in the code preamble block is used as the checksum for the above-mentioned 15 bits. In addition to the strong error-correcting code, we further repeat the 16-cell block for many times in the data frame (Fig. 5) to ensure successful delivery of the code preamble information, which is critical to decoding at the receiver side.

(3) **Data Blocks.** All the remaining space except for the above two parts are used for rendering the encoded data bits. Since screen-camera channel may suffer from burst errors due to rolling shutter effects and primary video contents, which means some continuous areas encounter errors. Burst errors will degrade the error-correcting capability of both RS code and convolutional code. To alleviate it, we borrow interleaving technique from radio communication to disperse potential burst errors such that error correction code can deal with them as random errors. To apply interleaving, we partition the data space of a data frame into 16 equally sized

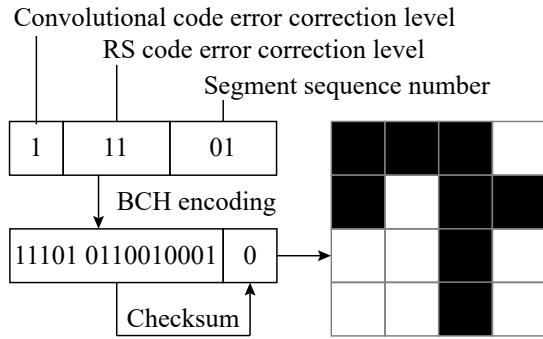


Figure 8: Code preamble blocks

blocks, each containing a certain number of cells determined by the specific code capacity. The blocks are filled with the encoded data bits in an iterative column-by-column manner, with black representing 1 and white representing 0. Specifically, we place the first 16 bits of the encoded data bits at the first left-top pixel of each data block respectively, in the order as indicated by Fig. 5. We repeat this procedure for all data bits, while during each iteration, we shift the pixel position to the next on the right (or the left first of the next row if current row is fulfilled). When all encoded data bits are rendered, an imagery code is produced as the data frame.

The data frame is then invisibly modulated onto the primary video frames for transmission by adaptive embedding, as stated above in Section 3. The resulting video is then displayed on the screen, delivering to users as normal video yet to cameras as embedded frames.

## 4.2 Demodulation and Decoding

Here we discuss data decoding and demodulation assuming that embedded video frames are available and code positions in the frames are detected, and postpone how to extract embedded frames from the captured video and detect code positions to next section.

**4.2.1 Demodulation.** When code positions are recognized, the data frame can be obtained by subtraction over two complementary video frames in lightness dimension. Note that CHROMACODE supports dynamic primary video contents. Even for dynamically changing video frames, we notice that the adjacent frames are mostly similar, allowing our subtraction mechanism to work. As a pair of complementary frames possess the same video content, only positive and negative lightness values remain after subtraction, which reflects the received data frame. Ideally, if the channel is perfect, this received data frame will be exactly the same with what was embedded at the sender side. However, significant biases exist as the captured video has undergone various channel distortions including imprecise code positions, rolling shutter effects and so on. In this subsection, we present how to

precisely recognize the code and deal with rolling shutter effects.

**Code Recognition.** The first step of demodulation is to recognize code capacity, which determines the size (width and height in pixels) of a cell. We also need to resolve the code positions as precisely as possible, especially when code boundary detection (§5) causes potential errors. We achieve precise code positioning and recognition jointly by looking into the four featured black-and-white border lines as shown in Fig. 5.

Recall that we have defined 40 ranks of code capacity, with a gap of 12 cells in width between adjacent ranks. Each rank is specified with a fixed aspect ratio of 16:9. Hence we can simply vote among the 40 candidate patterns by evaluating how each of them agrees with the captured patterns. For each of the four lines, denote  $g_1, g_2, \dots, g_N$  as the extracted lightness values of the  $N$  pixels it covers. Supposing the line should contain  $K$  cells under a specific code capacity, we can calculate a corresponding similarity  $S$  as:

$$S_K = \frac{1}{N} \left| \sum_{i=1}^N g_i(-1)^{\lfloor \frac{(i-1)K}{N} \rfloor} \right| \quad (7)$$

Exceptions are for left and right vertical lines, for which the lightness may be reversed due to rolling shutter effect. Suppose the rolling shutter happened at the  $j$ th pixel, then the true similarity  $S_K$  should be revised as:

$$S_K = \frac{1}{N} \left| \sum_{i=1}^{j-1} g_i(-1)^{\lfloor \frac{(i-1)K}{N} \rfloor} - \sum_{i=j}^N g_i(-1)^{\lfloor \frac{(i-1)K}{N} \rfloor} \right| \quad (8)$$

Since we do not know the true  $j$  at this moment (next subsection will present how to infer the exact  $j$ ), we choose  $j \in [1, N]$  that maximizes  $S_K$  in Eqn. 8.

We sum up the  $S_K$  from all four lines and compare the sums for every possible code capacity. The code capacity that yields the largest similarity is then selected. As adjacent capacity ranks are 12 cells apart, we can find the correct code capacity confidently.

Once we have the code capacity information, the cell size in pixels as well as the data block positions are also known. Then we compute the lightness of each cell as the averaged value over all pixels within the cell, which will be used for soft-decision decoding in §4.2.2 after eliminating the impacts of rolling shutter effect.

**Rolling Shutter Effect Correction.** When recording a picture or a video frame, modern cameras don't capture a snapshot of the entire scene instantly at one time, but instead scan the scene line-by-line horizontally, which is the well-known rolling shutter effect [33]. As a result, the pixels of different rows in one image are actually captured at different time instants. In CHROMACODE, cameras capture video frames at the same frame rate as video displays on



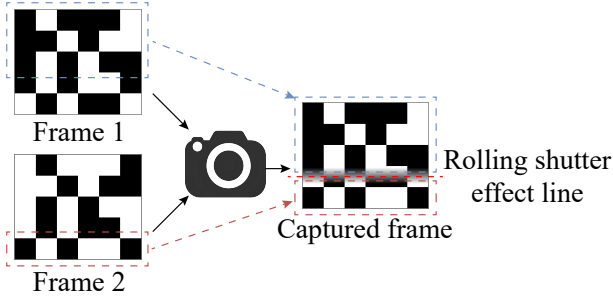


Figure 9: Rolling shutter effect

screen. Therefore, one frame captured by the camera may in fact come from two successive video frames played on the screen, with one dimmed horizontal line dividing them, as shown in Fig. 9. Upper the line is the previous frame while lower the line is the subsequent frame. This will lead to reversed lightness difference when taking the subtraction of two frames. In addition, the line itself is dimmed, leading to burst errors around it.

Albeit we have employed interleaving technique in the encoding scheme to alleviate unpredictable burst errors, we particularly correct rolling shutter effect before decoding. Recall Section 4.1.3, we have designed 5 specially featured vertical black-and-white lines in our code (See Fig. 5). We can identify where the rolling shutter exactly happened by examining from where the sequenced black and white cells are inverted. To detect that position, we can simply reuse the similarity formula in Eqn. 8, while we denote  $g_1, g_2, \dots, g_n$  as the average lightness of each cell and  $n$  as the number of cells contained in each line. The calculated similarity  $S_j$  will reach its maximum when  $j$  is the exact cell where rolling shutter happened, because for any other  $j$  there will always be some mismatches between the captured lightness and the predefined black and white sequence. Therefore, the position of rolling shutter is estimated as  $\hat{j} = \arg \max_j S_j$ . Each of the 5 vertical featured lines can output an estimate of the position. We then interpolate among the 5 estimates to derive the ultimate position, and revert the lightness change values below that position. By doing so, the impacts of rolling shutter effects are corrected.

**4.2.2 Decoding. Normalization.** After demodulation, we have the received data frame with corrected lightness for each cell. Ideally, each positive lightness value shall represent bit ‘1’ while negative represents bit ‘0’ (or vice versa). In practice, however, due to screen-camera channel distortions, there may be significant lightness offsets, making this simple strategy infeasible. To combat lightness distortions, we normalize the subtracted lightness values to  $[0, 1]$  and employ soft-decision decoding to leverage their tendency to bit ‘1’ or ‘0’.

Again we take the featured black-and-white lines as references for lightness normalization over the data frame. Particularly, we use the median lightness of all black and white blocks on these featured lines as reference values for bit 1 and 0, denoted as  $R_1$  and  $R_0$  respectively. Denote the lightness of the data cell centered at  $(i, j)$  as  $\Delta L(i, j)$ . Then we normalize  $\Delta L(i, j)$  to  $\Delta L'(i, j)$  by:

$$\Delta L'(i, j) = \begin{cases} 1 & v > 1 \\ v & 0 \leq v \leq 1 \\ 0 & v < 0 \end{cases} \quad (9)$$

$$v = \frac{\Delta L(i, j) - R_0}{R_1 - R_0} \quad (10)$$

By Eqn. 9, we map each cell’s lightness to a value within  $[0, 1]$ , which is sufficient for our soft-decision coding scheme, without the need to force them to binary values of 1 or 0.

**Convolutional Decoding and RS Decoding.** We read normalized lightness of each data cell in the same order as interleaving during encoding process, leading to a sequence of data values between  $[0, 1]$ . Then we identify the preamble blocks and extract code preambles including segment sequence number, RS code and convolutional code correction levels. With the data sequence and code preamble information, decoding basically works in two steps: First, we decode the inner convolutional code with Viterbi decoding algorithm [41], which, in brief, seeks for a Viterbi path of most likely sequence in terms of certain distance metric. This step takes advantages of its soft-decision decoding property, which calculates code distances from the normalized lightness values to 1 or 0 but not quantize them by a hard threshold of, e.g., 0.5. After that, we conduct RS decoding upon the decoded output of convolutional decoding, and finally obtain the original segmented data.

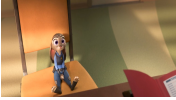




## 5 IMPLEMENTATIONS

We have developed data frame embedding program on computers as the sender, as well as Android application on smartphones as the receiver. To embed codes in normal video at sender, we use FFmpeg [1] tool to duplicate primary video frames to 120fps as well as compress the modulated video frames into the final ready-to-transmit video. We use MediaCodec[3] from Android API for hardware video decoding at receiver. We do image processing with Qt[6] at the sender and OpenCV[5] at the receiver.<sup>2</sup> In the following, we discuss some implementation details that address several practical challenges.

**Data Frame Localization.** Unlike InFrame++ [42] that uses visible locators and invisible alignment patterns around data frame borders for code block localization, which affects

<sup>2</sup>All our implementation and the following evaluation programs and data are available upon request.

**Table 1: Representative primary video clips and tags used in the experiments**

	Zootopia (Z) [8]	Football (F) [7]	GTA V (G) [7]	Minecraft (M) [7]	Town (T) [7]
					
<b>Texture</b>	Plain	Textured	Textured	Textured	Plain&Textured
<b>Switching</b>	Gradual	Gradual&Sharp	Sharp	Gradual	Gradual
<b>Luminance</b>	Bright	Bright	Dark	Bright	Dark
<b>Quality</b>	SD	SD	HD	HD	SD

video watching experience and shrinks precious code embedding spaces, we perform full-screen data frame embedding. The code borders are the same as the video borders, and also the same as the monitor inner border (we assume the common case that video is playing at full-screen mode). Thus we can perform code localization by detecting the screen border, without any extra visible locators or patterns. There are already mature techniques for screen border detection. In CHROMACODE, we combine Canny algorithm [15] and Hough transform [23] for this purpose.

**Handling Channel Distortions.** We also overcome several screen-to-camera channel distortions that may cause burst errors and noises for data transmission.

1) *Projection Distortions.* Geometric distortion is a unique challenge of screen-camera channel, which is due to the unparallel camera plane and screen plane, i.e., an angle exists between them. Therefore the boundary of captured video frames are not regular rectangles but distorted quadrilaterals. CHROMACODE addresses this issue by recovering the distorted frames via projection transformation under homogeneous coordinates [17].

2) *Moiré Pattern.* Images taken from an electronic display with a digital camera may exhibit Moiré patterns because both electronic screens and digital cameras display or capture images by line-by-line scanning [4]. Moiré patterns will blur the captured frames and act as noises to screen-camera channel, preventing precise code block recognition. Fortunately, as Moiré patterns usually occur at relatively high frequency [46], they can be eliminated by low-pass filtering. In CHROMACODE, we apply Gaussian smoothing to the received frames to attenuate high-frequency components, which is demonstrated to effectively overcome Moiré patterns.

## 6 EXPERIMENT EVALUATION

### 6.1 Experimental Methodology

Our evaluation contains two major parts, a user study that evaluates the user perceived data hiding quality and a data transmission benchmark that demonstrates the throughput and BER under various conditions. For both evaluation, we

implemented two related schemes, InFrame++ [42] and TextureCode [35], for comparison.

**Experiment Settings.** We use a DELL XPS 8900-R17N8 desktop as the sender for video generation and playing, which has a GeForce GTX 745 graphic card that supports hardware video acceleration for efficient 120fps video playing and an AOC AGON AG271QX 27 inch monitor that supports 120Hz refresh rate. The screen resolution is fixed to 1920×1080. By default the receiver is run as an App on a Nexus 6P smartphone. Note that the camera frame rate is not necessarily the same as the screen refresh rate. Any camera that supports a capture rate equal to or higher than the screen refresh rate can work as the receiver.

**Video Selection.** We select a group of videos as primary videos for experiments. A key consideration is to find a set of representative videos with various characteristics, e.g., incorporating both plain and textured frames, involving both sharply and slowly switching scenes, containing both dark and bright scenarios, and covering both high definition (HD) and standard definition (SD). Accordingly, we choose 10 different videos for experiments. Table 1 shows screenshots of 5 representative video clips and their characteristics. Resolutions of HD videos are all 1920×1080, while those of SD videos are relatively lower. Besides, CHROMACODE supports 4K video as long as sufficient 120 fps 4K video playing is supported by the graphic card.

**Comparative Approaches.** As we do not have access to the source codes of InFrame++ and TextureCode, we make our own implementations according to their paper. TextureCode uses YUV color space for lightness modification. InFrame++ authors did not clarify what color space they use. However, as their sender is mainly implemented in GPU, we speculate it uses YUV color space as well since GPUs commonly use YUV color space. TextureCode additionally faces a major issue that code positions are unknown to receiver, for which they simply assume that the receiver already has the knowledge from the sender. In our implementation, we follow the same settings for TextureCode.

**User Study.** We invited 20 participants with 8 females and 12 males ageing in range of 20 to 40 with healthy vision conditions to watch 67 experimental videos under various

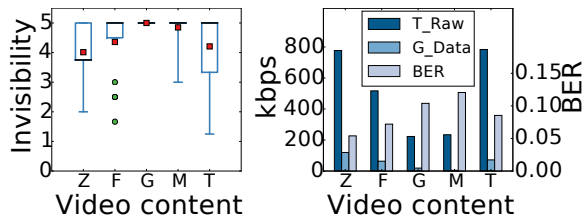


Figure 10: Impact of video sources

conditions and assess on the data hiding quality according to their watching experience. They are asked to score the perceived flicker level by 5 grades from 1 to 5, where 5 indicates completely imperceptible, 3 means slightly perceptible but not affecting much the watching experience, and 1 indicates severely affected watching experience. To reduce score bias among different users, we provide sample videos of score 1 to 5 for their references. Users are told to watch the videos normally as they usually do. They are not required to remain static during watching.

The experiment videos are displayed to users in a fixed, previously randomized order. To eliminate the interferences of the diversity of primary video qualities on the flicker level and avoid psychological implications, we mix primary videos with their embedded versions for user assessment and then calculate the relative score of each video as  $score = score\_embedded / score\_origin \times 5$ , where  $score\_origin$  and  $score\_embedded$  indicate the respective scores users gave on the video before and after embedding. If  $score > 5$  (which means  $score\_embedded > score\_origin$ ), we let  $score = 5$ .

**Data Transmission.** We conduct experiments to evaluate the throughput and BER and study the impacts of various factors and environment conditions. We use three metrics. 1) *Data goodput* (goodput or G\_Data hereafter): the effective throughput of correctly decoded data bits, which is the ultimate goal for data communication; 2) *Raw throughput* (throughput or T\_Raw hereafter): throughput accounting for all received bits, which is, albeit not achievable in practice due to errors, useful for indicating the upper bound of potential throughput; 3) *Bit error rate* (BER): the number of error bits divided by total received data amount.

## 6.2 CHROMACODE Performance

By default, we set  $\Delta E_{00} = 2.0$ ,  $k = 0.5$ , data cell size as  $10 \times 9$ , convolution code error correction level as (3, 1, 5), RS code error correction level as (30, 11), and test at typical watching/recording distance 50 cm. Under default settings, CHROMACODE achieves remarkable throughput of 777 kbps and goodput of 120 kbps, with a BER of 0.05. Note that the ultimate goodput is much smaller than the raw throughput. This is because the concatenated code occupies a significant portion of channel capacity for parity bits, which can be changed by using different error correction levels.

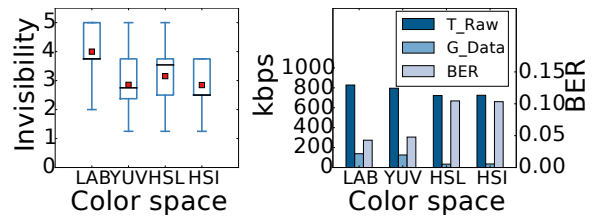


Figure 11: Impact of color space

In the following, we examine the impacts of some individual parameters by varying each of them while leaving others unchanged as the default values. Impacts of other parameters such as viewing angel, ambient luminance and energy efficiency are not shown due to paper space limitation. For all boxplots in Fig. 10-14 and Fig. 15, black lines are medians, red dots are means, and green dots are outliers.

**Video sources.** We first evaluate the results on different source videos in Fig. 10. Different videos exhibit considerably diverse performance due to their disparate properties in terms of texture, luminance, quality, etc (See Table 1). In short, videos achieving higher data hiding quality usually lead to low throughput and high BER. For example, both *GTA V* and *Minecraft* video clips, two HD videos with highly textured contents, yield the highest data hiding quality with mean score of nearly 5, but also the lowest throughput. The other two videos *Zootopia* and *Town* produce the contrary results. This is natural because when flicker is imperceptible to users, it also becomes difficult for cameras. We test with 5 additional highly textured videos [7]. The raw throughputs range from 312 to 621 kbps, goodputs range from 5 to 56 kbps, and BERs range from 0.09 to 0.12. These results prove that good performance remains over various videos with diverse properties. In the following, we choose *Zootopia* as the default video for parameter study. The results using other videos can be similarly inferred.

**Color space.** We compare bits embedding in LAB color space and three non-uniform color space, i.e., YUV, HSL, HSI, all with a lightness dimension. As shown in Fig. 11, thanks to the perceptual uniform color differences, CIELAB color space yields significantly better invisibility than the other color spaces, without downgrading the data transmission performance (the best data rate and BER are obtained by LAB color space).

**Color difference  $\Delta E_{00}$ .** Fig. 12 shows the impacts of  $\Delta E_{00}$ . As seen, as  $\Delta E_{00}$  increases from 1.5 to 3.5, both raw throughputs and goodputs increase, from 734 kbps to 846 kbps and from 94 kbps to 151 kbps respectively, while BER decreases from 0.07 to 0.03. The data hiding quality consistently remains at a graceful level (mean and median scores above 3) regarding different  $\Delta E_{00}$ , although it slightly drops to be observable in several cases when  $\Delta E_{00}$  is as large as 3.5. We

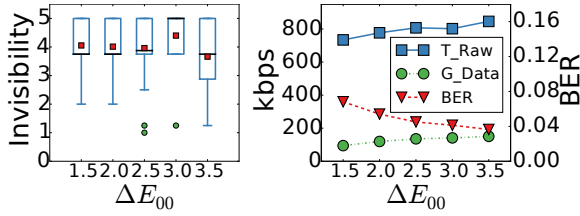


Figure 12: Impact of  $\Delta E_{00}$

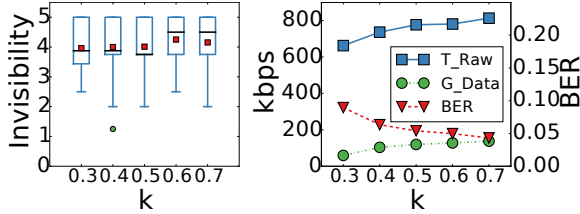


Figure 13: Impacts of  $k$

believe that  $\Delta E_{00} \in [2.0, 3.0]$  is a good compromise for both invisibility and transmission performance.

**Scaling factor  $k$ .** Fig. 13 presents the performance impacts of scaling factor  $k$  in Eqn. 5 for lightness adaptation. As expected, smaller  $k$  (which means smaller changes in smooth regions) results in less flicker, although the differences are marginal. However, transmission is more sensitive to  $k$ . As  $k$  decreases from 0.7 to 0.3, the BER increases significantly from 0.04 to 0.09, while goodput degrades from 138 kbps to 59 kbps. Considering the lightness changes even in smooth regions should be significant enough for capture, we suggest a safe range of  $k \in [0.4, 0.7]$ .

**Data cell size.** We examine the performance under different data cell sizes, which are changed at the sender side and automatically recognized by the receiver. As shown in Fig. 14, both throughputs and BER increase with smaller data cell size within one frame. This is intuitive since, smaller data cells mean more data blocks that convey more bits within a single frame while at the same time increase the probability of bit errors. Specifically, the overall goodput increases from 28 kbps to 137 kbps when cell size decreases from  $26 \times 17$  to  $8 \times 7$ , while suddenly drops to 58 kbps when data cell size further decreases to  $6 \times 6$ . The drop is due to too small data cells to be correctly recognized at the receiver.

**Watching and recording distance.** Fig. 15 illustrates the performance at various watching and recording distances. Evidently, the data hiding quality improves over larger watching distances. When watching from a distance of 180 cm, all participants give the highest invisibility scores for all videos. However, the flicker becomes challenging to be captured by cameras as well when it is too far to be seen by eyes. When recording distances increase from 60 cm to 180 cm, both throughputs and goodputs degrade from 697 kbps to 112 kbps and 56 kbps to 0.47 kbps respectively, while BERs increase from 0.09 to 0.12. Larger data cell sizes will improve

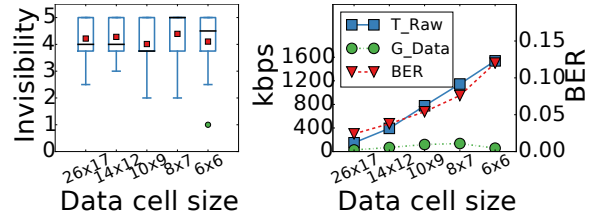


Figure 14: Impact of data cell size

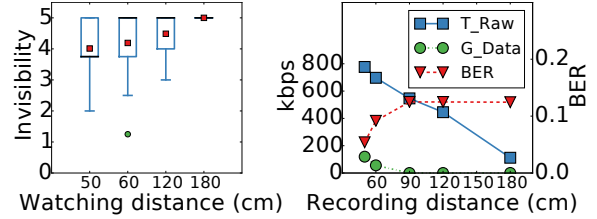


Figure 15: Impact of distance

performance over long distances. For example, a throughput of 309 kbps can be achieved at 180 cm when using a data cell of  $14 \times 12$ .

**Error correction level.** We evaluate the impacts of 4 out of our 8 different error correction levels and depict the results in Fig. 16, where we plot both the BERs after convolutional decoding (BER\_Conv) and after RS decoding (BER\_RS). We make two observations. First, concatenated code achieves lower BERs than any individual code scheme. Second, higher error correction levels of convolutional code or RS code lead to lower BERs, yet the latter's impacts are far more marginal than the former. However, higher error correction level also means lower data throughput, as parity bits occupy more space. As a trade-off, we by default recommend the error correction levels of convolutional code as (3, 1, 5) and RS code as (30, 11).

**Ambient luminance.** Fig. 17 shows the performance impacts of ambient luminance. We test under ambient luminances ranging from 0 to 300 lux, with the screen luminance fixed to 300 lux. As ambient luminance increases, data goodput slightly decreases and BER increases since more interference is resulted from higher ambient luminance. However, the impacts of ambient luminance on data rates and error rates are negligible.

**Recording angle.** We carry out experiments to test the impacts of different recording angles. The recording angle here refers to horizontal angle between the camera and perpendicular bisector of the screen. As shown in Fig. 18, the performance degrades with larger recording angles. Raw throughputs decrease to 531 kbps at 15 degrees and 365 kbps at 25 degrees. Data goodputs between 15 to 25 degrees are about 40 kbps and BERs are about 0.10. The performance degradation is not only owing to projection distortion, but also due to larger distances from the camera to the outermost side of screen due to the angle.

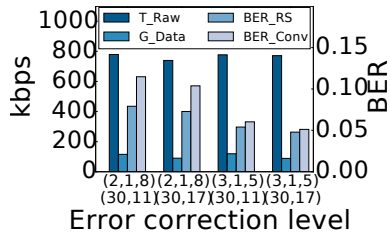


Figure 16: Impact of error correction level

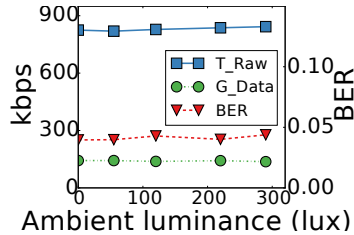


Figure 17: Impact of ambient luminance

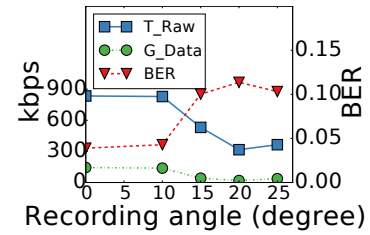


Figure 18: Impact of recording angle

Table 2: Energy consumption (W)

	Nexus 6P	Pixel 2
Video recording	0.05	4.11
Screen border detection	0.62	1.11
Demodulation and decoding	0.76	1.38

Table 3: Encoding/Decoding speed (fps)

	PC	Nexus 6P	Pixel 2
Encoding and Modulation	6.5	-	-
Screen border detection	-	3.8	8.0
Demodulation and decoding	-	0.6	2.2

**Hardware diversity.** We compare the performance on Pixel 2 smartphone. The raw throughput, goodput, and BER, under default settings, are 632 kbps, 55 kbps, and 0.09 respectively, which is slightly worse than Nexus 6P (recall the first paragraph in this Section). Such hardware diversity may arise from differences in camera quality and drivers that automatically control focusing and exposure time when recording video.

**Hand motions.** Hand motions during video recording will introduce extra distortions and potentially degrade performance. We test the performance in practical use cases when the receiver is held by user hand. The default performance achieved in raw throughput, data goodput, and BER are 627 kbps, 70 kbps and 0.09 respectively, which are slightly worse compared with fixed cameras.

**Energy consumption.** Table 2 shows energy consumption at the receiver implemented on Nexus 6P and Pixel 2 smartphones respectively. While CHROMACODE consumes more overall energy on Pixel 2 than Nexus 6P, we make two further observations: 1) Demodulation and decoding module consumes more energy than screen border detection module. 2) The power consumption for video recording varies violently on different devices, depending on the specific smartphone OS and camera hardware capability.

**Encoding/Decoding speed.** We evaluate the encoding and decoding speed of CHROMACODE. The sender is a DELL XPS 8900-R17N8 PC equipped with 8×3.40 GHz CPU, DDR4 RAM and 7200 rpm HDD. The receiver is run on Nexus 6P with 4×1.55 GHz & 4×2.0 GHz CPU as well as Pixel 2 with 4×2.35 GHz & 4×1.9 GHz CPU. By the time of writing

our implementation of CHROMACODE mainly uses CPUs for computing and has not performed GPU optimization. The experiment results are summarized in Table 3. At the receiver the demodulation and decoding speed is lower than that of screen border detection. Decoding speed on the more powerful Pixel 2 is faster than that on Nexus 6P. However, neither encoding nor decoding speeds have supported real-time capability now, which is remained for our future works.

Finally, we would like to point out that, despite of CHROMACODE’s significant improvements, the achieved BER is somewhat still high compared to typical radio or acoustic communications (typically under  $10^{-3}$ ). Screen-camera channel is a very noisy channel that may suffer from rolling shutter effect, Moiré pattern, etc. Moreover, in hidden screen-camera communication, we only make tiny modifications to videos to ensure invisibility. All these raise significant challenges in efficient decoding, which remains room for future enhancements.

### 6.3 Comparative Study

We compare the performance of CHROMACODE with two state-of-the-art approaches, namely, InFrame++ [42] and TextureCode [35]. Similar to CHROMACODE, both InFrame++ and TextureCode achieve different throughputs under different settings. For a fair comparison, we compare the invisibility and BER when they reach at equivalent throughputs. We adapt the data cell sizes of each system so that a specific throughput is obtained, and measure the corresponding invisibility scores and BER. Other parameters such as watching distances are kept identical for all systems.

The experimental results on 3 primary videos (Z, G, and T) are demonstrated in Fig. 19. As seen, although TextureCode retains graceful invisibility consistently, the throughputs it can achieve are considerably low, ranging from 1 kbps to only 95 kbps, yet the BERs are fairly high. This is because TextureCode does not embed in plain regions, and does not use error correction codes. As comparison, both InFrame++ and CHROMACODE yield throughputs up to 500 kbps, while CHROMACODE further reaches about 1360 kbps, nearly 3× over InFrame++. In addition, when producing equivalent throughputs from about 120 kbps to 540 kbps, CHROMACODE consistently outperforms InFrame++ by achieving significantly

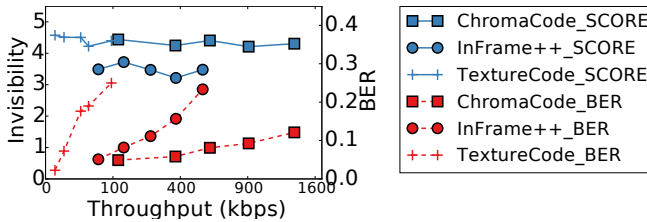


Figure 19: Comparative study

higher invisibility scores and lower BERs. The throughput gain over InFrame++ is mainly because that CHROMACODE embeds 1 bit using 1 cell, while InFrame++ uses multiple cells. The BER gain is obtained from CHROMACODE’s concatenated error correction coding and interleaving techniques. In a nutshell, CHROMACODE outperforms both InFrame++ and TextureCode with remarkably better flicker invisibility, higher throughputs and lower BERs.

We would like to point out that our evaluation results seem different from what was reported by TextureCode [35], which claims to reach equivalent or higher goodputs than InFrame++. This is because TextureCode uses a data frame rate of 60 fps for evaluation, while InFrame++ uses only 15 fps. In other words, in TextureCode one data frame is repeated within 2 video frames, while in InFrame++ it is repeated over 8 frames. In our comparison, we use 60 fps data frame rate for all.

## 7 RELATED WORKS

**Color Space.** Many color spaces have been defined and used in computer systems such as CIE1931 RGB and CIE1931 XYZ space [18]. These color spaces, however, are not perceptually uniform [31]. To overcome this, CIE later introduced uniform color spaces, e.g., CIE 1976  $L^*a^*b^*$  and CIE 1976  $L^*u^*v^*$  [10, 11]. However, the uniform color space is still not perfectly perceptual uniform. Decades of efforts are made by CIE to improve color difference calculation, from CIE76 to CMC(1:c) [30] to CIE94 [32] and finally CIE presented and recommended CIEDE2000, the most accurate color difference formula currently available [9], which is the exact formula used in CHROMACODE.

**VLC over Screen-Camera Links.** VLC grows as an increasingly hot topic [34, 40, 48]. Originated from one dimensional barcode, imagery codes like QR code [13] have been widely used for screen-camera communication nowadays. Recently, dynamic code becomes popular, which renders a sequence of images for streaming. PixNet [36] is the first-of-its-kind system that enables data streaming between LCD screens and cameras. COBRA [21] designs a novel colorful code for phone-to-phone communication. LightSync [24] enhances the coding scheme of COBRA and achieves higher rate. RDCODE [43] contributes a robust dynamic code design with multi-level error correction schemes. ARTcode

[47] aims to produce visual codes that encode data in the form of human-readable images. These works focus on efficient transmission over dedicated screen-camera links. On the contrary, CHROMACODE aims at invisible communication with screen-camera link as a side channel.

**Hidden Screen-Camera Communication.** Hidden communication conveys information via a side visual [42] or acoustic [45] channel. InFrame [44] and its extended version InFrame++ [42] pioneer the research area of unobtrusive screen-camera communication. They are the first to propose complementary frames to leverage flicker fusion property of HVS. However, as argued by a later work TextureCode [35], flicker still remains noticeable in InFrame++. TextureCode [35] improves invisibility by adaptive embedding based on video texture, but it only embeds data in textured regions and discards the others, which leads to degraded throughputs and changing uncertain code locations. Instead of lightness, HiLight [28] changes real-time accessible transparency in RGB color space and enables any-scene communication. Yet HiLight transmits bits by translucency changes at different frequencies, which largely limits its achievable throughputs. ImplicitCode [38] combines InFrame and HiLight to achieve better balance between invisibility and throughputs, which is, however, applicable to only grayscale carrier videos. Uber-in-Light [25] encodes data as complementary intensity changes over RGB channels and enables hidden communication for any screen and camera. Differently, CHROMACODE excels in its adaptive embedding in uniform color space, which is beyond the consideration of all existing works, robust coding scheme, and comprehensive system implementations.

## 8 CONCLUSION

We present CHROMACODE, a system that achieves all three goals on unobtrusive, high-rate, reliable screen-camera communication. We prototype CHROMACODE and evaluated it by experiments. The results demonstrate its superior performance over previous schemes in terms of invisibility, throughput and BER. CHROMACODE is on the way of changing video advertising industry by the new paradigm of advertisement placement for online video, TV, movie, outdoor electronic billboard, etc. Not only that, we anticipate even more imaginary spaces for novel applications.

## ACKNOWLEDGMENTS

We sincerely thank the anonymous shepherd and reviewers for their helpful comments and advices. We appreciate all 20 participants in our user study. This work is supported in part by the National Key Research Plan under grant No. 2016YFC0700100, NSFC under grant 61522110, 61332004, 61672319 and 61632008.

## REFERENCES

- [1] [n. d.]. FFmpeg. <https://ffmpeg.org/>
- [2] [n. d.]. The Interactive Effect: A Key to Surviving in the Attention Economy of a Mobile-First World. <https://medium.com/ipg-media-lab/the-interactive-effect-a-key-to-surviving-in-the-attention-economy-of-a-mobile-first-world-dcd8ace76ab1>
- [3] [n. d.]. MediaCodec | Android Developers. <https://developer.android.com/reference/android/media/MediaCodec>
- [4] [n. d.]. Moiré pattern. [https://en.wikipedia.org/wiki/Moiré\\_pattern](https://en.wikipedia.org/wiki/Moiré_pattern)
- [5] [n. d.]. OpenCV library. <https://opencv.org/>
- [6] [n. d.]. Qt | Cross-platform software development for embedded & desktop. <https://www.qt.io/>
- [7] [n. d.]. Xiph.org :: Derf's Test Media Collection. <https://media.xiph.org/video/derf/>
- [8] [n. d.]. Zootopia | Disney Movies. <http://movies.disney.com/zootopia>
- [9] 2002. Publications Briefly Mentioned: CIE 142-2001, improvement to industrial colour-difference evaluation. *Color Research & Application* 27, 1 (2002), 61–61. <https://doi.org/10.1002/col.10020>
- [10] 2008. Colorimetry – Part 4: CIE 1976  $L^*a^*b^*$  Colour space.
- [11] 2009. Colorimetry – Part 5: CIE 1976  $L^*u^*v^*$  Colour space and  $u', v'$  uniform chromaticity scale diagram.
- [12] 2013. Colorimetry – Part 6: CIEDE2000 Colour-Difference Formula.
- [13] 2016. Information technology - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification.
- [14] G. S. Brindley, J. J. Du Croz, and W. A. H. Rushton. 1966. The flicker fusion frequency of the blue-sensitive mechanism of colour vision. *The Journal of Physiology* 183, 2 (1966), 497–500. <https://doi.org/10.1113/jphysiol.1966.sp007879>
- [15] J. Canny. 1986. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8, 6 (1986), 679–698. <https://doi.org/10.1109/TPAMI.1986.4767851>
- [16] Charles Chubb, George Sperling, and Joshua A. Solomon. 1989. Texture interactions determine perceived contrast. In *Proceedings of the National Academy of Sciences of the United States of America (PNAS '89)*. 9631–9635.
- [17] H.S.M. Coxeter. 1969. *Introduction to geometry*. Wiley. <https://books.google.com/books?id=c0ld-crynsIC>
- [18] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. 1997. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research & Application* 22, 1 (1997), 11–23. [https://doi.org/10.1002/\(SICI\)1520-6378\(199702\)22:1<11::AID-COLA>3.0.CO;2-7](https://doi.org/10.1002/(SICI)1520-6378(199702)22:1<11::AID-COLA>3.0.CO;2-7)
- [19] D. J. Fleet and D. J. Heeger. 1997. Embedding invisible information in color images. In *Proceedings of International Conference on Image Processing (ICIP '97)*, Vol. 1. 532–535 vol.1. <https://doi.org/10.1109/ICIP.1997.647967>
- [20] Arunabha Ghosh, David R. Wolter, Jeffrey G. Andrews, and Runhua Chen. 2005. Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential. *IEEE Communications Magazine* 43 (2005), 129–136.
- [21] Tian Hao, Ruogu Zhou, and Guoliang Xing. 2012. COBRA: Color Barcode Streaming for Smartphone Systems. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*. ACM, New York, NY, USA, 85–98. <https://doi.org/10.1145/2307636.2307645>
- [22] Robert M. Haralick, K. Shanmugam, and Its'hak Dinstein. 1973. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3, 6 (1973), 610–621. <https://doi.org/10.1109/TSMC.1973.4309314>
- [23] P. V. C. Hough. 1959. Machine Analysis of Bubble Chamber Pictures. *Conf. Proc. C590914* (1959), 554–558.
- [24] Wenjun Hu, Hao Gu, and Qifan Pu. 2013. LightSync: Unsynchronized Visual Communication over Screen-camera Links. In *Proceedings of the 19th Annual International Conference on Mobile Computing and Networking (MobiCom '13)*. ACM, New York, NY, USA, 15–26. <https://doi.org/10.1145/2500423.2500437>
- [25] M. Izz, Z. Li, H. Liu, Y. Chen, and F. Li. 2016. Uber-in-light: Unobtrusive visible light communication leveraging complementary color channel. In *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM '16)*. 1–9. <https://doi.org/10.1109/INFOCOM.2016.7524513>
- [26] Rolf G. Kuehni. 2003. *Historical Development of Color Space and Color Difference Formulas*. John Wiley & Sons, Inc., 204–270. <https://doi.org/10.1002/0471432261.ch6>
- [27] IPG Media Lab. 2017. Media Trial Report: MAGNA and IPG Media Lab Turbocharge Skippable Pre-Roll Campaign.
- [28] Tianxing Li, Chuankai An, Xinran Xiao, Andrew T. Campbell, and Xia Zhou. 2015. Real-Time Screen-Camera Communication Behind Any Scene. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15)*. ACM, New York, NY, USA, 197–211. <https://doi.org/10.1145/2742647.2742667>
- [29] Hao Xue Liu, Bing Wu, Yu Liu, Min Huang, and Yan Fang Xu. 2013. A Discussion on Printing Color Difference Tolerance by CIEDE2000 Color Difference Formula. In *Advances in Printing and Packaging Technologies (Applied Mechanics and Materials)*, Vol. 262. Trans Tech Publications, 96–99. <https://doi.org/10.4028/www.scientific.net/AMM.262.96>
- [30] M R Luo and B Rigg. 1986. Uniform Colour Space Based on the CMC(l:c) Colour-difference Formula. *Journal of the Society of Dyers and Colourists* 102, 5-6 (1986), 164–171. <https://doi.org/10.1111/j.1478-4408.1986.tb01069.x>
- [31] David L. MacAdam. 1942. Visual Sensitivities to Color Differences in Daylight\*. *J. Opt. Soc. Am.* 32, 5 (1942), 247–274. <https://doi.org/10.1364/JOSA.32.000247>
- [32] R. McDonald and K J Smith. 1995. CIE94-a new colour-difference formula\*. *Journal of the Society of Dyers and Colourists* 111, 12 (1995), 376–379. <https://doi.org/10.1111/j.1478-4408.1995.tb01688.x>
- [33] Marci Meingast, Christopher Geyer, and Shankar Sastry. 2005. Geometric Models of Rolling-Shutter Cameras. *CoRR abs/cs/0503076* (2005). [arXiv:cs/0503076](http://arxiv.org/abs/cs/0503076) <http://arxiv.org/abs/cs/0503076>
- [34] S. Naribole, S. Chen, E. Heng, and E. Knightly. 2017. LiRa: A WLAN Architecture for Visible Light Communication with a Wi-Fi Uplink. In *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON '17)*. 1–9. <https://doi.org/10.1109/SAHCN.2017.7964932>
- [35] Viet Nguyen, Yaqin Tang, Ashwin Ashok, Marco Gruteser, Kristin Dana, Wenjun Hu, Eric Wengrowski, and Narayan Mandayam. 2016. High-rate flicker-free screen-camera communication with spatially adaptive embedding. In *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM '16)*. 1–9. <https://doi.org/10.1109/INFOCOM.2016.7524512>
- [36] Samuel David Perli, Nabeel Ahmed, and Dina Katabi. 2010. PixNet: LCD-camera Pairs As Communication Links. In *Proceedings of the Annual Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '10)*. ACM, New York, NY, USA, 451–452. <https://doi.org/10.1145/1851182.1851258>
- [37] Austin Roorda and David R. Williams. 1999. The arrangement of the three cone classes in the living human eye. *Nature* (1999), 520–522. <https://doi.org/10.1038/17383>
- [38] Shuyu Shi, Lin Chen, Wenjun Hu, and Marco Gruteser. 2015. Reading Between Lines: High-rate, Non-intrusive Visual Codes Within Regular Videos via ImplicitCode. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 157–168. <https://doi.org/10.1145/2750858>

- [39] George Stockman and Linda G. Shapiro. 2001. *Computer Vision* (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.
- [40] Zhao Tian, Kevin Wright, and Xia Zhou. 2016. The darkLight Rises: Visible Light Communication in the Dark. In *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*. ACM, New York, NY, USA, 2–15. <https://doi.org/10.1145/2973750.2973772>
- [41] ANDREW J. VITERBI. 2011. *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm*. Co-Published with Indian Institute of Science (IISc), Bangalore, India, 41–50. [https://doi.org/10.1142/9789814287517\\_0004](https://doi.org/10.1142/9789814287517_0004)
- [42] Anran Wang, Zhuoran Li, Chunyi Peng, Guobin Shen, Gan Fang, and Bing Zeng. 2015. InFrame++: Achieve Simultaneous Screen-Human Viewing and Hidden Screen-Camera Communication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '15)*. ACM, New York, NY, USA, 181–195. <https://doi.org/10.1145/2742647.2742652>
- [43] Anran Wang, Shuai Ma, Chunming Hu, Jinpeng Huai, Chunyi Peng, and Guobin Shen. 2014. Enhancing Reliability to Boost the Throughput over Screen-camera Links. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking (MobiCom '14)*. ACM, New York, NY, USA, 41–52. <https://doi.org/10.1145/2639108.2639135>
- [44] Anran Wang, Chunyi Peng, Ouyang Zhang, Guobin Shen, and Bing Zeng. 2014. InFrame: Multiflexing Full-Frame Visible Communication Channel for Humans and Devices. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets '14)*. ACM, New York, NY, USA, Article 23, 7 pages. <https://doi.org/10.1145/2670518.2673867>
- [45] Qian Wang, Kui Ren, Man Zhou, Tao Lei, Dimitrios Koutsonikolas, and Lu Su. 2016. Messages Behind the Sound: Real-time Hidden Acoustic Signal Capture with Smartphones. In *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking (MobiCom '16)*. ACM, New York, NY, USA, 29–41. <https://doi.org/10.1145/2973750.2973765>
- [46] Zhouping Wei, Jian Wang, Helen Nichol, Sheldon Wiebe, and Dean Chapman. 2012. A median-Gaussian filtering framework for Moiré pattern noise removal from X-ray microscopy image. *Micron* 43, 2 (2012), 170 – 176. <https://doi.org/10.1016/j.micron.2011.07.009>
- [47] Zhe Yang, Yuting Bao, Chuhao Luo, Xingya Zhao, Siyu Zhu, Chunyi Peng, Yunxin Liu, and Xinbing Wang. 2016. ARTcode: Preserve Art and Code in Any Image. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '16)*. ACM, New York, NY, USA, 904–915. <https://doi.org/10.1145/2971648.2971733>
- [48] Jialiang Zhang, Chi Zhang, Xinyu Zhang, and Suman Banerjee. 2016. Towards a Visible Light Network Architecture for Continuous Communication and Localization. In *Proceedings of the 3rd Workshop on Visible Light Communication Systems (VLCS '16)*. ACM, New York, NY, USA, 49–54. <https://doi.org/10.1145/2981548.2981556>