RollingLight: Enable Line-of-Sight Camera Communications

Hui-Yu Lee†, Hao-Min Lin†, Hsin-Mu Tsai†, and Kate Ching-Ju Lin‡
†Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
‡Research Center for Information Technology Innovation, Academia Sinica, Taiwan
{r01922028, d00922003, hsinmu}@csie.ntu.edu.tw, katelin@citi.sinica.edu.tw

ABSTRACT

The paper presents RollingLight, a visible light communication (VLC) system that utilizes a single LED light source as the transmitter and a CMOS rolling shutter camera as the receiver. In our system, we propose the use of Rolling Shutter Frequency Shift Keying (RS-FSK), a very simple modulation that can be implemented with very low cost embedded microprocessors, thus enabling LEDs that already exist in our daily life, such as LED indicators in electronics and light fixtures, to "talk". Today, almost every modern mobile device is equipped with at least one built-in CMOS rolling shutter camera, and it can receive signals from the LEDs without any modification, and thus, any additional cost. Our idea is to use the acquired images from the rolling shutter camera to extract the message transmitted from the LED. This technology can be used in many applications, such as advanced driver assistance system (ADAS), indoor positioning, advertising, and problem diagnosis. Our system allows rolling shutter cameras with different hardware specifications such as resolution, frame rate, read-out time, and exposure time, to all have the capability to decode the transmission, and the decoding performance scales with the camera specifications. In addition, the transmission is not visible to human eyes, thus keeping the original illumination or signaling function of the LEDs. Our design is evaluated with experiments performed with a range of different cameras. The RS-FSK modulation scheme can be used to transmit data at 11.25 bytes per second. We also develop a real-time application on the smartphone.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication; B.4.1 [**Data Communications Devices**]; C.3 [**Special-Purpose and Application-Based Systems**]

Keywords

Visible Light Communications, Camera Communications, Rolling Shutter, Frequency Shift Keying, smartphones

1. INTRODUCTION

Visible light communications (VLC) have gained wide attention in the research community recently. The possibilities of using every light emitting diode (LED) that already exists in our daily life to talk and to serve as a new pervasive communication infrastructure are endless - it enables interference-free, secure, low cost, and energy efficient short range communications to a wide range of electronic devices and appliances, from cars with LED taillights, light fixtures with LED light bulbs, to refrigerators with tiny LED indicators. VLC uses the optical signal to carry digital information by controlling the LED's light intensity in free space. When the light is transmitting, due to persistence of vision, human eyes cannot perceive the transmissions.

This work implements a form of VLC that uses a **single LED** as the transmitter and a Complementary Mental-Oxide-Semiconductor (CMOS) **rolling shutter camera sensor** as the receiver, a form of Camera Communications (CamCom). We chose to develop this form of VLC due to the universal availability of both the transmitting component - a simple LED (as opposed to an array of LEDs or an LCD screen), and the receiving component - a CMOS camera. Almost every modern mobile device, including smartphones, tablets, and laptops, is equipped with at least one built-in CMOS camera. We believe that this would be the best paradigm to jump-start the VLC technology in the market. As a result, one of the most important objectives of our work is to ensure the compatibility of a wide range of unmodified CMOS cameras to the LEDs transmission.

One very important advantage of CamCom is that the acquired images during the receiving process provide a means to **visually associate** the transmitting object with the transmissions, since the receiver has the knowledge of which pixels in the images are transmitting. Figure 1 shows a simple example, the user has the idea to know that every light is talking independently. We can use the idea in many applications. One is **Advanced Driver Assistance System (ADAS)**. we can use the LED tail lights of the front car to transmit a message to the camera of the rear cars. The mes-

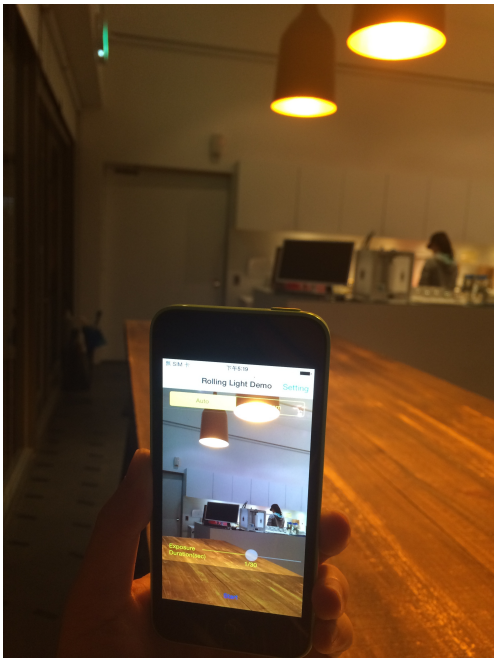


Figure 1: An example of utilizing CamCom for visual association.

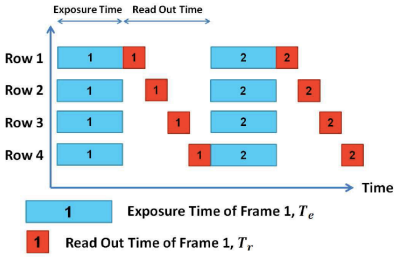
sage can contain safety-related status information, such as speed, location, heading, and brake/ turn signal status, or user-specified short text messages to form an advanced driver assistance system. Another example is **indoor positioning**. We receive the light ID transmitted by the LED light and obtain the vertical distance between the receiver and the transmitting lights, and the horizontal distances between the lights. Then, we can calculate the relative position and know where we are in the building. We also can use the LED lights for **advertising**. For example, in a supermarket, we can use the LED light on the shelf to transmit the sale and discount information of the goods. We can use the traffic lights and signs to tell us the road status ahead. Besides, we can apply this kind of communication for **system diagnosis**. For example, when a scooter breaks down, we can transmit the sensor log via the LED tail lights to briefly check and detect the problem of the scooter; in a server room, full of machines on the racks, it is often hard to find out which server is having trouble, the nature of the problem, and the IP address of the machine. Imagine if each server has VLC capability on one of its high brightness LED, then we can point our smartphone's camera at an array of servers, and immediately these information will be shown right next to the server, for the administrator to diagnose the problem in an easier way.

Challenge 1: sync (repetitive signal does need to deal with it because data is encoded in only a single image frame) NLOS: easy to fix sync, low signal sample loss prob., LOS: difficult: patterns are incontinuous so we have sync mechanism As the transmitting light and the receiving camera are not synchronized, the transmitting frame rate and the receiving frame rate are not usually the same. When the transmitting frame rate is higher than the receiving frame rate, there could be symbol loss. When the transmitting frame rate is lower than the receiving frame rate, reception of redundant symbols is possible. Although no information is lost, the receiver still needs to detect the redundant symbol so that it can be dropped to obtained the correct symbol sequence. A more common event when the transmitting frame duration and the receiving frame duration are different is that there could be multiple image areas each corresponding to a symbol. In other word, a mix of different symbols in a single received image frame. A mixed frame is a very common and periodic event. Even when the transmitting and the receiving frame durations are very close to each other, mixed frames would be received because of the phase difference between transmitter and receiver.Therefore, in this paper we also present a scheme that can handle **unsynchronized** transmitter and receiver pairs.

Challenge 2: compatibility + high data rate one-way communication, but we want a higher rate rolling shutter sampling (FSK) 1/tr is sampling rate, but different for different cameras Read-out-time estimation for compatibility among different cameras high data rate, two possibility (1) high symbol rate this is not possible, as a large portion of signal samples can be lost (2) high dimension modulation (high number of kinds of symbols) we need accurate and fast frequency estimation algorithm (YIN and variants, compared to FFT and general DIP results) (??) impact of exposure time: unfixable: we do measurement? In this work, we presented a **theoretical model** to describe the single-LED-to-rolling-shutter-camera channel. Furthermore, we proposed the **Rolling Shutter-Frequency Shift Keying (RS-FSK) modulation** and demonstrated that the modulation is compatible to a wide range of CMOS cameras with different resolution, read-out time, and exposure time, satisfies the lighting requirements, and can be implemented with cost-efficient hardware.

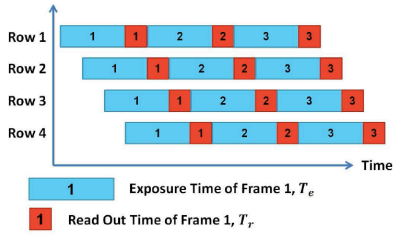
This paper proceeds as follows. Chapter 2 presents some prior research done in this area. Chapter 3 provides the background of the channel model of the rolling shutter camera, and RS-FSK modulation and demodulation techniques. Chapter 4 states the problem caused by unsynchronized transmitter and receiver pairs. Chapter 5 presents the hardware component and the software design of our system. Chapter 6 presents our experiment and evaluation results. Finally, chapter 7 gives the concluding remarks of this thesis.

2. PRIMER & MEASUREMENTS

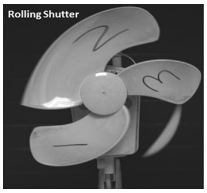


(a) Global Shutter Operation

(b) Global Shutter Example



(c) Rolling Shutter Operation



(d) Rolling Shutter Example

Figure 2: Shutter mechanisms

In this section we will describe how rolling shutter mechanism works and then introduce our frequency-shift keying modulation and demodulation scheme based on the rolling shutter camera. In addition, we will state the unsynchronized issue of transmitter and receiver.

2.1 Rolling Shutter Channel Model

2.1.1 *Rolling Shutter Operation*

Figure 2 compares global and rolling shutter operation [1]. Global shutters, which are commonly implemented on CCD sensors, expose all pixels on the sensor simultaneously and gather incoming light over all pixels during the exposure time. Although some CMOS sensors use a global shutter, the majority found in the consumer market utilize a rolling shutter. One of the key properties of a rolling shutter camera sensor is its sequential read-out architecture. As in most CMOS sensors there is no per pixel storage to hold the accumulated charge during the exposure operation, the exposure period happens right before each read-out operation. The read-out architecture can process charge voltage signals one row at a time. Since the read-out durations of two rows cannot overlap, each of the exposure durations of a row of pixels is shifted by a fixed amount of read-out time (represented by T_r in Figure 3), resulting in the so-called rolling shutter operation. For each pixel, the incoming intensity modulated signal is integrated for a time period called exposure time (represented by T_e in Figure 3).

Figure 3 shows the receiving operation of a rolling shutter camera sensor to an intensity modulated signal. Assume the original received signal in time is given by $r(t)$, which includes the signal from the transmitted

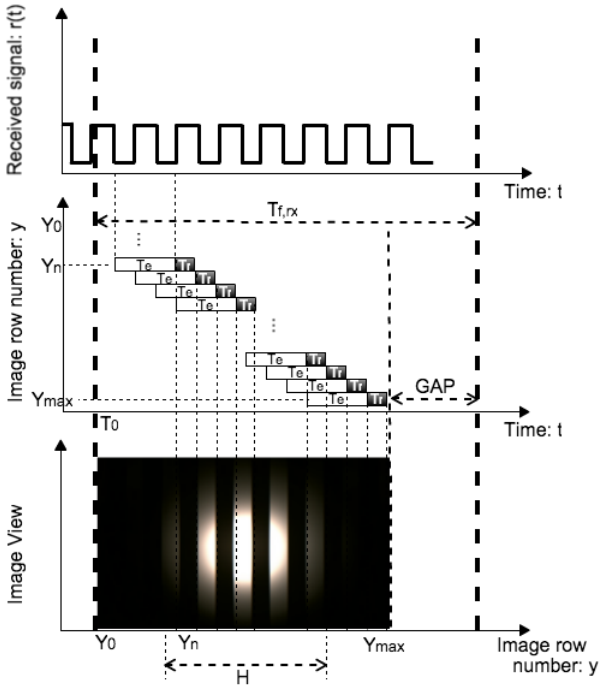


Figure 3: The receiving operation of a rolling shutter camera sensor. Top: the original received signal; middle: the exposure and read-out processes; bottom: the resulted image. $T_{f,rx}$: receiving frame duration; T_e: exposure time; T_r: read-out time.

light and other ambient interference. Considering the direct integration operation taking place during exposure and the rolling shutter operation, the intensity of a pixel in y-th row in the image, $I[y]$, which also represents the total amount of photons received by the pixel during exposure, is given by:

$$I[y] = \int_{T_0+yT_r}^{T_0+yT_r+T_e} r(t)dt,\ 0 \le y \le Y_{\max}, \qquad (1)$$

where T_r is the read-out time, T_e is the exposure time, T_0 is a reference time point that corresponds to the start of the exposure time of the first row of pixels in the image, and $Y_{\max}$ is the number of rows of pixels in the image. Note that the transmitting light might not illuminate all rows in the image. In that case, the transmitted signal is only observed in $I[Y_{\text{top}} \le y \le Y_{\text{top}} + H]$, where H represents the height of the image area illuminated by the transmitting light.

The direct integration operation can be considered as a low-pass filter (or a moving average filter), and thus $I[y], 0 \le y \le Y_{\max}$ is a filtered version of the original received signal $r(t)$. The filter distorts high frequency components in the received signal, especially the ones with a frequency higher than $1/T_e$.

2.1.2 *Time Gap in the Frame Duration*

Utilizing Equation 1, if the transmitting light occupies all rows of pixels in the image, the end of exposure of the last row in this image frame is $T_0 + Y_{\max} T_r + T_e$, while the start of exposure of the first row in the next image frame is $T_0 + T_{f,rx}$. The **time gap** between these two events is $T_{f,rx} - Y_{\max} T_r - T_e$, when T_e is small, is significant for most cameras. This is the amount of time during which the camera is not performing any exposure operation, i.e., not receiving the transmitted signal. Signal transmitted during this time period is lost.

Table 1 summarizes both the values of this time gap (without subtracting the exposure time T_e) and the percentage of time contributed by this time gap compared to a frame duration. Note that for some cameras, this could be up to half of the receiving frame duration. In addition, in the case that the transmitting light does not illuminate all rows in the image, this time gap could further increase.

Table 1: Summary of Camera Parameters

	Image Resolution ($X_{\max}$ x $Y_{\max}$)	Frame Rate (fps)	Measured Read-out Time (μs)	Time Gap (ms) (Percentage of Frame Duration)
Point Grey Flea3	2048x1080	30	14.73	17.42 (52.27%)
Apple iPhone 6 Plus	1920x1080	30	21.42	10.20 (30.60%)
Apple iPhone 5s	1920x1080	29.98	20.65	11.03 (33.10%)
HTC New One	1920x1080	29.94	19.08	12.79 (38.30%)
Samsung Galaxy S4	1920x1080	29.93	25.53	5.84 (17.48%)

2.1.3 *Channel Characteristics*

In summary, the single-LED-to-rolling-shutter-camera channel exhibits two major characteristics:

1. The signal that can be obtained from the image is a low-pass filtered version of the original received signal in time. The filter create distortions in high frequency components.

2. The receiving process is not continuous. This can be considered as a form of channel fading; when it takes place, the channel loss is infinite. The ratio of signal lost is determined by a constant time gap that depends on camera parameters, and the size of the image area illuminated by the transmitting light. A smaller image area corresponds to a higher ratio of signal lost.

2.2 Rolling Shutter Frequency-Shift Keying (RS-FSK) Modulation

The fundamental idea of RS-FSK is to use the square waves of a number of different **frequencies** as different symbols, mapped to different bit patterns of the same length. The signal of the i-th symbol is represented by the fundamental frequency f_i of the square wave:

$$s_i(t) = I_{\max} \left\lceil \frac{\cos(2\pi f_i t)}{2} \right\rceil \qquad (2)$$

where $I_{\max}$ is the maximum output intensity of the transmitting light.

A frequency modulation is advantageous in the following aspects: (1) The transmitted frequency can be demodulated by receivers with different sampling rate, as long as the sampling rate satisfies the Nyquist rate requirement. This implies that cameras with different read-out time are all compatible to a single transmission signal format. (2) Demodulation is still possible when some segments of signal samples are lost in a symbol period, as long as the longest remaining signal segment has a length larger than the transmitted signal period. This addresses the time gap challenge previously described. (3) The average intensity stays the same for all symbols, implying that the transmission is not observable by human eyes as long as the frequency is higher than 100 Hz.

In addition, a square wave has some key properties which make it suitable for our purposes: (1) It survives the moving average filtering mechanism due to the exposure operation, as long as the signal period is not an integer multiple of the exposure time. This is true even for very long exposure time. Most important of all, the fundamental frequency of the square wave remains the same and accurately observable after filtering. (2) It is extremely simple and can be implemented with cheap microcontrollers or PWM driver chips. No DAC is needed to output signal at different amplitude levels. (3) The modulation supports dimming by transmitting a modified square wave at the same frequency, but with different ratio of ON state and OFF state, i.e., duty cycle.

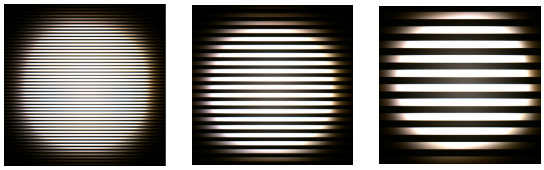
Figure 4 shows the received images of different transmitting frequencies.

2.3 Demodulation

The transmitted signal period is the inverse of the transmitted signal frequency f_i. To obtain the **signal period**, we need to know the pixel width of strip in the received image first. As rows of pixels are sequentially obtained with sampling period T_r, one cycle of square wave transmission results in a pair of bright strip and dark strip in the image. The sum of whole widths is given by

$$W = \frac{1}{f_i T_r} \quad . \qquad (3)$$

Note that the width of the bright strip could be larger than the width of the dark strip in the image. This is due to blooming, the overflow of charge from a saturated pixel into its neighboring pixel [2]. We thus consider the widths of the bright strip and the dark strip together in the calculation, to average out the effect. Since W might not be an integer, averaging the observed widths



(a) 8000 Hz (b) 4000 Hz (c) 2000 Hz

Figure 4: Received images of different transmitting frequencies.

over a large image area would help to improve the accuracy of the estimate of the width. Then, the transmitted period $1/f_i$ can be calculated with Equation 3.

To have a more robust method to accurately determine the period of the signal, i.e., the average width of a pair of bright and dark strips in the image, we modified a well-known pitch detection algorithm (PDA), YIN [3], that is originally designed to determine the fundamental frequency of a segment of audio signal in speech or music. We chose to use a method that operates in time domain, so that computationally expensive Fourier Transform operation can be avoided.

We start by summing up all pixels in each row

$$I[y] = \sum_{x=1}^{X_{\max}} I[x][y] \quad , \tag{4}$$

obtaining a one-dimensional signal, where $I[x][y]$ is the intensity (luminance)[1] of the pixel at location (x, y) in the received image. This operation uses pixels in different columns for redundancy, averaging out noises that exists in different columns of pixel.

To find the period of the periodic signal presented in $I[y]$, a difference function may be constructed:

$$d_\Delta(\delta) = \sum_{y=1}^{H/2-1} (I[y] - I[y+\delta])^2 \tag{5}$$

and we search for the values of δ that are closest to zero. Note that δ represents both the shift in space in number of pixels and the shift in time in multiples of read-out time of the camera. As pointed out in [3], using the difference function instead of the standard autocorrelation function (ACF) can avoid a large portion of the error caused by change of amplitudes in the signal, causing by the change of luminance in different rows of pixels. An added advantage is that subtraction is computationally more efficient than multiplication.

Another possible source of error happens at small δ values, since when $\delta < \frac{1}{fT_r}$, the difference function could produce a large value due to the fact that we use square waves - there is no significant change of signal amplitude except at the sharp transitions. We instead use cumulative mean normalized difference function (CMNDF) [3] to mitigate this problem:

$$d'_\Delta(\delta) = \begin{cases} 1, & \text{if } \delta = \{0, 1\} \\ d_\Delta(\delta) / \left[(1/\delta) \sum_{j=1}^{\delta} d_\Delta(j) \right] & \text{otherwise.} \end{cases} \tag{6}$$

The function now starts from 1 at $\delta = 0$ and remains large with small δ values, and only drops below 1 when $d'_\Delta(\delta)$ falls below average. The smallest local minimum in $d'_\Delta(0 \leq \delta \leq H/2 - 1)$ is then located and serves as the period estimate of the received signal.

Using CMNDF, the system can determine the period of the transmitted signal with a resolution of read-out time. However, as the period of the signal is not always an integer multiple of the read-out time, i.e., the sampling period, the output could results in error up to half of the read-out time. We again resort to the method proposed in [3] - to use parabolic interpolation to estimate the location of the actual minimum that could exist between samples. Assuming that $\hat{\delta}$ is the integer value that corresponds to the smallest minimum in $d'_\Delta(\delta)$, this method only needs three function values, $y_{-1} = d'_\Delta(\hat{\delta} - 1)$, $y_0 = d'_\Delta(\hat{\delta})$, and $y_{+1} = d'_\Delta(\hat{\delta} + 1)$, to determine the new estimate:

$$\hat{\delta}' = \hat{\delta} + \frac{y_{+1} - y_{-1}}{2(2y_0 - y_{+1} - y_{-1})} \tag{7}$$

With parabolic interpolation, our modified YIN algorithm can very accurately determine the period of the transmitted signal. Note that the output of the algorithm is in pixel. It can be converted back to be in unit of time by multiplying the number with T_r.

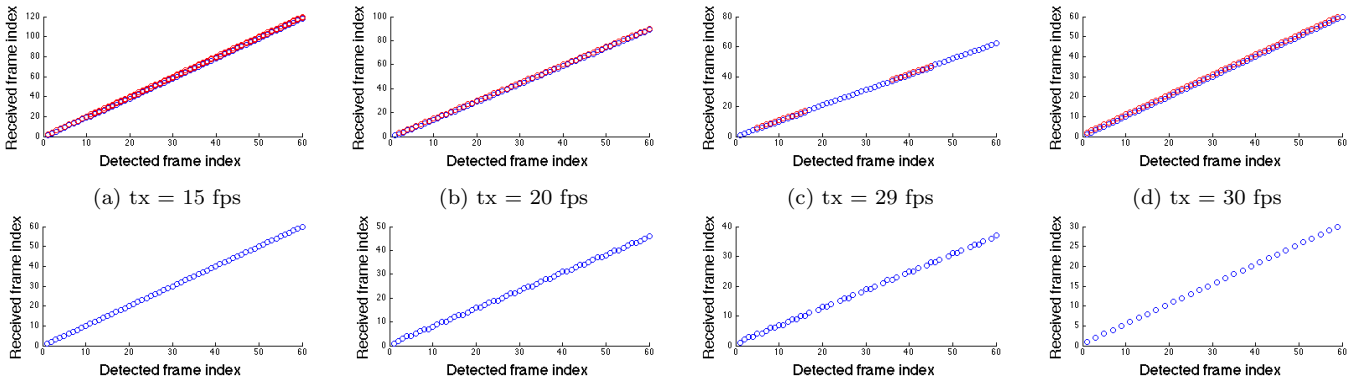
2.4 Unsynchronized Transmitter and Receiver

2.4.1 *The Problem*

As the transmitting light and the receiving camera are not synchronized, the transmitting frame rate and the receiving frame rate are usually not the same. As mentioned is [4], the received frame rate exhibits more variability for several reasons. It described as follows: HTC One X camera appears to record videos are 24 fps, but the camera callback API can only support saving the frames at 15 to 20 fps. The callback slows down when there is insufficient memory available in the system. The frame rate on the Samsung Galaxy S III is only stable if the CPU is locked to its maximum frequency with its setCPU app. Otherwise, it fluctuates hugely between 21 to 29 fps for an entirely white foreground, and have an average of around 25 fps. Even when the frame rate appears steady, the inter-frame interval still varies.

Received frame patterns. We follow the experiement in [4], given various potential combinations of the transmitting and receiving frame rates, we perform a simple experiment to study the received frame pattern. We

[1] If the obtained image is in RGB instead of grayscale, it needs to be converted to obtain the luminance information.



(a) tx = 15 fps (b) tx = 20 fps (c) tx = 29 fps (d) tx = 30 fps

(e) tx = 30 fps (f) tx = 40 fps (g) tx = 50 fps (h) tx = 60 fps

Figure 5: Received frame patterns under different tx fps

transmit the data at several frame rates, and record the video with the PointGrey Flea3 camera [5] with 30 fps.

Figure 5 shows the received frame pattern. For each received frame, we can detect which of the originally transmitted frames corresponds to it, and plot a circle for each detected frame. When two or more detected frame indices correspond to the same received frame index, this captured frame contains a **mixture of two or more transmitted symbols**. When two or more received frame indices correspond to the same detected frame index, these captured frames have **redundant transmitted symbols**. Any gap between consecutive received frame indices indicates a **symbol loss**.

At 15 fps and 20 fps, we capture at least one complete frame with a full symbol. However, other frames show that the mixed frames and redundant frames are also possible due to random phase offsets. When the transmitting rate is close to the receiving rate - 30 fps, mixed frames would be received for a few consecutive frames. Even when the transmitting frame rate is exactly 30 fps, mixed frames can still occur, in which case, it would always be the case received due to the constant phase offset, as shown in Figure 5(d). In the other case, a single transmitted symbol would always be received in each frame, as shown in Figure 5(e). As the transmitting rate increases further, we start to experience mixed and missed symbols at times.

2.4.2 *The Probability*

To figure out the symbol loss and mixed frame problems, we first want to know how often do they happen, thus we derive the probabilities under different conditions. In addition, we want to know what factors affect them and how to address them.

Case I: When the transmitting frame rate is higher than the receiving frame rate, there could be symbol loss, defined as no part of the transmitting frame duration of a particular symbol is covered by the exposure time of any receiving image frame. Figure 6 illustrates the missing symbols. Let the transmitting frame duration be $T_{f,tx}$ (symbol time), the receiving frame duration be $T_{f,rx}$, and the Y size of the image area illuminated by the transmitting light be H. The gray part is the time gap that the camera is not receiving the transmitting symbols. If $T_{f,tx} < T_{f,rx} - HT_r$, which means the transmitting symbols may appear in the gray part, then a symbol loss is possible. In that case, the probability of a symbol lost is given by

$$P_{\text{miss}} = \frac{T_{f,rx} - HT_r - T_{f,tx}}{T_{f,rx}} \quad . \tag{8}$$

We can see that the probability of the symbol loss in Figure 6 is around 1/2, which means there is a symbol loss followed by every received symbol.

A more common event which could happen when the transmitting frame duration and the receiving frame duration are different is that in a single image frame there could be multiple image areas each corresponding to a symbol. In other words, a mix of different symbols in a single receiving image frame. Figure 7 illustrates the mixed frames. If the boundary between two consecutive transmitting symbols locates in the red part, during which the camera is receiving the transmitting symbols, then there is a mixed frame. The probability for this event is given by

$$P_{\text{mix}} = \begin{cases} \frac{HT_r}{T_{f,tx}}, & \text{if } T_{f,tx} \geq HT_r \\ 1, & \text{otherwise.} \end{cases} \tag{9}$$

We can see that the probability of the mixed frame in Figure 7 is around 1/3.

A mixed frame is a very common and periodic event. Even when the transmitting and the receiving frame durations are very close to each other, mixed frames would be received for a few consecutive frames. After receiving a few frames with only a single symbol, consecutive mixed frame would appear again. Figure 9 shows the received frame with mixed symbols. If the boundary of the image areas corresponding to different symbols

Figure 6: Missing symbol illustration.

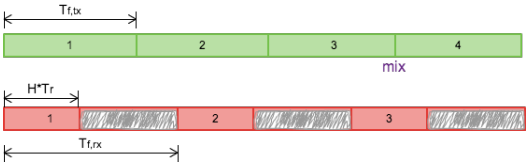
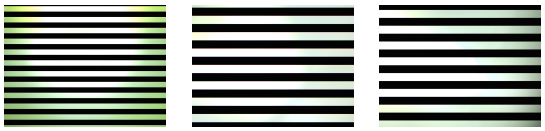


Figure 7: Mixed frame illustration.



(a) iPhone 5c (b) iPhone 5s (c) iPhone 6Plus

Figure 8: Received images of different devices with same transmitting frequency

is not determined, then the period detection algorithm would output one value. The value would usually be closer to the signal period of the symbol that occupies a larger image area, but usually has a large error.

Case II: When the transmitting frame rate is lower than the receiving frame rate, a redundant symbol is possible. Although no information is lost, the receiver still needs to detect a redundant symbol so that it can be dropped to obtained the correct symbol sequence.

On the other hand, a mixed frame is also possible in this case. The probability for a mixed frame is given by

$$P_{\text{mix}} = \frac{HT_r}{T_{f,tx}} \quad . \tag{10}$$

which is same as the probability in Case I. We can see that the probability of the mixed frame in Figure 10 is around 1/3.

According to the probability of lost symbol and mixed frame, the transmitting frame duration ($T_{f,tx}$), the receiving frame duration ($T_{f,rx}$), the height of the LED in the image (H), and the read-out time of camera sensor (T_r) and the factors which may affect the probabilities. We will introduce and evaluate how well our design address these issues in the following chapters.

3. ROLLINGLIGHT DESIGN

3.1 Overview

Figure 11 illustrates the system architecture of our proposed system. Data bits to be transmitted will first

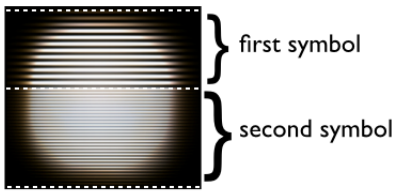


Figure 9: Received frame with mixed symbols.

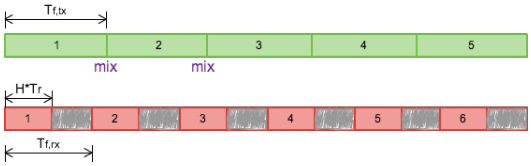


Figure 10: Mixed frame illustration.

be split into bit patterns of a constant length. Each bit pattern is then mapped to a data symbol. Besides data symbols, we also add parity symbol, sequence number, preamble symbol, and symbol delimiter. The transmitting light is then driven to transmit a sequence of square waves according to pre-determined list of signal periods representing the entire data packet.

The receiving camera captures a series of images with the transmitting light. The receiver starts the demodulation process by determining the image area occupied by the transmitting light. A symbol delimiter detector is then executed to determine whether a symbol delimiter area presents in the image. If The estimated signal period will be mapped to a meta symbol. Then, the YIN algorithm can be executed to determine the signal periods of two image areas separated by the symbol delimiter. In this case, the signal periods of both symbols can be accurately estimated. The meta symbol is then split into a sequence number and a data symbol. The data string is finally obtained after mapping the sequence of data symbols back to the bit patterns, forming the original bit stream.

3.2 Parity Symbol and Sequence Number

To deal with the unsynchronized problem, we add additional schemes. The sequence number is used to identify any missing symbol or redundant symbol. Missing symbols will be reconstructed with corresponding parity symbols, while redundant symbols with the same sequence number will be dropped.

As symbols could be lost or corrupted due to the time gap of the channel or if the exposure time is an integer multiple of the signal period, a parity symbol is inserted for every n data symbols by **XOR-ing** all n data symbols, so that if only one of the n data symbols or the parity symbol is not correctly received, the receiver can **recover the lost symbol**. The number of parity symbols that should be added to a packet can

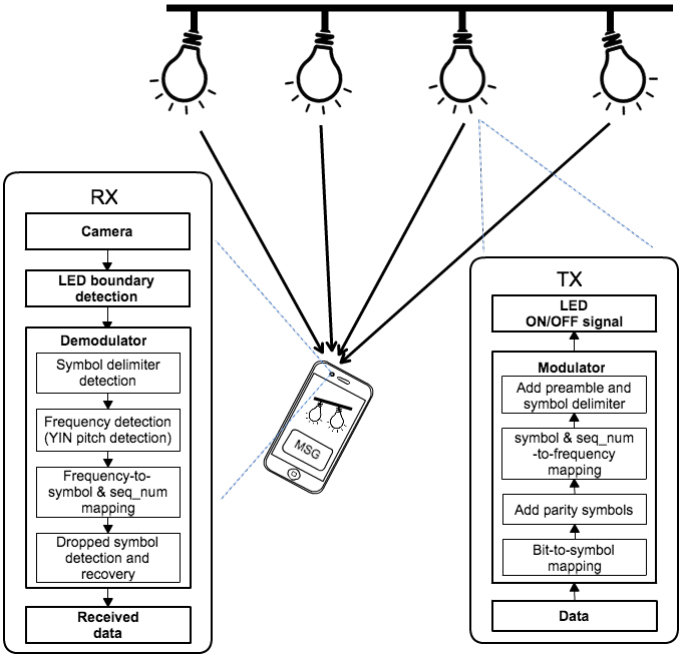


Figure 11: System Architecture of RollingLight

be determined by calculating expected number of lost symbols in a packet using the probability equation in Section 2.4.2.

After constructing a sequence of symbols, including the parity symbol, to be transmitted, we label each symbol with a **1.5 bit** sequence number $= \{0, 1, 2\}$. The sequence number can be obtained by calculating the index number modulo 3. The sequence number is then combined with the data symbol to become one meta symbol, then mapped to one of the selected signal periods. Note that the sequence number reduces the number of bits that can be represented by each symbol by 1.5 bits. However, this is required to detect **lost symbols** (when the transmitting frame duration is smaller than the receiving frame duration) or **redundant symbols** (when the transmitting frame duration is larger than the receiving frame duration). Taking the assumption into consideration, it can be proved that there could be no consecutive symbol losses. Thus, a 1.5 bit sequence number, $\{0, 1, 2\}$, would be sufficient determine the location of a lost symbol in the sequence.

3.3 Preamble Symbol

The preamble symbol, which has a signal period known to the receiver, is inserted at the beginning of the symbol sequence. The preamble symbol serves two functions: one is for the receiver to detect the start of the packet and start the subsequent demodulation process, while the other is for the receiver to accurately **calibrate its T_r value** based on the period estimate of the preamble, so that the error of the period estimates of all subsequent symbols in this packet can be minimized. As a result, the signal period with the smallest error variation is selected as the period for the preamble symbol.

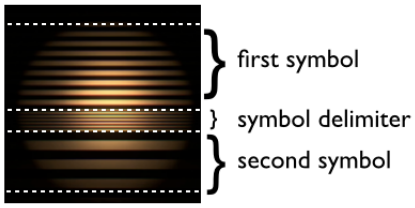


Figure 12: Symbol delimiter illustration.

3.4 Symbol Delimiter

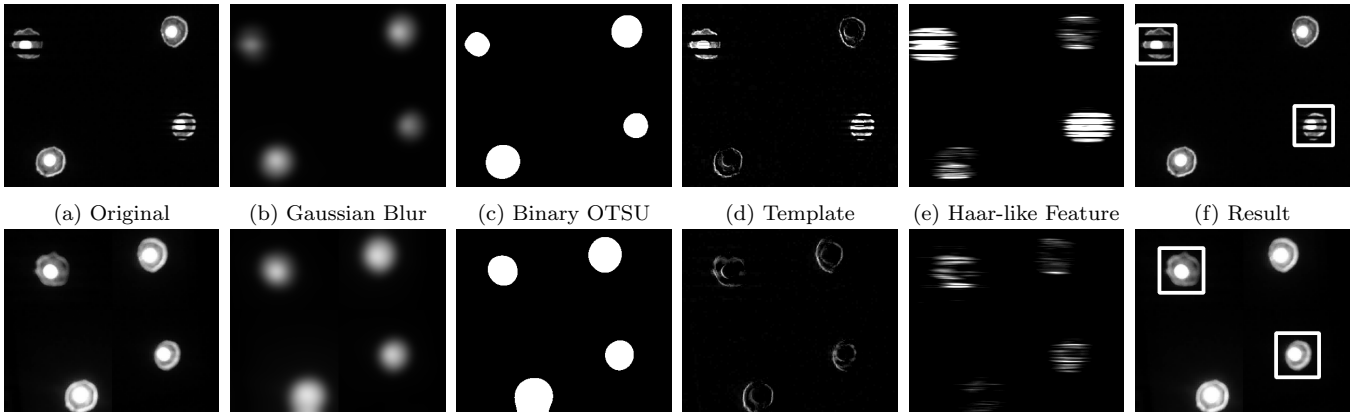
This scheme is also for addressing the unsynchronized issue. We select another signal period as the symbol delimiter. In order to easily **separate consecutive symbols** in the same image, i.e., two areas in the image showing signals with different periods, a signal with this signal period will be transmitted for a short duration between any two symbol transmissions as Figure 12 shows.

The detection step is done by calculating the value of the difference function described in Section 2.3 for all possible symbol delimiter locations in the image, but only with a fixed shift value that equals the the signal period of the symbol delimiter. If the minimum value of the function for all possible delimiter locations is larger than a pre-defined threshold, then it will output the location of the symbol delimiter.

3.5 LED Boundary Detection

To decode the lights, the first step is to find the boundary of each light in the received image.We follow the [6] to detect the lights individually. We slightly modified the detection process. To detect the light taken with larger exposure duration, we need to eliminate the background noise. Our method is maintain a buffer with latest few images, for the current image, we make a template by averaging all images in the buffer. Then we subtract the template from the current image to eliminate the noise. Another modification is the haar-like feature. Since there would be some other light in the image, we want to further detect the modulated light, that is, the light with strips. Thus the haar-like feature is added to filter the light with strips.

Figure 13 shows the light detection process step-by-step. There are four lights, two of them (the top-left and the bottom-right one) are modulated. We test the detection method with two exposure time settings: the larger one 1/400 sec and the smaller one 1/5000 sec. More sophisticated detection and tracking algorithms based on computer vision techniques can be utilized if necessary.



(a) Original (b) Gaussian Blur (c) Binary OTSU (d) Template (e) Haar-like Feature (f) Result

(g) Original (h) Gaussian Blur (i) Binary OTSU (j) Template (k) Haar-like Feature (l) Result

Figure 13: LED boundary detection step-by-step. The top row of images taken with exposure duration 1/5000 sec. The bottom row of images taken with exposure time 1/400 sec. The result image detects the light with strips.

4. IMPLEMENTATION

4.1 Components

Figure 14 shows the components in our proposed system. In the transmitting end, we use two different architectures to modulate the LED. First, as the system is still under development, the developed system should be sufficiently flexible so that it is easy to change various desires of the system, e.g., the modulation format, the protocol design, etc. To this end, we use a PC with USRP Hardware Driver (UHD) application to modulate the digital information in a packet into analog signal, in the format of digitized samples. The software-defined radio (SDR) is connected to the PC via Gigabit Ethernet. The SDR converts the digitized samples sent from the PC into analog signal. The VLC 2.0 front-end board converts the voltage varying signal to current varying signal and outputs that to the LED. The signal would determine the output intensity of the LED.

In the next stage, we try to minimize the cost of the transmitter. We use the Arduino Mega 2560 board instead, and we use Grove MOSFET which enable us to control higher voltage project, say 15VDC, with low voltage, say 5V, on microcontroller.

In the receiving end, we just use a camera to receive the optical signal and use simple computer vision and digital image processing mechanism to demodulate. Again, we have different architectures. First, we use PointGrey Flea3 camera [5], a high-speed camera built for experimental purposes, thus various parameters such as exposure time can be adjusted to a wide range of values. This allows us to examine system performance in different conditions.

We also use a number of smartphones to compare their performance. We take the videos from the built-in camera app, then decoding the video on PC.

The third architecture, ideally, is to decode the preview images on the smartphone directly without storing a video and decoding at external machine. We developed an iOS app.

4.2 Encoder

For the transmitting end of the system, we implement a UHD application to generate the signal. It takes a given bit stream, and produces encoded frames following a predefined bit-to-symbol mapping and the frame layout described in the previous subsection. We use 1 MHz sampling rate and a given transmitting frame rate to generate samples within one frame. These samples are then encoded with bit 1s and 0s with the corresponding symbol frequency. The frames are then turned into analong signal and transmitted via LED light.

For another architecture, we use the Fast PWM with OCRnA top mode and the interrupt mechanism to control the clock of the Microcontroller of the Arduino Mega board. We can set the symbol frequency, the duty cycle, and the symbol duration.

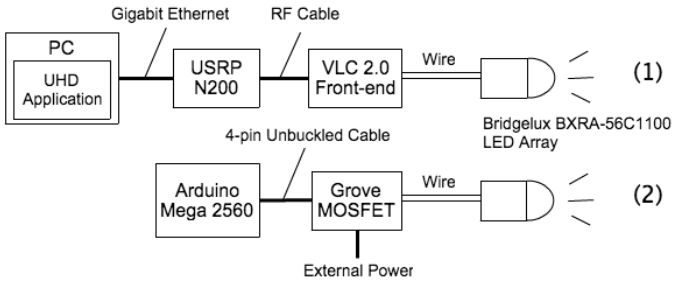
4.3 Decoder

We use the ffmpeg package to convert video into image frames. The decoder is implemented with OpenCV, which is a very powerful and efficient library to process images. The other is implemented in Objective-C with OpenCV as an iOS application for real-time processing. We will evaluate the processing time in Section 5.7.

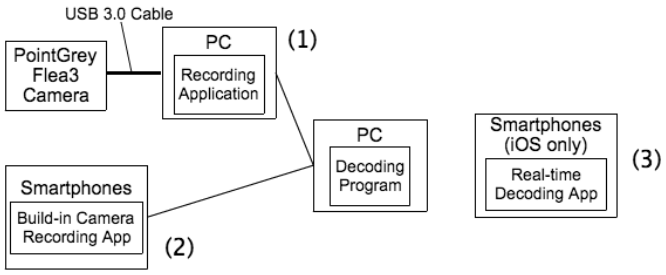
5. EVALUATION

5.1 Experimental Methodology

In Section 2.4.2, we derive the probability of symbol loss and mixed frame. The value is affected by the transmitting frame rate ($T_{f,tx}$), receiving frame rate ($T_{f,rx}$), the read-out time of the camera (T_r) and the height of the LED in the received image (H). Therefore, we



(a) Transmitter Components



(b) Receiver Components

Figure 14: System Componenets

present the experimental results of our unsynchronized VLC system with respect to these factors. The performance metrics we use in the experiments are Packet Reception Rate (PRR) and throughput.

Each symbol represents 4-bit, and thus we have 16 symbols. The symbol-to-frequency mapping table is generated by.... (formula). Figure 15 shows a photo illustrating our setup.

5.2 Different Receiving Frame Rate

First, we want to show the improvement of the decoding performance due to the design of the symbol delimiter, the sequence number, and the parity symbol.

We alter the receiving frame rate to obtain different tx/rx frame rate ratios from 0.9 to 2 to force unsynchronized transmitter and receiver pairs. That is, we fix the transmitting frame rate at 30 fps, and alter the receiving frame rate from 15 fps to 33 fps. We randomly generate 20 string sequences for each of the three settings: (1) no additional schemes; (2) with only symbol delimiter; and (3) with symbol delimiter, sequence number, and parity symbol. The length of the string sequence is 20 bytes. Note that we also record the videos at random starting time to introduce random phase difference between the transmitter and the receiver.

Figure 16(a) shows the result. When the receiving frame rate and the transmitting frame rate are very close to each other, the first setting has slightly lower PRR, as a number of received image frame will be mixed with two symbols and a portion of them will be demodulated correctly. The other two settings perform similarly. However, as the difference between the frame rates increase, the differences in PRRs become very obvious.

(a) (b)

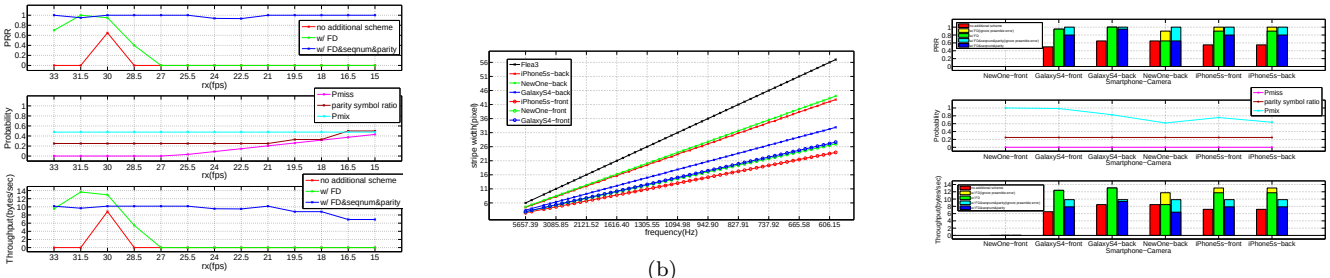
Figure 15: Experimental setup photos with different Receivers: (a) Smartphone (b) Flea3

The P'_{miss} and parity symbol ratio are also shown in Figure 16(a). For the third setting, we use P_{miss} to determine the number of parity symbols required to be inserted into a packet. When the receiving frame rate is higher than 19.5 fps, the parity symbol ratio is configured to be 0.25, which means for every three symbols, we add a parity symbol. (It is possible to use a parity symbol rate less than 0.25, following the pink line in the figure. However, we do not discuss such a case here, and will discuss the maximum throughput in Section 5.5.) When the receiving frame rate is between 19.5 and 18 fps, we use a parity symbol ratio of 0.33. When the receiving frame rate is lower than 18 fps, we use a parity symbol ratio of 0.5, which means for every symbol, we transmit twice. With all 3 schemes, close to 100% of the packets are received correctly, while the other two settings rapidly lose the ability to receive correct packets when the difference between frame rates increases. The results verify that our design can indeed address the issues caused by unsynchronized transmitter and receiver pair.

We can see when the receiving frame rate and the transmission frame rate are close, the setting with the symbol delimiter has the highest throughput. As the difference between the frame rates increases, the throughput of the third setting decreases due to the large parity symbol ratio.

5.3 Varying Inter-frame Interval

According to [4], the receiving frame rate of smartphone cameras exhibit more variability for several reasons. Even when the average frame rate appears steady, the inter-frame interval still varies. In order to show the improvement with our new design scheme, we emulate this phenomena with a system with time varying inter-frame interval. However, it is hard to alter the receiving



(a) (b) (c)

Figure 16: (a) Result under different rx fps. (b) Pixel width under different smartphone devices. (c) Result under different smartphone devices.

frame rate while receiving frames. We therefore use the Flea3 camera with fixed receiving frame rate and alter the transmitting frame rate before transmitting every symbol.

To simulate the various of transmitting frame rate, we use a truncated normal distribution with the mean value of 36.9458 fps, the standard deviation of 3.94, the min value 31.03 fps and the max value 42.86 fps. The values are based on observations presented in [4], which states that the receiving frame rate fluctuates significantly between 21 to 29 fps for an entirely white foreground, with an average of around 25 fps.

We again generate 20 random string sequences. The length of each string sequence is 20 bytes. For each transmitting symbol, we randomly pick a number from the truncated normal distribution and transmit the symbol with the generated frame rate. We use the Point-Grey Flea3 camera to record the video at fixed 30 fps.

5.4 Different Smartphone Cameras

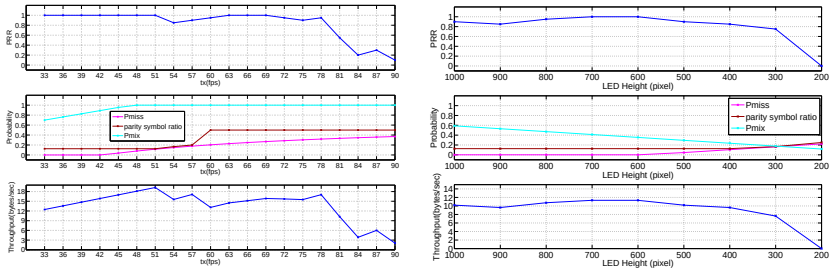
Third, we use several smartphones to compare the decoding performance under the three previous mentioned settings. Note that the receiving frame rate of the smartphones are not fixed. Moreover, the read-out time and the resolution of the camera of smartphones are also different. These will affect the probabilities of symbol loss and mixed frame. We again generate 20 random string sequences. The length of each string sequence is 20 bytes.

As the read-out time of the smartphones are not the same. This means different smartphone cameras may observe the different pixel width (signal period) under the same transmitting frequency. Therefore we use the preamble symbol to calibrate the T_r to minimize the error of the period estimates of all subsequent symbols in the packet. Figure 16(b) shows the pixel width of different smartphone cameras.

Figure 16(c) shows the result. The X-axis shows different smartphone cameras. We first calculate P_{miss} by averaging the frame rate of each smartphone camera. The obtained values are all zeros since the average fps are close to 30. However, we can see that P_{mix} is high.

For the first setting, we can see the PRR is about 50% to 60%. The result also shows that when the P_{mix} is higher, PRR of the first setting gets lower. For the second and third settings, basically that PRRs are similar since no symbol miss take place. However, we can see that most of the devices show that the third setting gets lower PRR then the second setting. Some of the devices also show that the third setting gets lower PRR then the first setting. Furthermore, the overall PRR is on the low side. We think the problem is due to decoding error. Because of the instability of the smartphone cameras, if the received pixel width of the preamble has slightly error, it will affect the third setting very much because the pixel width range we use in the third setting is larger than the first two settings. The yellow and cyan bars in the figure represent the result ignoring the preamble error. We can see that PRRs of the second and third settings are very close to 1. (From this experiment, we can see that the second setting is good enough for the communications in spite of using different smartphones. However, the current setting uses regular video recording mode rather than preview mode, and the receiver is very close to the transmitter. That is, the are many factors not considered yet and these will be discussed in the following subsection.) Therefore, the error are mainly caused by the preamble calibration rather than the unsynchronized issues. To address the preamble calibration error, we can transmit more preamble symbols at the beginning of the packet and average them when receiving these symbols. This solution can minimize the error caused by calibration but will increase the packet overhead at the same time.

We also found that the front-facing camera of HTC NewOne cannot decode any packet in all three settings. We observed that the behavior of the camera records every frame twice. Even when we extract half of the frames in the video, it is still not able to decode. Therefore, we think the problem is not related to our decoding method. Our results indicate that, the second setting can obtain the maximum throughput in most settings.



(a) (b)

Figure 17: (a) Result of iPhone5s under different transmitting frame rate. (b) Result of iPhone5s under different LED height.

5.5 Max Throughput under Different Transmitting Frame Rate

In this experiment, we fix the receiving frame rate and gradually increase the transmitting frame rate to determine the maximum achievable throughput, and when we can achieve it. As the transmitting frame rate gets higher, P_{miss} increases, and thus we need to use a higher parity ratio to compensate. However, the transmitted data rate also increases. We would like to investigate at what transmitting frame rate the maximum throughput can be obtained. In this experiment, we increase the transmitting frame rate from 33 fps to 90 fps. The iPhone5s is at 29.98 fps on average. We only use the third setting in this experiment.

Figure 17(a) shows the result of iPhone5s. We can see P_{miss} grows from 0 to over 0.5, and the parity symbol ratio we select is just above the P_{miss} line. Note that the parity symbol ratio we use is at most 0.5. Because we think the overhead costs too much if a symbol is transmitted over two times. We can see that PRR is more than 90% until the transmitting frame rate reaches 81 fps. Theoretically, the upper limit of the transmitting frame rate is bounded by the case of having two symbols in a received frame, which is given by

$$FPS_{Max,tx} = \frac{1}{(H - 2H_{SD})T_r} \quad . \qquad (11)$$

The max transmitting frame rate for iPhone5s is about 53.39 fps, two symbols in a received frame rarely happens. At that time the parity symbol can be used to recover data. Therefore, we can get even higher transmitting frame rate in practice.

The PRR is above 80% until reaches the transmitting frame rate of 81 fps. We can see that the iPhone5s can have high PRR when the transmitting frame rate is lower than 81 fps. We can see the throughput line grows with a trend - at the time the parity symbol ratio increases, the throughput would stay the same or drop a little. The maximum throughput of iPhone5s happens at 51 fps, and the throughput has increased by 1.28 times and reaches 19.25 bytes per second.

5.6 Different LED Size in the Image

So far, we have considered the factors of transmitting frame rate, receiving frame rate and the read-out time of the camera. There is one more factor - the image height, which may affect P_{miss} and P_{mix}. Thus, in this experiment, we try to change the height of the LED in the received image. There are two methods to do so, one is use the same setting with the previous experiments (which let the LED light full of the image) and simply crop the image into the height we want. The other is to move the receiver from near to far. We choose the latter method. When we get far away from the LED, although the intensity of light and the detection of the LED boundary may also cause some decoding error, we think the latter method exhibits a setting closer to the real situation.

We use the height (pixel) of the LED in the received image as our X-axis in Figure 17(b). The transmitting frame rate is configured to 30 fps. The iPhone5s is at 29.98 fps on average. We only use the third setting in this experiment. As to how far away we can reach, it depends on the sensor size of the camera, the focal length of the lens, and the size of the LED. If we know all the parameters of the camera and lens, the distance from the LED to the camera is proportional to:

$$distance(cm) \propto \frac{focal\_length(mm) * LED\_height(mm) * image\_height(pixels)}{LED\_height(pixels) * sensor\_height(mm)}$$

We also provide a comparison in Table 2, showing the relationship between the distance and LED height in the received image of Flea3 and iPhone5s. The real LED height we use in the experiment is 4.7 cm. In the table, we can see if the LED object in the image is around 500 pixel, the distance from the object to the Flea3 can be half a meter, while the iPhone5s can only be 15 centimeters. However, if the size of the image area illuminated by the light becomes bigger, (For

transmitter-end, we can simply use the lampshade to enlarge the light range. For receiver-end, we can use the reflected light from surrounding areas, such as a wall.) we can further lift the distance limitation.

Note that we still have a limitation that the transmitting symbol in the received image need to occupy at least two black-white strips for detecting the signal period. For example, symbol with the largest strip width received from Flea3 is 58 pixel, so the image area need to have at least a height of 232 pixel. It means if the height of LED in the received image is smaller than 232 pixel, then the received symbol cannot be decoded.

Figure 17(b) shows the results of iPhone5s, due to its longer read-out time and shorter time gap between receiving frames, P_{miss} is relatively smaller. Thus, when the LED has a height of around 300 pixel in the image, the transmission can still be received. The PRR is about 75%.

We can see the iPhone5s can achieve 9.5 $\sim$ 12 bytes per second when the LED height is above 500 pixel. As the LED object in the image becomes small, the throughput decreases significantly; due to both a smaller PRR and a large parity symbol ratio.

5.7 Time Measurement of Real-time iOS Decoder App

We have shown that our system work well for cameras on mobile devices. In this subsection, we want to know whether the decoding process can be executed in real-time with smartphone processors, we also implement an iOS decoding app, running on iPhone5s.

We use the AV Foundation framework, which provides an Objective-C interface for managing and playing audio-visual media in iOS applications. The application uses multiple threads. One thread obtains the frame buffers captured by the camera, and put the image buffers into the queue to be processed. The other thread takes the frame at the front of the queue and decodes it in real time without saving the image. Decoding results for the frame will be shown on the phone screen. Figure 18 shows the GUI of the iOS application; one can first touch the screen where the light locates to lock the focus and the exposure time. After pressing start button, the application will start to detect the LED location and decode the message.

Since we have set the FrameDuration of the captured video data output (for example, we set the video to 30 fps), we need to make sure that we have enough time to process a frame within one FrameDuration.

To optimize the processing time, we first add the optimization level of Apple LLVM 5.1 compiler as "-Os", generating a faster and smaller executable.

Next, we time individual operations to assess their computational complexity. We found that the majority of the processing time happens when we try to get

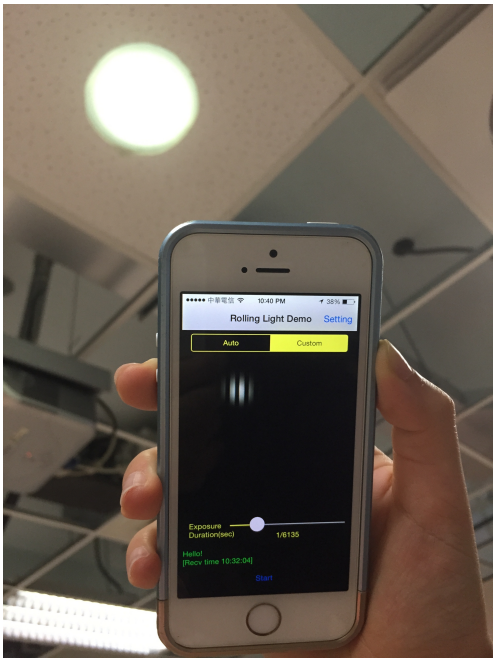


Figure 18: iOS application GUI

Table 3: Processing time breakdown in ms

Operation	time(ms)
Image Buffer to Mat Conversion	0.03
Symbol Delimiter Detection	3.25
YIN Period Detection	3.74
GUI Text Output	0.04
Total	7.06

the frame buffer and convert it to the UIImage, then convert the UIImage to the OpenCV Mat type. This step takes about 0.04 sec, which is greater than a frame duration. To reduce the time, we directly get the pixel base address and put it into Mat. Surprisingly, the time reduces to 0.00003 sec.

Moreover, we found that when we convert RGB to gray scale, it also costs about 0.02 sec. To reduce the time, we change the video output pixel format to YUV color space, and use the luminance(Y) channel as the gray scale value. As a result, we save 0.02 sec.

After a series of optimizing steps, the total processing time of a frame is about 0.007 sec which is less than 1/30 sec. Since our current implementation already meets the real-time target, we do not explore further optimizations, even though some operations could be sped up more. Table 3 lists the time taken for each operation.

6. RELATED WORK

6.1 Screen-to-camera Communications

Table 2: Distance table of Flea3 and iPhone5s.

LED height in image (pixel)	1000	900	800	700	600	500	400	300	200	100
Flea3 distance (cm)	48.5	53.9	60.6	69.3	80.9	97.0	121.3	161.7	242.6	485.2
iPhone5s distance (cm)	7.75	8.61	9.68	11.07	12.91	15.49	19.36	25.82	38.73	77.46

A number of prior works [7, 8, 4] utilized cameras as the receiver to implement one-way communication links. Those works utilized a LCD screen as the transmitter. The main benefit of this approach is that both the ability to display different colors and the high resolution of the LCD could be fully exploited, to improve the data rate.

PixNet [7] utilized the large quantity of pixels on a LCD to form OFDM symbols to transmit data to the camera receiver, and the implemented prototype has perspective correction ability.

COBRA [8] proposed a similar approach that uses pixels as as barcodes and utilized the color information of pixels to achieve higher signal to noise ratio. These works, however, did not solve the problems raised by the unsynchronized transmitter and receiver pair. The presented solution is to repeat the same packet transmission twice in order to guarantee the reception of all signal segments.

LightSync [4] introduces inter-frame erasure coding and line sequence number to reduce the high packet error rate due to synchronization issues. The approach also actively filters out rolling-shutter effects.

VRCodes (NewsFlash) [9] takes advantage of the rolling shutter effect to decode visual tags displayed on LCDs. The tags use multiple pixels of different colors, modulated at up to 120Hz to transmit data. The technology exploits the "flicker-fusion threshold" of the human eye to blend the tags into the background by rapidly flashing complimentary hues of color, still visible to a rolling shutter camera. This work suggests how different color channels could be used to increase data throughput, while still keeping transmissions imperceptible to humans.

Visual MIMO [10, 11, 12] is a method that uses any light-emitting spatial array as the transmitter and uses ordinary cameras as the receivers. It proposes a photometric modeling for hidden imagery and can realize a dual use of the electronic display: to display an image for human observation while simultaneously transmitting hidden bits for decoding by a camera.

Compared to these works, we proposed to use only a single LED light as the transmitter and a commonly available CMOS camera as the receiver, and many design considerations are not the same as these prior research works utilizing multiple optical transmitting sources.

6.2 Single-LED-to-camera Communications

Three works [13, 14, 15] have previously investigated how to implement single-LED-to-camera communication links. In [13], the author proposes undersampled frequency shift OOK (UFSOOK), which allows the use of a high signal frequency while the modulation can still be decoded by a common camera with 30 fps (frame rate per second). The scheme is compatible to both global shutter and rolling shutter cameras, but the maximum achievable data rate is only half of the frame rate per light source.

In [14], the authors utilized rolling shutter sampling to encode data with Manchester coding at high symbol rate, and was the first work to take advantage of the rolling shutter to implement CamCom. However, the implementation exhibits a high packet drop rate due to the long processing time of the decoder application per frame and lack of considerations for synchronization issues. Thus, the reception is not continuous and some frames are inevitably lost. The transmitter then needs to send signal. In addition, the transmitting light needs to illuminate the entire image for the receiver to demodulate the transmission.We can achieve a data rate much higher than 50% of the frame rate per light source, and have designs to address the issues caused by the exposure operation of the camera and unsynchronized nature of the communication link. A further drawback in this work is that the modulated signal can produce a human perceivable flicker from transmitting LED. The authors alleviate this by imposing a DC bias on the signal, which in turn decreases its dynamic range and SNR at the receiver. This makes the scheme require significantly brighter lights and more complex driving hardware than our proposed approach.

In [15], the author used binary frequency shift keying (BFSK) of a high frequency PWM signal to encode data that can be used as localization landmarks. On the receiver side, they exploited rolling shutter camera sensors to detect high-frequency changes in the intensity of light reflected off surfaces, which is indirect line-of-sight of the camera. The data rate is only 1.25 bytes per second (the authors claimed that it is fast enough to send an ID code) while in our work there is a need to increase the data rate to communicate rather than just perform localization. To detect the transmitting frequency, they used 1-D fast Fourier transform. However, we have tried the FFT method before and it usually requires a threshold to eliminate the noise. It is hard to have a general threshold which can be adopted in different environments. For the receiver which is unsynchronized with the transmitter, they proposed a sliding

window approach where they first stitch all of the captured frames together into a single, long image. Then they can detect the frequency by sliding the window and locating the window position corresponding to the highest power. However, we can stitch the images only if the light is full of whole image (for example, reflected from the surface). Moreover, if there is a symbol loss between the two frames, then the result of stitching images would likely be inaccurate. In our work, we use additional schemes to address these problems.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis, we implemented a CamCom system that utilizes a **single LED** as the transmitter and a common **rolling shutter CMOS camera** as the receiver. We presented considerations for addressing issues caused by unsynchronized transmitter and receiver, including the introduction of **symbol delimiter, sequence number, and parity symbol** in the transmission. In each situation, we calculate the probability of symbol loss and mixed frame first, then decide which setting is suitable and adjust the parity symbol ratio by the probability.

Experimental results confirm that the additional designs would significantly improve the packet reception rate (PRR) from 0.6 to 0.99 when the transmitting and receiving frame rates are close. We also can improve the PRR from 0.0 to 0.8∼1.0 even when the transmitting and receiving frame rates have a large difference or when the the receiving frames have varying inter-frame intervals. A variety of front/ back-facing cameras of smartphones can be used in our system without any modification. One can increase the throughput by 1.2 times by increasing the transmitting frame rate. Data still can be received when the height of LED object in the image is more than 400 pixel. The overall throughput reaches 18 bytes per second. The decoding application proves that series of step can be processed in real-time, with each image frame completely demodulation in merely 7 milliseconds.

7.2 Future Work

Improve the iOS real-time decoding app. There are many things can be improved. First is the LED detection. For now, we just threshold the image to binary and find the bright part as the boundary. However, it is not an efficient way and is affected by the surrounding light noise. We want to adopt the characteristics of the strips and use some Digital Image Processing (DIP) techniques to figure out. On the other hand, to get shorter exposure time, the current method is to move the camera to be very close to the LED light and lock the exposure time. It is not convenient in the real life scenarios since we are not able to get very close to the light. The Landmark paper [15] have proposed the algorithm which find the brightest part in the image then focus/ expose at it, which could be adapted to our system.

Increase the data rate. There are severals ways to increase the data rate, we have tried increasing the transmitting frame rate. We also can increase the number of bits represented by each symbol. We can obtain a higher data rate with more transmitting symbols to carry the bits.

Consider more scenarios. Currently we focus on the static environments. We still need to do more experiments in the moving scenario or dynamic environments to access the feasibility and reliability of our design.

8. ACKNOWLEDGEMENTS

This work is supported in part by National Science Council, National Taiwan University, and Intel Corporation under grants NSC-102-2911-I-002-001, NSC-102-2221-E-002-093-MY2, and NTU-103R7501.

9. REFERENCES

[1] J. Nakamura, *Image Sensors and Signal Processing for Digital Still Cameras*. Boca Raton, FL, USA: CRC Press, Inc., 2005.

[2] A. El Gamal and H. Eltoukhy, "CMOS image sensors," *Circuits and Devices Magazine*, vol. 21, no. 3, pp. 6–20, 2005.

[3] A. De Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[4] W. Hu, H. Gu, and Q. Pu, "LightSync: unsynchronized visual communication over screen-camera links," in *Proceedings of Mobile computing and networking*. ACM, 2013, pp. 15–26.

[5] "PointGrey Flea3 Product Datasheet," http://www.ptgrey.com/products/flea3/Flea3_Datasheet.pdf.

[6] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta, "Luxapose: Indoor positioning with mobile phones and visible light," in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '14. New York, NY, USA: ACM, 2014, pp. 447–458. [Online]. Available: http://doi.acm.org/10.1145/2639108.2639109

[7] S. D. Perli, N. Ahmed, and D. Katabi, "Pixnet: interference-free wireless links using lcd-camera pairs," in *Proceedings of Mobile computing and networking*. ACM, 2010, pp. 137–148.

[8] T. Hao, R. Zhou, and G. Xing, "COBRA: color barcode streaming for smartphone systems," in

Proceedings of Mobile systems, applications, and services. ACM, 2012, pp. 85–98.

[9] G. Woo, A. Lippman, and R. Raskar, "Vrcodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter," in *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, ser. ISMAR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 59–64. [Online]. Available: http://dx.doi.org/10.1109/ISMAR.2012.6402539

[10] A. Ashok, M. Gruteser, N. Mandayam, J. Silva, M. Varga, and K. Dana, "Challenge: Mobile optical networks through visual mimo," in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '10. New York, NY, USA: ACM, 2010, pp. 105–112. [Online]. Available: http://doi.acm.org/10.1145/1859995.1860008

[11] A. Ashok, M. Gruteser, N. Mandayam, T. Kwon, W. Yuan, M. Varga, and K. Dana, "Rate adaptation in visual mimo," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*, June 2011, pp. 583–591.

[12] M. Varga, A. Ashok, M. Gruteser, N. Mandayam, W. Yuan, and K. Dana, "Demo: Visual mimo based led - camera communication applied to automobile safety," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 383–384. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000046

[13] R. D. Roberts, "Undersampled frequency shift ON-OFF keying (UFSOOK) for camera communications (CamCom)," in *Proceedings of Wireless and Optical Communication Conference*. IEEE, May 2013, pp. 645–648.

[14] C. Danakis, M. Afgani, G. Povey, I. Underwood, and H. Haas, "Using a CMOS camera sensor for visible light communication," in *Proceedings of Globecom Workshops*. IEEE, Dec 2012, pp. 1244–1248.

[15] N. Rajagopal, P. Lazik, and A. Rowe, "Visual light landmarks for mobile devices," in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 249–260. [Online]. Available: http://dl.acm.org/citation.cfm?id=2602339.2602367