# TUDelft

Delft University of Technology

# Passive Screen-to-Camera Communication

Ghiasi, Seyed Keyarash; Kaldenbach, Marco ; Zuniga, Marco

**Citation (APA)**
Ghiasi, S. K., Kaldenbach, M., & Zuniga, M. (2024). Passive Screen-to-Camera Communication. In C. Ceballos (Ed.), *Proceedings of the 2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)* (pp. 35-43). IEEE. https://doi.org/10.1109/DCOSS-IoT61029.2024.00016

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

# Passive Screen-to-Camera Communication

Seyed Keyarash Ghiasi, Marco Kaldenbach, and Marco Zuniga

Delft University of Technology, The Netherlands

Email: {s.k.ghiasi, m.n.j.kaldenbach, m.a.zunigazamalloa}@tudelft.nl

*Abstract*—A recent technology known as *transparent screens* is transforming windows into displays. These smart windows are present in buses, airports and offices. They can remain transparent, as a normal window, or display relevant information that overlays their panoramic views. In this paper, we propose transforming these windows not only into *screens* but also into *wireless transmitters*. To achieve this goal, we build upon the research area of screen-to-camera communication. In this area, videos are modified in a way that smartphone cameras can decode data out of them, while this data remains invisible to the viewers. A person sees a normal video, but the camera sees the video plus additional information. In this communication method, one of the biggest disadvantages is the traditional screens' power consumption, more than 80% of which is used to generate light. To solve this, we employ novel transparent screens relying on ambient light to display pictures, hence eliminating the power source. However, this comes at the cost of a lower image quality, since they use *variable* and out-of-control environment light, instead of generating a *constant and strong* light by LED panels. Our work–dubbed PassiveCam– overcomes the challenge of creating the first screen-to-camera communication link using passive displays. This paper presents two main contributions. First, we analyze and modify existing screens and encoding methods to embed information reliably in ambient light. Second, we develop an Android App that optimizes the decoding process obtaining a real-time performance. Our evaluation, which considers a musical application, shows a Packet Success Rate (PSR) of close to 90%. In addition, our real-time application achieves response times of 530 ms and 1071 ms when the camera is static and when it is hand-held, respectively.

## I. Introduction

In today's technology-driven world, novel communication methods are becoming prevalent. An emerging example is screen-to-camera communication, where data is embedded into videos in a way that is invisible to viewers, but not to cameras. This opens up opportunities to increase communication capacity, since these screens are pervasive and operate on visible light, instead of the overly-congested and expensive radio spectrum. Motivated by these advantages, there are several studies in screen-to-camera communications [1]–[5], whose focus is on increasing the PSR without affecting the viewer's experience.

The research community has achieved large feats in screen-to-camera communication, but the main downside is the power cost. As an example, a regular LCD monitor uses at least 26 Watts[1]. The culprit is its back light, which consumes more than 80% of the total power. A new generation of transparent screens are removing that power cost by using ambient light instead of LEDs. The image quality is not as sharp as in traditional screens but they consume much less energy, which is a major advancement in designing novel screens with a low ecological footprint.

---

[1]Based on measurements on a LG LM215WF4-TLE7



Fig. 1: An application of passive displays: a video is shown in an airport using ambient light. The display remains transparent and does not block the outside's view. Image from [6].

Transparent screens are already being deployed in buses, airports and offices despite being a novel technology. They are transforming the facades of different environments to transmit relevant information on demand. For example, smart windows facing the runway of airports display advertisements to the passengers, as shown in Fig. 1. However, displaying plain pictures and videos on these monitors is not exploiting their potential. In other words, what if the transparent screen transmitted more information? For instance, while showing an enticing holiday destination some users may want more information on discounted hotels or restaurants in that specific location. To achieve that, a user could point her smartphone's camera towards the video and obtain that *embedded and invisible* information.

Inspired by advances in screen-to-camera communication, we present PassiveCam, a system that enables the first wireless link between transparent screens and smartphones. While being advantageous in terms of power consumption, designing this new link introduces two main challenges. First, compared to the original back light of a display, ambient light is weaker and variable, resulting in lower image quality, which in turn makes the communication link more error-prone. Second, most state-of-the-art (SoA) works do not consider proper human-computer-interaction designs. Some studies do not evaluate the effect of a user's camera movements, evaluating their work only with their camera on a tripod; other studies do not process the videos in real time; and a few systems rely on visual markers, either static or dynamic, to delineate the borders of their transmitting area, affecting the system's aesthetics.

To overcome the limitations mentioned above, in this paper, we propose a novel communication system with a passive screen acting as transmitter and a smartphone (App) acting as a receiver. In general, PassiveCam provides the following contributions:

*1) A passive screen transmitter.* Passive screens are not off-the-shelf components yet, but we test two transparent screens: one obtained from a specialized vendor [6] and the other created by us. We describe how to remove the back-light of normal displays so the interested researchers can create their

own passive screen. Our modified screen reduces the power consumption by 80%. Regarding the modulation process, we build on top of the SoA, however, we improve their encoding with three steps: Gaussian kernels, step encoding, and an optimization of texture analysis.

*2) A reliable real-time receiver.* Our receiver, running on a custom Android App, tackles two important points to facilitate a seamless user experience. First, unlike many prior studies where a static phone on a tripod is used, we implement a simple hand motion filter so that users can hold the phone. Second, to attain a real time response, we propose a new method to detect the Region of Interest (ROI) and discard invalid frames.

*3) A new application:* As far as we know, the use of passive screens for communication (commercial or modified) has not been explored before. We evaluate the link's reliability and delay with controlled experiments, but we also propose a novel application. Together with personnel of a company, we target the following scenario. Considering the massive windows (passive screens) at airports, multiple types of videos could be displayed (music, tourism, news, etc) and users may want to listen to one of those streams. To do so, they could point their phone toward the desired video, and our passive link would decode the video ID and the exact time of the frame to provide its synchronized audio. A prototype of such application is shown in two videos: one in black-and-white [7] and the other in color [8]. Overall, PassiveCam has an offline code success rate of close to 90%. Besides, our real-time Android application acquires synchronization within 530 ms when mounted on a tripod and 1071 ms when hand-held.

## II. A PASSIVE TRANSMITTER

In order for a display to qualify as "passive", it requires using ambient light, in contrast to *normal* Liquid Crystal Display (LCD)s emitting light by themselves. Fig. 2 shows the difference between active and passive screens. Transparent screens are not widely sold in the consumer market except by few select companies. Therefore, we will first describe how a passive screen can be obtained from a normal screen, and then, we will explain the modulation process.

### A. Passive displays

To understand how a passive screen can be made, first, we need to explain the structure of a typical display. LCDs are made of several components as shown in Fig. 3. ① is an array of Light-Emitting Diode (LED)s or a fluorescent lamp emitting the monitor's backlight. Layer ② is an acrylic layer dispersing the light uniformly across the monitor. The dispersive layer scatters light to both the back and front, but the light that goes into the back side of the monitor is wasted. Therefore, a reflective layer (③) is used to recover this light and send it towards the viewer. At the end, layer ④ is a transparent glass layer containing pixels and displays the image to the user.

Based on our measurements, the screen's backlight consumes more than 80% of the total LCD's power. Thus, to make
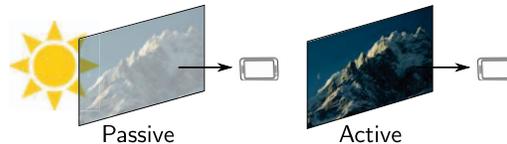


Fig. 2: In passive displays, the backlight is provided by sunlight. In standard (active) displays, the backlight is provided by LEDs. If data is embedded in the frames, a smartphone can decode information with screen-to-camera communication.
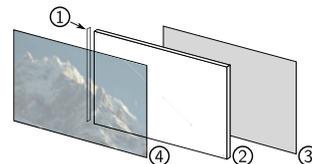


Fig. 3: Layers and components inside an off-the-shelf monitor: ① an LED or lamp, ② a light dispersion layer, ③ metallic reflective layer, ④ a transparent LCD glass containing pixels

a passive screen, we need to remove all components except layer ④. It is important to note that the interface that controls the display is only connected to layer ④, which contains the pixels. Thus, it can still show pictures and videos. The process required to disassemble a screen is not complex and can be done with any LCD, however, it has to be done carefully to avoid damaging the glass layer. An example of our passive screen, placed over a window, is shown in the videos of [7] and [8].

Another type of passive display we use is a "smart window" [6]. Smart windows have similar operating principles to the layer ④ of a monitor. Nevertheless, they are optimized to cause *less attenuation* to the light, resulting in more transparency and a *higher contrast*. This lower attenuation of light comes at a cost: smart windows only provide black-and-white images. To show colored pictures, normal monitors have color filters on their pixels, which dissipate light excessively. As a result, a smart window is made without these color filters. On the contrary, a modified LCD *looks* darker than a smart window, providing in a lower Signal-to-Noise-Ratio (SNR) when used as a transmitter.

Overall, a smart window is optimized to operate as a black-and-white transparent screen, however, they are not easily accessible yet. Therefore, we evaluate both types of displays. Besides, we develop our final application on a modified display to show that a passive link can also be made without a high-end smart window.

### B. Data embedding

Embedding data into videos has been considered by several works [1]–[4], all of which use active screens. In this section, we first present the basic principles of data embedding, and then, we describe how we build upon these SoA.

*1) Basic principles:* Data embedding exploits the slow response of human eyes. If a sequence of colored pixels are shown fast enough, the human eyes would perceive the

36

average color. Based on this principle, we present a general encoding algorithm that reduces visual effects to human eyes.

An algorithm that embeds data takes two main inputs: a video with frames indexed by $f_0, ... f_n$, and the *data* that has to be encoded in those frames. In visual communications, data is represented by *binary frames*. A binary frame is similar to a Quick Response (QR) code showing a checkerboard pattern. To take advantage of the *basic principle* above, the data encoding algorithm works as follows. First, every frame in the original video is duplicated. Then, a modulation value of $\Delta$ is calculated for each pixel. The pixel at location (x,y)– denoted by $f_i(x, y)$–in the first frame is then increased by its respective $\Delta(x, y)$, i.e. $f_i^+(x, y) = f_i(x, y) + \Delta(x, y)$. In the other frame, the modulation values are subtracted, i.e. $f_i^-(x, y) = f_i(x, y) - \Delta(x, y)$. The resulting $f_i^+$ and $f_i^-$ are called complementary frames, and if played fast enough, the viewer's eyes shall average them to see the frame $f_i$, although modulated frames are being displayed.

Regarding the creation of binary frames, the SoA proposes custom procedures, however, we opt for QR codes for two reasons: (1) They are mature and well-developed, thus can be easily integrated into an application, and (2) The implementations are already optimized and tested, hence reducing the overhead of designing a receiver pipeline.

*2) Lightness modulation:* The binary frame indicates the location of the pixels that need to be modulated, but it does not state how to derive $\Delta(x, y)$. For this, the SoA proposes various modulation schemes [1]–[4]. We build upon schemes that modulate *lightness* instead of *color*, as our smart window only operates in black-and-white.

Several approaches use color spaces to guide the modulation process. The CIELAB standard divides the color space into two main dimensions: lightness ($L^*$) and color ($a^*, b^*$). In our case, to modulate a given pixel, the lightness is changed by $\Delta L^*$. Following the complementary modulation above, if a pixel falls in a white region of the QR code, we set the its $\Delta(x, y)$ to $+\Delta L^*$. Otherwise, it is set to $-\Delta L^*$. Therefore, for each frame, we two complementary modulated frames are generated. Next, we first describe the building blocks we borrow from the SoA, ChromaCode [2], and after that, we describe our improvements.

**Building blocks from the SoA.** ChromaCode's key contribution is to show that the parameter $\Delta L^*$ depends on the *human perception of color* and the *texture* around a pixel. The human perception is self explanatory. Regarding the texture, for instance, changes in lightness are less perceptive in *rough* images (like rocks) than in *soft images* (like clear skies). Thus, the same modulation depth $\Delta L^*$ would affect the flicker effect of each pixel differently.

*Step 1: Perception-based derivation of $\Delta L$.* To determine a proper change of lightness $\Delta L_1^*$ for a pixel location (x,y), ChromaCode employs the CIEDE2000 formula to take human perception into account. The complete formula is explained in [2]. Below, we present a re-arranged equation that suits our purpose. Denoting $L_1^*(x, y)$ as the lightness of pixel $(x, y)$, $k_L$

as a parameter to compensate for different viewing conditions, and $\Delta E_{00}$ as the perceived difference in color between two *complementary* pixels; the modulation depth $\Delta L_1^*(x, y)$ is given by:

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

In the above equation, $L_1^*(x, y)$ is given for every pixel and $k_L$ is set to 1. The important part is to determine $\Delta E_{00}$. A large value increases the signal's SNR but also the flicker effect. A $\Delta E_{00}$ between 1 and 2 results in acceptable flicker levels.

*Step 2: Texture-based adaptation of $\Delta L$.* In the prior step, $\Delta L^*$ considers only the perceived difference in color $\Delta E_{00}$, but not texture. This step estimates the texture of a pixel area by measuring how abruptly the pixels change. The bigger the changes, the rougher the texture, and the less obvious the flicker effects.

A common approach to calculating texture is Regional Texture Analysis, which constructs a Gray Level Co-occurrence Matrix (GLCM) for every pixel. Based on this matrix, one can calculate the contrast $C(x, y)$ in a pixel area. The construction of the GLCM matrix and contrast is detailed in the appendix. Using this method, ChromaCode [2] further proposes an *average* texture metric $T(x, y)$ around pixel $p(x,y)$:

$$T(x, y) = \frac{C(x, y)}{S(x, y)} \quad (2)$$

where $C(x, y)$ is the contrast and $S(x, y)$ is the number of pixels in a $n \times n$ area around pixel $(x, y)$. Based on $T(x, y)$, the following scaling parameter $\alpha(x, y)$ is derived to adjust the modulation depth based on texture:

$$\alpha(x, y) = \frac{T(x, y)}{T_{max}}(1 - k) + k, \quad (3)$$

where $T_{max}$ is the maximum value of $T(x, y)$ in the frame, and $k$ is a parameter with a default value of 0.5.

In the end, the modulation depth of a pixel $\Delta L_2^*(x, y)$ is determined by the *color perception* $\Delta L_1^*$ and *texture* $\alpha(x, y)$.

$$\Delta L_2^*(x, y) = \Delta L_1^*(x, y) \cdot \alpha(x, y). \quad (4)$$

This value modifies the lightness of each pixel in the video.

**Improvements over the SoA.** The SoA provides a valuable starting point to embed information, but there are a few shortcomings that need to be improved for PassiveCam .

The SoA relies on better transmitters, with constant strong light; and better receivers, with cameras having 120 or 240 Frames Per Second (FPS). Our passive transmitter has weak variable light, and the phone we use has 60 FPS (to develop a more inclusive system). These two differences cause more flicker effects due to variable lighting conditions and a slower modulation. To ameliorate these flicker effects, we propose two improvements.

*Improvement 1: Gaussian kernels.* Given that the *sharp* borders in the binary frames cause noticeable artefacts, we apply a Gaussian kernel over the binary frame to smooth the
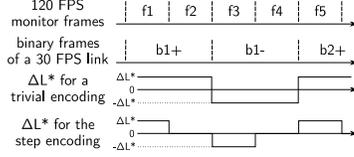
Fig. 4: Timeline of step encoding versus standard encoding.

sharp transitions. This process blurs the borders, making the embedding less noticeable, as surveyed in Section IV-E.

*Improvement 2: Step encoding.* In screen-to-camera communication, the frame rate of the receiver (camera) must be the same or higher than the transmitter's (screen). There are still many smartphones limited to capturing at 30 FPS in real-time. In this case, the screen must also transmit at a maximum of 30 FPS. This limitation causes the capacity of most screens with 60 or 120 FPS to be underutilized. Therefore, we propose a method to use this extra frame rate for the sake of flicker reduction. In the SoA, a modulated pixel changes from $+\Delta L^*$ to $-\Delta L^*$, making for an abrupt change of $2\Delta L^*$. However, if the screen can work at a higher rates than the camera, the encoding can be done in smaller, giving the impression of a smoother transition. For instance, let us consider a display with 120 FPS. If we aim for the transmission of binary frames at 30 FPS, a trivial configuration is to have two frames with $+\Delta L^*$ and two frames with $-\Delta L^*$. On the contrary, we can have one frame with no modulation, one frame with $+\Delta L^*$, another frame without modulation, and the last frame with $-\Delta L^*$. In this way, the lightness transition between frames is reduced by half, from $2\Delta L^*$ to $\Delta L^*$, ameliorating flicker effects. The trivial modulation and and the step encoding are both shown in Fig. 4 for this example.

*Improvement 3: Optimized texture analysis.* Our third improvement is related to optimizing the computational performance. Embedding information on videos requires several signal-processing steps. The most demanding step is the texture analysis done with the GLCM matrices. This high overhead would not be a problem if data was embedded only once. However, for applications that require periodic encoding, potentially in real-time, it is worth using the optimized method presented in Section VIII, which can reduce the encoding time by a factor of 500.

## III. DECODING

To decode a video, a smartphone's camera is used. We consider two scenarios: *offline* and *real-time* decoding. The *offline* process first records the video and then recovers data. In contrast, the *real-time* process starts recovering data as soon as the frames are captured. The *real-time* process targets realistic scenarios of a user receiving data from the screen, and the *offline* process aims at evaluating the encoder's performance, regardless of the processing power of a user's phone.

### A. Offline decoding

For the offline decoding, we mount a smartphone on a tripod, and its camera records the encoded video, as described in Section II-B. Next, the files are transferred to a PC for data extraction using the following processing steps:

Step 1) Adjusting the camera's frame rate: As stated by the Nyquist law, the sampling rate should be at least twice the highest frequency in the transmitted signal. With complementary modulation, the highest frequency is half the modulation's frame rate. Thus, if the camera captures data with at least the same frame rate as the video, Nyquist is satisfied. Capturing at a higher rate can be advantageous as there is less chance of frame loss due to camera jitter. However, rolling shutter effects might become present.

Step 2) Adjusting the camera's ISO and shutter speed: ISO is the sensitivity of the camera's sensor to light, and the shutter speed is the duration of the sensor's exposure to light. In CMOS image sensors, there is a chance of capturing a rolling shutter effect, depending on the shutter speed. Usually, increasing the shutter speed (reducing the exposure) removes this effect. However, it darkens the pictures, resulting in the loss of encoded data. Hence, by increasing the ISO, brightness can be adjusted again. Based on the ambient light, these values can differ. In Section IV, we do experiments using different levels of ISO and shutter speed to find the best setup.

Step 3) Data extraction: In Section II-B, we mentioned that the encoding algorithm makes complementary frames: if in frame $i$ (denoted by $f_i^+$), the pixel (x,y) has a lightness of $L_i(x,y)^* \pm \Delta L_i(x,y)^*$, the lightness of the (x,y) pixel in its complementary frame (denoted by $f_i^-$) will be $L_i(x,y)^* \mp \Delta L_i(x,y)^*$. Therefore, subtracting $f_i^+$ and $f_i^-$ yields a new frame with pixel values of $\pm 2 \times \Delta L_i(x,y)^*$. This last frame contains the encoded QR information. Considering this differential process, we implement the decoder.

Following Step 1, we assume that the camera's frame rate is equal to the transmitter's, thus, each frame is captured once and no frames are missed[2]. As a result of the matching rate between the screen and camera[3], the receiver has all the frames and their complements in order, i.e. $\{..., f_{i-1}^+, f_{i-1}^-, f_i^+, f_i^-, ...\}$. Then, we subtract all captured frames from the one before. Our goal is to keep the result when a frame is subtracted from its complement. However, an important point is that the receiver cannot know in advance which frames are complementary. Therefore, in every other subtraction, it operates on frame $i$ ($f_i^+$) and the complement of the previous (non-matching) frame ($f_{i-1}^-$), yielding a corrupt QR code since two different binary frames are subtracted. As we will show in Step 5, these corrupt frames can be detected and do not pose any problems besides consuming the decoder's time.

To generalize our assumption about the frame rates, if the camera captures at a higher rate, there will be more instances of the same frame. In that case, subtracting consecutive frames

---

[2]This assumption helps simplify the explanation. In practice, some frames will be missed, but they can be recovered using error correction codes

[3]Later, we generalize the decoding procedure for higher camera frame rates

sometimes results in a dark image, i.e. when the captured instances of the same frame are subtracted.

Step 4) ROI detection: Since the binary codes may not cover the entire frame, a captured video will likely contain both encoded pixels and other surroundings, including non-modulated regions of the video. Before passing the pixels to a QR decoder, the region containing valid data has to be identified. Previous works use visual markers around a screen so that the ROI can be easily detected, but we use a simpler technique. Going back to Step 3, we notice that in subtracted frames, all regions that are not encoded (shifted by $\Delta L^*$) have a lightness of 0, as their pixel values are the same as in their complementary frames. Using this fact to determine our ROI, we just need to find the smallest region inclusive of all pixels with a non-zero lightness, which is a simple task in image processing.

Step 5) QR decoding: This is the last step to extract binary data from pictures. For offline decoding, we use a QR decoder library in Python, and process the ROIs found in Step 4. However, as mentioned in Step 3, every other subtracted frame contains corrupt data. This does not pose any problems in offline decoding, since QR codes have checksums and corrupt frames can be detected and dropped. However, attempting to decode those frames delays the processing, which becomes problematic in a real-time system, which will be discussed below.

### B. Real-time decoding

After validating the encoding with the offline process, we implement the receiver pipeline on a smartphone to build a real-time link. When a person uses the phone in real-time, the receiver faces extra challenges mentioned below.

*1) Hand movement filtering:* A user, unlike a tripod, cannot hold a phone completely steady. Therefore, we have to remove the effect of camera movements before decoding. To address this dynamic, we implement a simple hand motion filter that uses prominent points in consecutive frames as *anchors*, and calculates a matrix to map these prominent features between frames. By applying the inverse transform to the shifted image, its pixels can align well with their counterpart in the other frame, hence alleviating the hand movement effect.

*2) Latency optimization:* As mentioned before, subtracting two non-complementary frames yields undecodable QR codes. Following the method in Step 3, undecodable codes would appear in every other subtracted frame. If the corrupt data is speculated preemptively, it can be discarded right away, without waiting for the QR decoder to do so. Such receiver is called to be *synchronized*.

To acquire synchronization, we build upon the observation that the correct frames are alternating, therefore, at the beginning, the pipeline makes an attempt to decode all frames, corrupt and healthy, but soon after the first successful decoding, the pipeline sends every other frame to the decoder, and discards the rest. Some factors, such as camera jitter, might result in losing this synchronization over time. Therefore, we


(a) Active monitor     (b) Smart window

Fig. 5: Setup of the experiments.
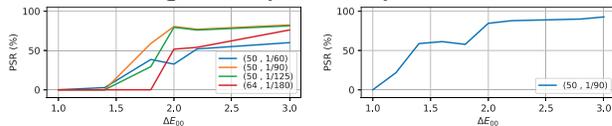

(a) Active monitor at 60 Hz    (b) Passive Monitor at 30 Hz

Fig. 6: Decoding performance on active and passive monitors.

should perform periodic checks, and if the decoding error is above a threshold, a synchronization has to be performed again.

## IV. EVALUATION

### A. Core setup

In our experiments, we use three screens: The first screen is an active (normal) LCD that we use as a baseline to compare with passive displays (Fig. 5a), the second one is a modified passive display obtained from an LG LM215WF4-TLE7 monitor using the procedure of Section II, and the third one is a smart window obtained from a manufacturer [6]. The passive screen and the smart window do not emit any light, therefore, we install them in front of our office's window, as shown in Fig. 5b. On the receiver side, a smartphone camera is placed at various distances up to 1.5 m from the screen. Using this setup, we carry out the experiments described below.

### B. Performance in cloudy winter days

The fundamental difference between active and passive screens is the backlight source. During the day, sunlight can range from one thousand Lux on a very cloudy day to 100 kLux on a clear day. With normal sunlight, our modified passive screen and the smart window work well. However, it is important to assess how different passive screens perform under unfavorable lighting conditions.

To have a controlled scenario, we use offline decoding during a *cloudy winter day*. In these tests, we encode a single QR code in a video with different levels of visibility. We sweep the $\Delta E_{00}$ value in Equation 1 from 1.0 to 3.0 in steps of 0.2, yielding different $\Delta L^*$ to embed the QR codes. This video is displayed on the three monitors explained above, and the results are discussed next.

*1) Active display:* First, we use the *active* monitor as a benchmark with a frame rate of 60 FPS. A camera is installed on a tripod at a distance of 50 cm from the monitor capturing videos at 60 FPS. For each transmitted video, we set the camera to work with four different combinations of ISO and shutter speeds. Then, the decoding pipeline mentioned in Section III-A is used to calculate the PSR. Fig. 6a shows the link's reliability (PSR) as a function of modulation depth ($\Delta E_{00}$).

The four camera settings are denoted by the tuple (ISO, shutter speed), with the shutter speed being in seconds. Among the four combinations, (50,1/90) results in the best PSR, which is about 80% for $\Delta E_{00}$ values above 2.0. However, if $\Delta E_{00}$ is above 1.8, the encoding becomes gradually visible to people. The problem is that if we use this lower value of $\Delta E_{00}$, the PSR drops to about 50%. Hence, overall, with an active screen at 60 FPS, we get around 80% PSR but causing some flicker.

*2) Smart window:* For this step, the videos are the same but the smart window is bigger than the active display. To capture the whole monitor, we place the camera at a distance of 1.5 m. The smart window has a maximum frame rate of 30 FPS, therefore, we also capture videos at 30 FPS. The results are shown in Fig. 6b. In these experiments, only one (ISO, shutter speed) setting worked reliably. The results are better than with the active monitor, around a 90% PSR with a $\Delta E_{00}$ above 2.0. This improvement is due to the lower frame rate since it ameliorates the rolling shutter effect. Overall, despite the low sunlight conditions, the active screen and smart window provide comparable performance. This is thanks to the smart window's emphasis on reducing attenuation but at the cost of working only in black-and-white.

*3) Modified screen:* The active display and the smart window could transmit data packets with various PSRs, as discussed earlier. However, our modified screen could not operate reliably because it attenuates more light than the smart window, as mentioned in Section II. This low performance, however, is only for very low lighting conditions, the demo in the videos is done with our modified screen under normal sunlight conditions. The advantage of our modified screen is that it is obtainable from any monitor.

*C. Real-time decoding*

In Section I, we mentioned a sample application for passive screens that synchronizes the audio of a smartphone to a displayed video. In this section, we develop a real-time Android application using the decoding techniques described in Section III and benchmark its performance. Unlike the offline decoding used in the prior evaluation, a real-time system must have low response times to keep the user engaged. According to studies in this area [9], the maximum allowable response times and their interpretations are listed in Table I. These thresholds are marked with vertical red dashed lines on all graphs in this section. To measure the response times, we calculate the Cumulative Distribution Function (CDF) of successfully decoding a packet within a given time. The response time has a positive correlation with reliability (PSR), i.e. a link of better quality has a shorter decoding time. However, this depends on the phone's processing capabilities and the parameters used in data embedding. The phone we use is a medium-end device (Xiaomi Redmi Note 10 4G) and the Android application is optimized with multithreading to take advantage of the processing power.

In this section, we run the tests on our modified display to provide an upper bound of response times, since the smart

| Response time | User's perception of the system |
|---|---|
| $< 300$ ms | Instantaneous |
| 300 ms to 1 s | Immediate |
| 1 to 5 s | Transient, not immediate, but the user does not disengage from the activity. |

TABLE I: User's experience thresholds according to [9]

window performs even better, as described earlier. The encoding parameters we control and their effect on the system's performance are mentioned below.

*1) The impact of $\Delta E_{00}$:* This value determines how perceptible the encoded data is. We sweep this parameter from a low value of 1 to a large value (most visible) of 3 in steps of 0.5. Then we measure the cumulative distribution of the response times. The results are shown in Fig. 7b. Interestingly, the CDF curves for $\Delta E_{00} \in \{2, 2.5, 3\}$ are similar. There is a 95% chance of successfully decoding packets within 265 ms, 269 ms, and 267 ms, respectively. However, when $\Delta E_{00}$ is decreased to 1.5, the response time increases to 400 ms to have the same 95% success. Furthermore, for $\Delta E_{00} = 1$, the response time further escalated to 1305 ms. Although increasing the $\Delta E_{00}$ results in a shorter response time, it also makes the QR codes more visible. In Section IV-E, we perform a survey on visibility, where we show that with $\Delta E_{00} = 1.5$, the users *only* perceive a slight improvement compared to $\Delta E_{00} = 2.0$. Therefore, considering that the response time is substantially better in the latter case, we opt for a $\Delta E_{00}$ value of 2.0 in the rest of the experiments.

*2) Redundancy of embedded data:* A single QR code occupies a small space and could leave large regions of a frame unused. By duplicating the QR code, the unused space can be utilized in favor of redundancy. To investigate this approach, the performance of a single QR code is compared with that of a video containing six QR codes copied all over the frame. As shown in Figure 7a, the CDF of a single QR code has a probability of 95% at a response time of 805 ms, while the link with 6 QR codes has the same reliability at 267 ms. This is a substantial improvement. As a result, we use the version with 6 QR codes in the rest of the experiments despite being more noticeable to a user. In our survey, we will show that the slight increase in visibility is a valuable trade-off to achieve a considerably lower response time.

*3) Hand motion filter:* To verify the effect of hand motion on the response time, we do three experiments: (i) the phone is placed on a tripod and the filter is not active, (ii) the phone is held by hand and the filter is still not enabled, and (iii) the phone is held by a user and the hand motion filter is enabled. The results are shown in Figure 7c. In experiment (i), the CDF reaches a probability of 95% at 267 ms. In experiment (ii), the same success probability is achieved at 1737 ms, a substantially longer period. This can be attributed to unaligned frames, causing the phone to struggle to decode. Finally, in experiment (iii), a remarkable improvement is seen, with a CDF of 95% at 372 ms. This represents a significant enhancement compared to the configuration without the hand motion filter and is only slightly inferior to the static setup.
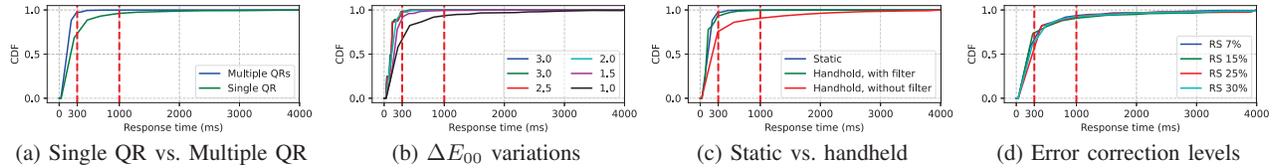
| (a) Single QR vs. Multiple QR | (b) $\Delta E_{00}$ variations | (c) Static vs. handheld | (d) Error correction levels |

Fig. 7: Response times for different implementation methods. The thresholds of Table I are marked with red dashed lines.



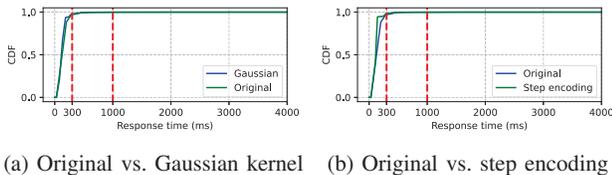| (a) Original vs. Gaussian kernel | (b) Original vs. step encoding |

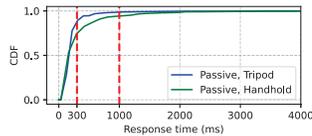Fig. 8: Response times of different encoding methods.



Fig. 9: Response time of final system on passive display

*4) QR error correction level:* When investigating the effect of $\Delta E_{00}$, we mentioned that a $\Delta E_{00} = 2$ satisfies the real-time requirements but it is not invisible. On the other hand, a $\Delta E_{00} = 1$ performs poorly in terms of response time while making the codes almost invisible. To investigate this trade-off further, we look into the inherent Reed-Solomon (RS) error correction of QR codes. We hypothesize that by increasing the error correction, the *unrecoverable* codes with a $\Delta E_{00}$ of 1 might become *detectable*, resulting in a lower response time.

In this regard, we set the $\Delta E_{00}$ to 1, and use four different RS levels on the QR codes. The CDFs of these experiments are shown in Fig. 7d. We can see that none of the curves provide good performance according to the real-time classification of Table I. For a CDF of 85%, the response times with different error correction levels are: 7% RS error correction yields a response time of 532 ms, 15% results in 545 ms, 25% gives 528 ms, and 30% results in 530 ms. Contrary to the hypothesis, all error correction levels demonstrate comparable performance. Considering these results, it is better to use a QR code with 7% RS error correction as it occupies less space and makes room to place more data.

*5) Gaussian kernel:* Applying a Gaussian kernel to the binary frames makes the encoded data less visible. To verify its effect on response time, we compare two cases with and without a kernel. The results are shown in Fig. 8a. Without a Gaussian kernel, the CDF reaches 95% at 266 ms. With the kernel, the response time does not vary and is 267 ms. Hence, the Gaussian kernel is harmless to the response time while it reduces the visual effects of the encoding.

*6) Step encoding:* The CDF of the response times for transmitters with and without step encoding is shown in Figure 8b. When step encoding is enabled, a CDF of 95% is reached at 244 ms. When disabled, it gives a response time

of 267 ms. Hence, this enhanced encoding does not affect the response time much and is safe to use.

*D. Final implementation*

Our final implementation uses the results from the last sections. The video embeds 6 QR codes with an error correction of 7%, the $\Delta E_{00}$ is set to 2.0, and a Gaussian filter is applied. The real-time pipeline is tested with a passive screen and a camera in two conditions: hand-held and on a tripod. These results are shown in Fig. 9. Based on the measured CDF, there is a 95% chance of decoding packets successfully within 530 ms using a tripod and 1071 ms when held by hand. According to [9], these response times are both classified within or very close to the *immediate* response category

*E. Survey results*

Although we propose methods to hide the encoding, visual distortions are inevitable since videos are modified. To evaluate modifications, we perform a survey where we place two videos side-by-side: one with and the other without the intended modification. Then, we asked a total of 20 users to compare the flicker of the two videos by choosing one out of five levels. The results are shown in the box plots of Fig. 10. In these experiments, we consider the following four cases:
**a) Apply a $\Delta E_{00}$ of 1.5 or 2?** As mentioned in Section IV-C, there is a significant advantage in the response time when we move from a $\Delta E_{00}$ of 1.5 to 2. It is important to assess the user perception as well. As shown in Fig. 10a, the median of users said there is "slightly less flicker" for the lower value of $\Delta E_{00}$. Considering this minor effect, and the big advantage in the response time of $\Delta E_{00} = 2$, we opt for this higher value.
**b) Apply Gaussian kernel or not?** Following Fig. 10b, applying a Gaussian kernel helps reduce flicker.
**c) Apply step encoding or not?** Following Fig. 10c, users see a better performance when step encoding is applied.
**d) Use single or multiple QR codes?** Using more QR codes occupies more space in a video, which can cause more flicker as validated by the survey in Figure 10d. However, this flicker is not significant. Considering its lower response time, the redundancy of QRs is worth implementing.

*F. PassiveCam in action*

In Section I, we mentioned that passive screens are being deployed more and more in public places. This opens opportunities for new applications. Together with members of a company, we propose the development of a real-time link for the following application: when a video is being publicly displayed, it is usually difficult for a user to hear quality audio.

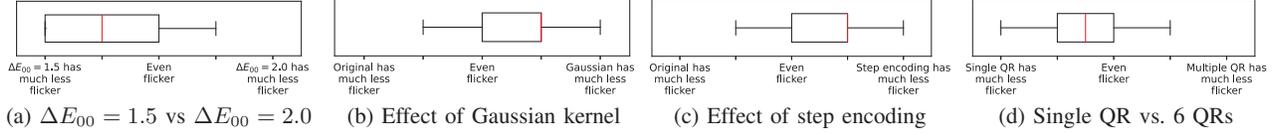| (a) $\Delta E_{00} = 1.5$ vs $\Delta E_{00} = 2.0$ | (b) Effect of Gaussian kernel | (c) Effect of step encoding | (d) Single QR vs. 6 QRs |

Fig. 10: Survey results comparing flicker for different parameters and encoding techniques.

Therefore, we use a phone to synchronize the video so users hear the video's sound on their phone. To make this system, we encode synchronization packets–i.e. frame numbers of the video–unobtrusively using the methods discussed before. Then, with our real-time application, the user's phone can read the hidden messages and play the audio accordingly. A demo of this application for a black-and-white and a color video has been put in the links of [7] and [8]. More details of this implementation are described in Appendix VIII-B.

## V. RELATED WORK

**Unobtrusive screen-to-camera communication.** Numerous studies have explored imperceptible data encoding in videos [1]–[5], [10]–[12]. The pioneering works in this area are InFrame [3] and its subsequent improvement InFrame++ [4]. They are the first to exploit the flicker fusion property of human eyes with complementary frames. However, Texture-Code [5] identified that some flicker was present in those studies. The flicker appeared in smooth areas, hence, TextureCode proposed texture analysis to selectively pick the regions where data should be embedded. The latter improves invisibility but it sacrifices throughput, as it only uses areas with rough texture. To maintain invisibility while leveraging the entire frame, ChromaCode combines luminance with texture analysis [2]. Instead of using a uniform delta value to modulate the entire frame, ChromaCode adjusts the delta based on the expected human perception of color and texture. AirCode [1] employs the same video encoding as ChromaCode with the addition of an audio channel transmitting metadata, which compensates for the more error-prone screen channel. Given ChromaCode advancements and the fact that it uses lightness modulation instead of color, we used it as the basis of our work.

Other papers explore color domains to achieve invisibility. Deeplight [10] applies the delta value only to the blue channel as human eyes are less sensitive to blue light. Uber-In-Light uses both the blue and red channels to embed data [11], modifying the green channel just to compensate for the visual distortions. Other studies focus on chromaticity, considering that the maximum perceptible chromatic flicker occurs at approximately 25 Hz [12]. These color based studies are valuable, however, not applicable to black-and-white smart windows. Overall, compared to all the above studies, our work the first to design a system for passive screens. Furthermore, our system works in real-time with low-end smartphones.

**Passive communication with light.** Another area that inspired our work is one using optical devices to modulate ambient light [13]–[18]. These platforms can work with photodiodes and cameras but all of them focus on point-to-point links. To our knowledge, the only paper that moves towards passive screens is Sunbox [19]. This work employs a tiny microdisplay (mm in size) to transmit QR codes to a smartphone via ambient light reflections. However, there is no video transmission, only QR codes, and the phone has to be placed statically on a holder at a distance of three centimeters from the microdisplay.

## VI. CONCLUSION

In this paper, we explored a novel screen-to-camera communication link by using passive displays. Besides, by improving on top of the SoA, we encoded data unobtrusively in a video, which was then displayed by our passive and active screens. We showed that by using a smart phone, this encoded data could be extracted. As a proof-of-concept, we tested our custom Android application in a real-time audio synchronization scenario. Additionally, we assessed the performance of this system both in real-time and offline, based on several factors and user surveys. Overall, our work is the first to make a step towards new opportunities in *passive* screen-to-camera communication domain, paving the way for pervasive applications such as smart surfaces and in-video advertising.

## VII. ACKNOWLEDGEMENTS

## VIII. ADDITIONAL REMARKS

### A. Optimization of texture analysis

In prior literature, regional texture analysis is implemented by constructing a GLCM for every pixel. GLCM is a known method that quantifies the *texture* inside an $H \times W$ window around a pixel location (x,y) of the image. H and W are arbitrary numbers denoting the dimensions of the chosen window. The resulting GLCM has a dimension of $N_g \times N_g$, where $N_g$ is the number of distinct grey levels found in the image. Therefore, it is possible to obtain large GLCM matrices depending on the context of an image. Larger matrices have negative effects on calculations in two directions: (1) They consume time to calculate and access and (2) they take up space in the memory. To optimize for these issues, we simplify the calculations as mentioned below.

Let us denote an element of the GLCM matrix by $p_{(x,y)}(i, j)$, where (x,y) is the location of the pixel in the original image and the tuple (i,j) is the index of an element

within the GLCM matrix of that pixel. Following the calculation procedure of GLCM [2], each $p_{(x,y)}(i,j)$ can be written as follows:

$$p_{(x,y)}(i,j) = \frac{1}{R_{(x,y)}} \sum_{r=0}^{H-1} \sum_{c=0}^{W-2} V(c,r) \quad (5)$$

where $V = \begin{cases} 1, & \text{if } L^*(c,r) = i \text{ and } L^*(c+1,r) = j \\ 0, & \text{otherwise} \end{cases}$

where $H$ and $W$ are the respective height and width of the arbitrary window around the pixel (x,y), and $R_{(x,y)}$ is the total number of horizontal neighboring in that window. In the summation, the index pair (c,r) sweeps over all pixels in the arbitrary window and sets the value of $V$ to 1 only if the pixel at (c,r) has the lightness of i and its right neighbour has the lightness j. In short, the element (i,j) of the GLCM matrix counts pixel pairs in the arbitrary window such that the left one has the lightness of i and its immediate right neighbour has the lightness of j.

After this, the contrast equation of the pixel at (x,y) reads as follows, assuming that $H = W = n$:

$$C(x,y) = \sum_{n=0}^{N_g-1} n^2 \{\sum_{i=0}^{N_g} \sum_{j=0}^{N_g} p_{(x,y)}(i,j)\}, \quad \text{s.t.} |i-j| = n \quad (6)$$

Substituting Equation 5 in Equation 6 gives:

$$C(x,y) = \sum_{n=0}^{N_g-1} n^2 \{\sum_{i=0}^{N_g} \sum_{j=0}^{N_g} \frac{1}{R_{(x,y)}} \sum_{r=0}^{H-1} \sum_{c=0}^{W-2} V(c,r)\} \quad (7)$$

Moving the $\frac{1}{R_{(x,y)}}$ factor out of the summation and rearranging the terms will yield:

$$C(x,y) = \frac{1}{R_{(x,y)}} \sum_{n=0}^{N_g-1} n^2 \{\sum_{r=0}^{H-1} \sum_{c=0}^{W-2} \sum_{i=0}^{N_g} \sum_{j=0}^{N_g} V\} \quad (8)$$

Now the part $\sum_{i=0}^{N_g} \sum_{j=0}^{N_g} V$ will either be 1 if $L^*(c,r) = i$ and $L^*(c+1,r) = j$ or 0 otherwise. Therefore the equation in (8) can be simplified to:

$$C(x,y) = \frac{1}{R_{(x,y)}} \sum_{r=0}^{H-1} \sum_{r=0}^{W-2} \sum_{n=0}^{N_g-1} n^2 \quad (9)$$

$$\text{s.t. } |L^*(c,r) - L^*(c+1,r)| = n.$$

Substituting for the value of $n$ in Equation 9 will result:

$$C(x,y) = \frac{1}{R_{(x,y)}} \sum_{r=0}^{H-1} \sum_{r=0}^{W-2} (L^*(c,r) - L^*(c+1,r))^2 \quad (10)$$

Using Equation 10, we can bypass creating a GLCM for each pixel to reduce the computational time significantly. When benchmarked on a Graphics Processing Unit (GPU), the computational time of the optimized method was 40 ms for two complementary frames, whereas it was 20190 ms using the original method of formulating a GLCM for each pixel, yielding a speed-up of more than 500×.

### B. Implementation details of the music application

In this appendix, we describe the details of our application: a user points his smartphone camera towards the passive screen while a video is playing. The phone is then triggered to play the audio synchronized with the video. In our example we use different songs including the video clip "Coldplay - The Scientist". The embedded QR codes contain a song identifier and its frame's index. This information is represented by a JSON string in the form of $\{'s':s_{id}, 'f':f_{num}\}$, where the $s_{id}$ is an integer representing the song, and $f_{num}$ is the frame index of the video. Each song is saved on the smartphone prior to the experiment. With the $s_{id}$, the application can select the correct song, and with the frame number, it calculates the current time of the video. To synchronize the audio with the video, the decoded frame number is converted to an absolute time value in milliseconds ($t_{abs}$) using the following equation:

$$t_{abs} = \frac{2f_{num}}{FPS_{TX}} \cdot 1000 \quad (11)$$

where $FPS_{TX}$ is the screen's frame rate. Finally, the application must compensate for the time taken to decode, which is measured using the internal clock of the phone, and is then added to $t_{abs}$ to have an estimate of the current video timestamp. After this, the phone starts playing the audio from this calculated time. Thus, the synchronization is achieved.

### REFERENCES

[1] K. Qian *et al.*, "Aircode: Hidden screen-camera communication on an invisible and inaudible dual channel," in *USENIX NSDI*, 2021.

[2] K. Zhang *et al.*, "Chromacode: A fully imperceptible screen-camera communication system." ACM Mobicom, 2018.

[3] A. Wang et al., "Inframe: Multiflexing full-frame visible communication channel for humans and devices," in *ACM HotNets*, 2014.

[4] A. Wang *et al.*, "Inframe++: Achieve simultaneous screen-human viewing and hidden screen-camera communication," in *ACM MobiSys*, 2015.

[5] V. Nguyen *et al.*, "High-rate flicker-free screen-camera communication with spatially adaptive embedding," in *IEEE INFOCOM*, 2016.

[6] "Videowindow," https://www.videowindow.eu/, last accessed: March 11, 2024.

[7] "Demo, black and white," https://youtu.be/S_hxk3TQCmI, 2023.

[8] "Demo, colored," https://youtu.be/7FoaacEiG60, 2023.

[9] R. Doherty *et al.*, "Keeping users in the flow: mapping system responsiveness with user experience," *Elsevier Procedia Manufacturing*, 2015.

[10] V. Tran *et al.*, "Deeplight: Robust & unobtrusive real-time screen-camera communication for real-world displays," in *IEEE/ACM IPSN*, 2021.

[11] M. Izz *et al.*, "Uber-in-light: Unobtrusive visible light communication leveraging complementary color channel," in *IEEE INFOCOM*, 2016.

[12] S. Abe *et al.*, "Imperceptible color vibration for screen-camera communication via 2d binary pattern," in *ITE Transactions on Media Technology and Applications*, 2020.

[13] R. Bloom *et al.*, "Luxlink: Creating a wireless link from ambient light," in *ACM SenSys*, 2019.

[14] X. Xu *et al.*, "Enabling practical visible light backscatter communication for battery-free iot applications," in *ACM Mobicom*, 2017.

[15] S. Ammar *et al.*, "Sun-fi: Architecting glass for sunlight data transmission," *IEEE Communications Magazine*, 2023.

[16] Z. Yang *et al.*, "Wearables can afford: Light-weight indoor positioning with visible light," in *ACM MobiSys*, 2015.

[17] J. Li, *et al*, "Retro-vlc: Enabling battery-free duplex visible light communication for mobile and iot applications," in *HotMobile*, 2015.

[18] S. K. Ghiasi *et al.*, "A principled design for passive light communication," in *ACM Mobicom*, 2021.

[19] M. Chávez Tapia *et al.*, "Sunbox: Screen-to-camera communication with ambient light," in *ACM IMWUT*, 2022.