High-Rate Flicker-Free Screen-Camera Communication with Spatially Adaptive Embedding

Paper # 1570204645

***Abstract*—Embedded screen-to-camera communication techniques encode information in screen imagery that can be decoded with a camera receiver yet remains unobtrusive to the human observer. These techniques have applications in tagging content on screens similar to QR-code tagging for other objects.**

This paper characterizes the design space for flicker-free embedded communication. In particular, We identify an orthogonal dimension to previous techniques: spatial content-adaptive encoding, and observe that it is essential to combine all dimensions to achieve both high capacity and minimal flicker. From these insights, We develop content-adaptive encoding techniques that exploit visual features such as edges and texture to unobtrusively communicate information. These can then be layered over existing techniques to further boost the capacity. Our experimental results show that we can achieve a mean goodput of about 15 kbps, significantly outperforming existing work while remaining flicker-free.

I. Introduction

With the pervasive use of screens and cameras, screen-to-camera communication through QR-code-like tags has emerged in diverse applications from pairing devices to obtaining context from advertisements and other screen content. When placing such codes on screens, they occupy valuable screen real estate, which results in undesirable compromises. Either the visual code replaces most of the imagery on the screen, which usually distracts from the aesthetics of the image or video, or the code only uses a small area of the screen, which leads to less throughput and requires the camera receiver to be closer to the screen. This conundrum motivates embedding such codes into the screen imagery so that the code is detectable with camera receivers but imperceptible for the human visual system.

While there have been a few existing efforts on embedded screen-camera communications, they tend to achieve *either* high throughput but noticeable flicker *or* virtually flicker-free embedding but low throughput, as illustrated in Fig. 1. In particular, InFrame++ [23] utilizes the flicker fusion property of the human visual system to embed data. It relies on high screen refresh and camera frame rates to modulate the image at rates faster than the human eye can perceive. It can therefore transmit data at 18 kbps, but noticeable flicker remains. HiLight [21] modulates bits through slight pixel translucency changes, which reduces flicker to unnoticable levels but only supports a low bit rate. The setting is also related to the classic watermarking literature, but only some of the work considers camera capture, usually involving ultra-low data rates of a few bits per second. These are sufficient for digital rights management applications to address movie piracy, but do not meet the capacity and flicker requirements of pervasive screen

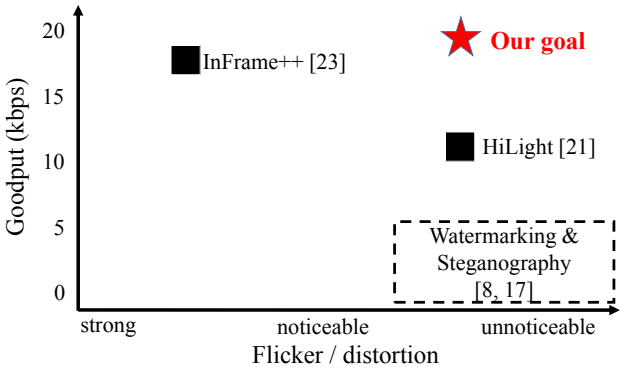


Fig. 1: State-of-the-art methods for screen-to-camera communication compromise either visual obtrusion or goodput. TextureCode selectively embeds in textured regions of images to reduce flicker, but still achieves superior goodput.

tags. Overall, existing work tend to each explore one technique for embedding, and it is unclear what the limitations are.

In this paper, we systematically explore psychovisual factors leading to flicker perception and uncover additional dimensions of the flicker-free embedding design space. In particular, we study adaptive spatial encoding in the screen-camera communication channel, which has hitherto remained unexplored.

Conceptually, spatially adaptive encoding in screen-camera systems resembles adapting modulation and coding rates on different streams in a spatially multiplexed precoded Multiple Input Multiple Output (MIMO) radio system. In practice, however, the screen-camera communication channel imposes very different challenges. Radio frequence MIMO often requires precoding because typical MIMO spatial streams interfere with one another and need to be decorrelated for the best encoding opportunities and decoding performance. In the case of screen-camera communication, or visual MIMO, however, the individual pixel-to-pixel links are very directional, and there is little interference between such "spatial" links (unless there is undersampling). Instead, the primary challenge is that the modulation and coding techniques should not only maximize communication performance but also minimize image distortions and flicker for the human observer. In addition, the communication techniques must be robust to noise from the carrier image or video and work without feedback from the receiver, since the screen-camera channel is a one-way channel.

Our work addresses the challenges by exploring several

factors for flicker perception and combining corresponding coding opportunities. First, since both flicker perception and receiver noise depend on the visual content of the frame that the information is embedded in, we design a texture-based estimator that determines the suitability for embedding in each pixel block of the screen. This information then governs the choice of modulation and lends to the spatially adaptive approach. It also addresses the unknown channel state at the transmitter, since the texture analysis effectively provides an estimate of receiver noise on each block. Second, the technique aligns the boundary of each encoded region along the existing edges in the video sequences to minimize the visible artifacts caused by encoded messages. Third, akin to earlier work, we also modulate at a rate beyond the critical flicker fusion threshold for most observers but remains decodable with the high-frame rate (slow motion) cameras available in today's smartphones. Finally, we identify a lightweight approach following the same principles and delivering similar performance at a much lower computational complexity.

In summary, the salient contributions of this work are:

- We analyze factors contributing to distortions and the flicker perception of embedded screen-camera communication.
- We identify techniques to achieve spatially and content adaptive embedding. Further, it is possible to achieve similar performance using a lightweight approach.
- We explore and combine multiple encoding methods to embed information into arbitrary video content without noticeable distortions or flicker.
- We show through experiments that such methods have the potential to more than double the goodput of existing flicker-free screen-camera communication techniques.

II. Flicker perception for embedded communication

Flicker is a perceptual attribute normally defined for displays, seen as an apparent fluctuation in the brightness of the display surface [6]. Prior psycho-visual studies have revealed various effects in the displayed video that may contribute to the perceivable flicker, such as the frame rate, image content, saccades, and the viewer's field of view.

By inducing brightness changes in a regular video to modulate bits, embedded screen-camera communication can naturally generate flicker. Therefore, we explore how to balance the conflicting goals of embedding bits and avoiding flicker. Where applicable, we perform simple experiments to provide qualitative hints. These follow the same settings as in Section V, and the flicker level is assessed visually by the first two authors.

A. Frame rate

It has long been known that flicker perception is prominent for luminance fluctuations below 100 Hz [13], [18]. Although this frequency threshold was determined using a single light source, it is still applicable if we consider the modern display as a collection of LED light sources.

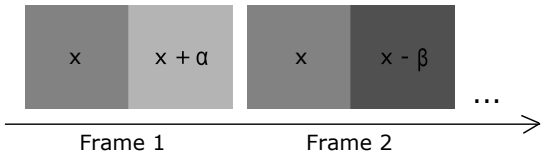


Fig. 2: Signal amplitude experiment.

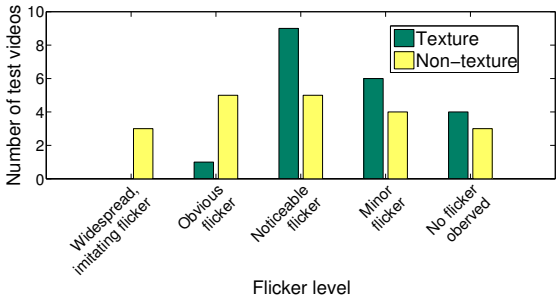


Fig. 3: Performance of flicker perception for different video samples

In our case, the fluctuation is caused by switching between bits at the same position of the video across frames. Since such bit streams are random, we are constrained by the largest differences between the codewords, the available display refresh rate (up to 144 Hz), and the camera capture rate (up to 240 fps). Given the latter two constraints, we can expect to display at 120 fps. The maximum codeword distance can then be determined accordingly.

We place two uniform grayscale blocks side by side (Fig. 2). In each run, the left block has a fixed intensity value x, while the right block's color flips between $x+\alpha$ and $x-\beta$ at 120 fps. Across runs, x varies from 0 to 255 at steps of 25. Experiments show that the color deviation without inducing flicker perception is α = 2 and β = 3. In other words, only very slight color differences between adjacent blocks can be tolerated. This suggests very limited scope for encoding bits directly using pixel intensity changes.

B. Image content

Images of natural scenes often contain many textured regions that we can use in our coding method. It is well known that human vision is sensitive to even small intensity edges [5], [10] and that texture affects the perception of intensity transitions [9]. As a practical consequence of these perception traits, intensity modifications in smooth regions are more likely to cause flicker than textured regions. To take advantage of this flicker reduction, our method adapts to image content by detecting textured regions and embedding message bits within this space.

To qualitatively evaluate the intuition of texture-based embedding, we experiment with 20 videos of varied content. We divide each video frame into smooth and textured regions (detailed in Section III), and embed bits into the smooth regions only, the textured regions only, or all regions to

compare the flicker perception. Fig. 3 shows that embedding into textured regions exhibits the least amount of flicker.

C. Saccades

Saccades are rapid, ballistic movements of the eyes that abruptly change the point of fixation. In [12], the authors introduced an edge between a white half frame and a black half frame. The colors of the two halves were inverted in rapid succession, and the human subjects still observed flicker artifacts regardless of the switching frequency, even at 500 Hz.

Since it is common to use a block of pixels to encode a bit, we also encounter edges between adjacent blocks of different bits. When the two neighboring blocks are modulated with "different phases", i.e., one block changes from $x+\alpha$ to $x-\beta$ while the other changes in reverse, flicker is noticeable. However, separating the blocks with some distance can reduce or minimize the effect.

D. Viewer's field of view

In the course of experiments, we also observe that the level of flicker perception depends on the size of the encoded regions in the video and the distance of the viewer from the video displayed. We capture both effects with a single metric, the size of the "viewer's field of view". To measure this size, we use a square block of different sizes for encoding without changing other parameters and view the video from difference distances. Results show the smaller area fell into viewer's retina, i.e., the smaller block size or further distance, the less flicker the viewer perceives. This suggests using only small code blocks for encoding and avoiding parts of the image scene that might attract attention.

E. Hints for code design

We make several observations from the exploration so far. First, the first three factors above suggest opportunities for modulating bits, while the field of view cannot be leveraged easily, since the encoder side has no control. Second, each factor alone offers limited flexibility in modulation. In other words, to control flicker perception in the encoding process, we have to work within a small range of brightness fluctuation, which significantly constrains the code capacity. This is precisely why HiLight and Inframe++ *either* achieves a high goodput *or* negligible flicker perception, but not both simultaneously. Third, the first three factors are orthogonal, paving way for combining the corresponding techniques leveraging the factors. Frame rate is a temporal property of the video, whereas the image content and saccades mostly affect the spatial domain.

Based on these insights, we design TextureCode to achieve high capacity at negligible flicker.

III. Spatial-Temporal Embedding

We exploit these observations of flicker perception and explore schemes that operate both in the spatial and temporal dimensions. We first discuss the temporal dimension through the design of high-frame rate embedding that seeks to operate beyond the human flicker fusion frequency. We then discuss schemes that employ spatial adaptation based on texture analysis to address the image content factor. Finally, we align the boundary of each encoded region along the existing edges in the images, to minimize the effect of visible artifacts caused by encoded messages and address Saccades. This is accomplished through a superpixel encoding technique. We refer to combining these ideas in an approach that we call TextureCode. A block diagram of this approach is shown in Figure 4.

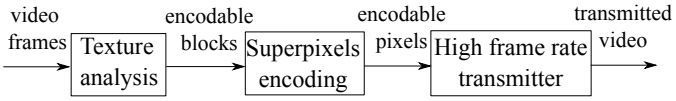


Fig. 4: The block diagram of the different components in TextureCode.

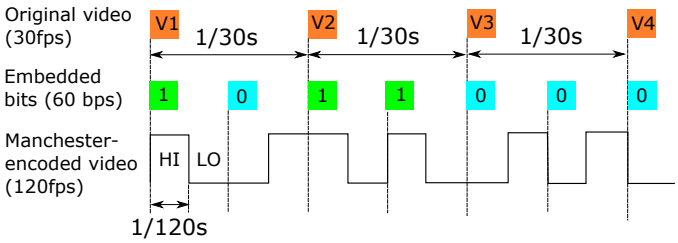


Fig. 5: Illustration of the encoding method.

A. Temporal embedding

We apply basic temporal embedding as follows. In our system, we utilize a screen capable of playing video at high speed (120 Hz) as our transmitter. The 120 fps video is created from an original video at 30 fps by duplicating each frame in the original videos to 4 new frames. These 4 new frames are then used to embed messages.

Figure 5 shows how we embed messages inside a video sequence. For each frame, the message structure is a rectangular M×N grid where each grid block carries one bit of information. We choose a Manchester-like encoding scheme for modulating bits, to keep the frequency components of the encoded video signal above 60 Hz. Since Manchester encoding ensures a transition on every bit, it generates less low frequency components in the modulated signal when multiple consecutive bits are identical. For example, a block with bit 0 encoded, its luminance will be denoted as LOW-HIGH in two consecutive frames. For a block with bit 1 encoded, its luminance will be denoted as HIGH-LOW in two consecutive frames. This modulation signal is then combined with the sequence of carrier image frames. The carrier pixel values inside each HIGH block are increased by α, which means these pixels are made brighter. The pixel values inside each LOW block are decreased by β, which is equivalent to making these pixels darker. In our implementation, the two values are chosen as $\alpha = 2, \beta = 3$. This change is applied to the Y channel in a YUV encoded frame.

B. Spatial embedding based on texture analysis

The effect of message-hiding to human eyes is not universal across the video sequence. We observe that in some regions, especially in regions having no or little texture, the flicker is more obvious to see. This becomes motivation for us to use texture analysis to select "good" regions to embed in the video sequence.

In particular, we seek to categorize the blocks inside each video sequence as "good" or "bad" based on its flicker effect when being encoded by Manchester coding described above. We propose two techniques for this task: one based on a machine learning technique called **texton analysis** and the other one is **pixel-based texture analysis** method. The former is the more accurate and complete technique for identifying "good" and "bad" blocks, but it is computationally heavy. Therefore, although the technique allows us to explore to what extent of data throughput we can achieve with our texture analysis, for dynamic scene videos, we employ the second simpler method.

Texton analysis For texture analysis, we employ texture classification based on textons [19], [20], [11]. The algorithm is divided into a learning stage and a classification stage. In the learning stage, training blocks are convolved with a filter bank to generate filter responses as shown in Figure 6. Exemplar filter responses are chosen as textons via K-mean clustering and are collected into a dictionary. After learning a texton dictionary, we model texture as a distribution of textons. Given a block in a video frame, we first convolve it with a filter bank and then label each filter response with the closest texton in the dictionary. The histogram of textons, which is the count of each texton occurring in the labeling, provides us a model corresponding to the training block.

Next, we continue use K-mean clustering to divide our training set of texton histograms into groups. For each group of texton histogram, we segment videos so that only blocks belong to that group are encoded. The videos are then graded based on their level of flickers. Then, each texton histogram group is labeled "good" if the videos have low flicker , and "bad" otherwise. In this manner, we identify the type of texture that is amenable to message embedding.

Each new block of an input video is pre-processed to compare its texton histogram with our training set of texton histograms to find its label ("good" or "bad"). Based on this label, the block is either used for message embedding or not.

Pixel-based texture analysis Texton analysis is a computationally intensive process and becomes more challenging to use in the dynamic scene videos as the varying content on each frame will require recomputing the "good" blocks to encode. To address this issue, we also propose a computationally efficient method to find the "good" regions to encode. This pixel-based texture analysis is based on the variations of spatial pixel intensities. A larger variation value indicates a more textured region.

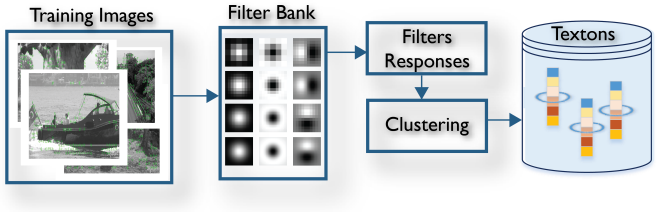


Fig. 6: Texton Method of Representing Textured Regions

C. Superpixels

In above sections, we described **the Manchester encoding** at 120 fps, which ensures the frequency component in our temporal video signal are above critical frequency threshold, and **texton analysis**, which excludes the region with the kind of textures that are likely to cause flicker. The flicker, although significantly reduced, is still observable. We observed that the flicker artifacts appear along the edges between checkerboard blocks. This is the results of the phenomenon described in section II, where two neighboring blocks modulated at different phases would cause flicker artifacts at their edges, even at frame rate as high as 120 fps. In addition, these edges are not naturally aligned to existing edges in original video contents, causing visible flickers to human eyes. From these observations, we are motivated to seek another technique to improve the unobtrusiveness of the encoded videos. This technique would ensure: (1) Separate the encoded regions (i.e., get rid of edges between them), and (2) align the border of each encoded region to the existing edges in the original frame.

The technique we chose to fulfill this requirement is **superpixels**, a computer vision technique that provides a convenient primitive from which to compute local image features. It belongs to over-segment techniques: an image is divided into sub-regions with respect to image edges, and pixels inside each region are uniform in color and texture. To generate superpixels, we use SLIC (Simple Linear Iterative Clustering) algorithm [4], which is fast and has high segmentation performance.

We use superpixels to determine which pixels inside checkerboard grid to embed information. Recall that in our scheme, pixels inside each block alternate between dark and bright intensity values. However, we observe that if all pixels inside each block are allowed to alternate, boundary between blocks are perceivable by human eyes. Superpixels, therefore, are employed to limit the region inside each block where pixels are allowed to alternate. This approach also allows boundaries of each encoded region aligns well with the real edges in the video frame, thus reduce significant perceivable flickers to human eyes.

Although superpixels can align the boundary of each encoded region to the existing edges of the original video frame, the receiver needs to know the location of each super pixel (i.e. which pixels in the original video frame belongs to which superpixel), which means it needs to rebuild the superpixel map for each video frame. The superpixels are also varying in

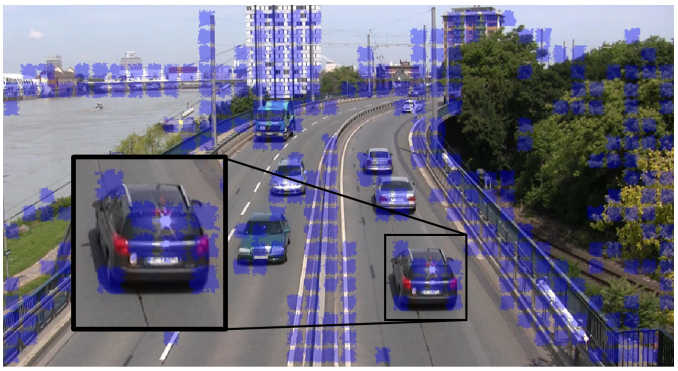


Fig. 7: Texture encoding method (the pixels inside the blue regions are encoded). The texton analysis avoids encoding in plain texture area in the video frame, such as the road, the sky, etc. while encoding in the high texture area, such as the cars, the building, etc. The superpixels method then further separates encoded blocks (avoid boundary effect), and also aligns their boundaries to the existing edges in the video frame.

size and shape, which can cause varying quality of decoding. To solve these problems for block-based decoding, we propose a **block-superpixel hybrid encoding method** as follows. Each video frame is first segmented into superpixels and also divided into checkerboard blocks. In each checkerboard block, we find superpixels that completely fall into that block, and mark pixels inside these superpixels to be encoded. These pixels are then alternated following the previously described method, while other pixels in the block are kept the same. Figure 7 shows an example of how pixels inside each block are chosen to be encoded. This hybrid encoding method has the following desirable properties: 1) it aligns the boundary of each encoded region to existing edges in each video frame; 2) it ensures there is no common edge between any two encoded units (blocks), and 3) it allows an easy block-based decoding method - there is no need to rebuild the superpixel map on the receiver side. This method has been proved to show good performance, both in flicker reduction and bit error rate, as will be illustrated in section V.

D. Receiver and decoder

The receiver in our system is a camera capable of capturing video at 240 Hz. To evaluate our decoding algorithm, we first captured high frame rate videos from the camera and then extracted all frames inside these videos for offline processing. The offline processing algorithm is implemented in Matlab.

1) Frame perspective correction and spatial block division: Because of the capturing angle and the camera distortion, the received video frames are normally trapezoids. This would bring some difficulties to the decoder when recovering the correct location of spatial blocks. Therefore, after extraction, all frames need to be warped into correct perspective. We use projective transformation for frame correction. After a frame has been corrected, it will be divided into blocks for later decoding process.

2) Decoding algorithm: The main challenge for the decoder is to extract the desired intensity change among the intensity changes due to noise and the video content itself. To minimize the effect of video contents to the decoder, we choose to decode 8 encoded frames from an original video frame. We are able to choose these 8 frames thanks to the following observation: the change by pixel modulation is relatively small compared to the change in video contents. Therefore, by calculating the pixel-by-pixel difference between consecutive frames, we can detect the starting point of each 8-frame group. The details of our decoding method are described in Algorithm 1.

Algorithm 1 Decoding algorithm

Input: a captured video.
Output: a decoded stream of bits.
Extract frames from the capture video.
for each frame in the extracted sequence of frames **do**
  Warp the frame into the correct perspective.
  Crop the video region inside the frame.
end for
Detect the starting point of each 8-frame group.
for every 8-frame group of the same content **do**
  for each block inside each frame **do**
    $a_1, a_2, ..., a_8$:= average intensities of all pixels inside this block in these 8 frames.
    for $i = 0, 1$ **do**
      if $a_{4i+1} + a_{4i+2} < a_{4i+3} + a_{4i+4}$ **then**
        $outBit = 0$
      else
        $outBit = 1$
      end if
      Save $outBit$ to the output buffer.
    end for
  end for
end for

IV. Implementation

The implementation of TextureCode consists of a transmitter and a receiver component. For the transmitter, we take an original image or video stream and a data bitstream as input, generate an YUV sequence, and use *glvideoplayer* [3] to play the video at 120 fps on a computer screen, whose refresh rate is set to 120 Hz. We choose an uncompressed YUV format to avoid any artifacts caused by video compression schemes. The receiver is a smartphone camera with high frame rate video recording capability. We chose the iPhone6 since it allows for 240 fps capture. It captures the video sequence displayed on the dipslay screen and detects the message embedded inside the video sequence.

Currently, both the transmitter and the receiver work offline. We use Matlab to multiplex the original video sequence with the data stream to create encoded version of the video. For the receiver, we use a Matlab script post-process the video file recorded on the the iPhone.

We implement two algorithms to evaluate TextureCode. One is the refined texton analysis and superpixels based method for finding "good" regions to encode. The other is more computationally efficient, using pixel-based texture analysis to find the "textured pixels" and encode only in those blocks with a high number of "textured pixels". For the latter method, encoded blocks are separated by small gaps to avoid flickers at the block boundaries. We use the first method for videos with static scene, as the texton analysis and superpixels are better at detecting regions with near-zero flicker perception. We use the lightweight pixel-based texture analysis for dynamic scene videos.

Since the objective is primarily to evaluate the limits of spatially adaptive embedding, we assume that the decoder knows the checkerboard size, the original video resolution and the encoded regions for each frame. This eliminates pixel offsets and error for texture analysis on the receiver side introduced from several factors, including video distortion, ambient light change and camera exposure setting. In a full protocol design, these parameters can be included in packet headers or inferred through additional receiver processing.

V. Evaluation

We experimentally evaluate the effectiveness of spatial-temporal embedding and its orthogonality to different schemes. In particular, we study the communication link performance of the TextureCode approach in terms of goodput and bit error rate and compare it with the existing HiLight [21] and InFrame++ [23] schemes as baselines.

Experiment Settings. We conducted experiments in a well-lit indoor office room environment using a display monitor screen as the transmitter and a smartphone camera as the receiver. We used an ASUS VG248QE 24-inch monitor to display a set of test videos at a rate of 120 Hz[1]. The screen resolution is 1360×760 while video resolution is 1280×720. The displayed videos were recorded as video streams at 240 fps with an iPhone6 using its built-in camera application in the *Slo-Mo* mode. The default distance between the screen and the camera was set to 70 cm, where the screen fills the camera image. The iPhone was mounted on a tripod as shown in Figure 8.

We selected a set of 10 videos from two publicly available standard data sets [1], [2]. Table I shows screenshots of sample test video sequences.

Metrics. The primary metrics for evaluation are bit error rate and goodput. We chose goodput over throughput since the bit error rates can be highly variable for embedded screen-camera communications. We define goodput as follows.

$$\text{Goodput} = \sum_{\text{all frames}} \frac{D}{t} \tag{1}$$

where D is the number of correctly decoded bits and t is the transmission time.

[1]the maximum refresh rate of the monitor is 144 Hz

In addition, we also consider the transmit rate to understand the effectiveness of the spatially adaptive embedding approach. Note that the transmit rate in TextureCode is dependent on the content of the carrier frames, because it encodes more bits in image areas that are conducive to embedding. The transmit rate therefore varies in TextureCode, while it remains constant in the baseline schemes.

$$\text{Transmit Rate} = \sum_{i=1}^{N} \frac{B_i \times b \times V}{N \times F} \tag{2}$$

where B_i is the total number of encoded blocks in frame number i, b is the number of bits encoded in each block, V is the video frame rate, N is the total number of frames in the video sequence, and F is the number of frames needed to encode one bit.

Schemes for Comparison. The two existing techniques can be summarized as follows. **HiLight** utilizes the alpha channel to encode messages into each frame. In each carrier frame, the alpha value is either 0 or $\Delta\alpha$, which is small (about 1-4%). The messages are embedded using Binary Frequency Shift Keying (BFSK), where 6 frames are used to encode bit 0 or 1 by translucency at 20 Hz or 30 Hz respectively. **InFrame++** uses Spatial-Temporal complementary frames (STCF) to design frame structures. In InFrame++, a *Cell* consists of p×p physical pixels; a *Block* consists of c×c neighboring Cells, and it is considered the basic information carrying unit. InFrame++'s design boosts data throughput by allowing each block to deliver multiple bits, distinguished by different visual patterns.

For each sample video, we generated a test video sequence each for the three candidate encoding schemes (TextureCode, HiLight and InFrame++) where each image frame of the test video was embedded with a random bit stream. While we used the original implementation of HiLight using the code provided by the authors, we implemented InFrame++ based on the description available in the paper [23], as we did not have access to the code.

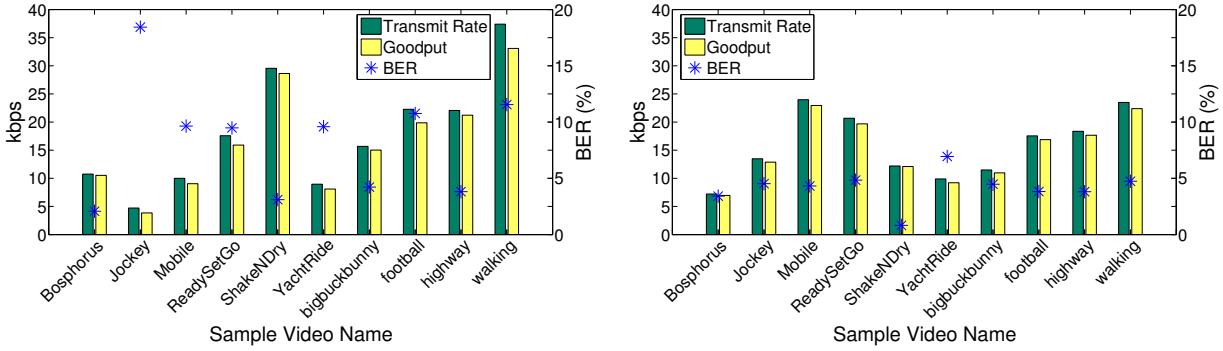
In addition, we implemented a *hybrid* encoding scheme where we used our proposed TextureCode technique and the HiLight scheme on different regions of each video frame. As we will show through our evaluations, the hybrid scheme improves the communication performance of HiLight while inducing no flicker.

A. Communication Performance of TextureCode

We evaluate the communication link performance of TextureCode for two use-cases: (i) *dynamic*, where the visual content (i.e. background) of the test video is changing, and (ii) *static*, where the visual content of the test video does not change. The experimental results are shown in Fig. 9, where we plot the transmission rate, goodput, and BER of TextureCode for each test video, for both the dynamic and static cases, respectively. As mentioned earlier, TextureCode achieves near-zero flicker perception for all the tested videos.

bigbuckbunny	Bosphorus	football	highway	Jockey

TABLE I: The screenshots of some test video sequences.



(a) (b)

Fig. 9: Transmit rate and goodput performance. (a) Dynamic scene, (b) Static scene.

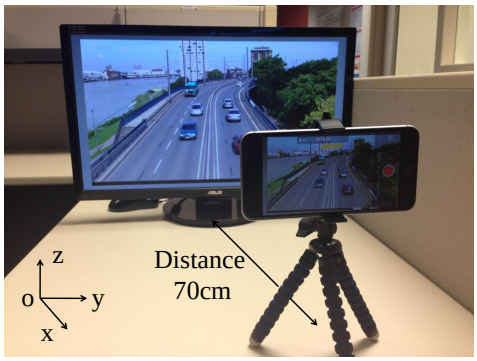


Fig. 8: Experiment setting

The experimental results from these plots indicate that TextureCode has an average goodput of 16.52 kbps for the dynamic case and 15.16 kbps for the static case, while bounding the average BER within 7% for the static case and within 20% for the dynamic case. We observe that the BER achieved in the static case is usually smaller than the dynamic case. This large error spike in the dynamic case happens because the original, unaltered video signal is changing, but our algorithm assumes a constant base-video signal. In fact, the *Jockey* dynamic video sequence has the fastest motion in our test, causing the highest BER (18%).

B. Comparison of TextureCode with prior Work

We compare the performance of TextureCode with HiLight, InFrame++ and the Hybrid schemes in terms of the goodput and flicker perception for the dynamic video cases, as shown in Figure 10. We elaborate our inference on each of these dimensions as follows:

1) Perceived flicker: We observed that while TextureCode, HiLight and Hybrid schemes showed no signs of flicker (flicker level was much below perceivable (subjective) threshold), there was still some residual flicker in InFrame++ at the test viewing distance of 70cm. In their paper, the authors of InFrame++ describe reducing the encoding block size to 12×12 to reduce flicker. Reduction in block-size reduces the area over which the block spatial transitions may be perceived by the human-eye. However, a reduction in block size translates to a reduction in communication range, an undesirable tradeoff. Such a problem is avoided in TextureCode as the design inherently reduces flicker without reducing block size. In particular, the *texton analysis* and *superpixels* methods ensure there are no edges between neighboring encoded units and align the edge of each encoded block to the edges already present in the content of the image.

2) Goodput and BER: To ensure a fair comparison of the candidate schemes we use a 32×32 pixel block-size for all schemes. We can observe from Figure 10 that TextureCode can achieve higher goodput than HiLight, slightly lower goodput than InFrame++, but InFrame++ introduces more visible flickers. The measured BER of TextureCode is 10%, which is also lower than the measured BERs of InFrame++ (31%) and HiLight (40%). HiLight encodes bits by modulating the *alpha channel* [21] to reduce human flicker perception. Although this technique significantly reduces flicker, with a block size of 32×32, the luminance changes are not easily captured by the camera, resulting in larger bit errors. We observed that the

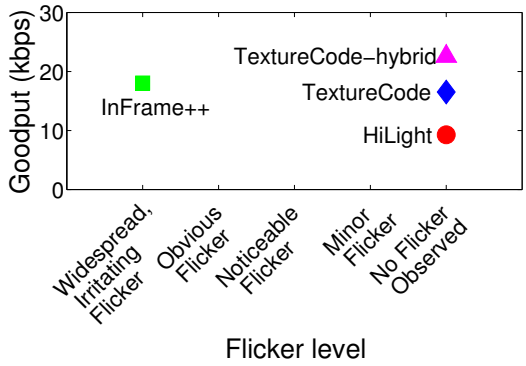


Fig. 10: Comparison between systems. The video resolution is 1280×704, the block size is 32×32 (the checkerboard size is therefore 40×22 blocks), and the screen-camera distance is 70 cm.

	Average (kbps)	Max (kbps)	Min (kbps)	Standard deviation (kbps)
InFrame++ (S)	19.26	23.78	15.93	2.05
InFrame++ (D)	18.05	21.23	15.21	1.63
HiLight (S)	9.66	9.82	9.01	0.13
HiLight (D)	9.27	9.47	8.99	0.1
TextureCode (S)	15.16	22.94	6.96	5.55
TextureCode (D)	16.52	33.08	3.85	9.31
TextureCode-hybrid (S)	21.9	28.68	13.7	4.4
TextureCode-hybrid (D)	22.57	39.13	9.89	8.84

TABLE II: Summary of goodput for the four systems: InFrame++, HiLight, TextureCode and Hybrid system. S: Static scenes, D: Dynamic scenes

bit error rate is as high as 40%, for HiLight, hence reducing effective goodput to about 10kbps.

The main advantage of using TextureCode is that it selectively encodes pixel regions in the frame that have a high signal-to-noise ratio at the receiver thus reducing the number of errors in decoding such pixels, resulting in a high goodput. To further improve the goodput of TextureCode, we also explored a new hybrid technique, named **TextureCode-Hybrid**, where we employ a mix of HiLight and TextureCode in a screen-camera communication system. In particular, we use TextureCode in "good" (high texture) blocks, and apply HiLight encoding to embed messages in the "bad" (plain texture) blocks, resulting in a higher transmit rate. This technique still ensures that there is no flicker in the encoded videos. Table II shows the average goodput (averaged over all test video samples) of the four candidate schemes: InFrame++, TextureCode, HiLight and the Hybrid systems. On average, the hybrid system achieves 22 kbps of goodput, increasing the goodput of TextureCode by 45% and the goodput of HiLight by 125%. There is a larger deviation in goodput of TextureCode and TextureCode-hybrid systems, because different texture content results in different amounts of encoded video content.

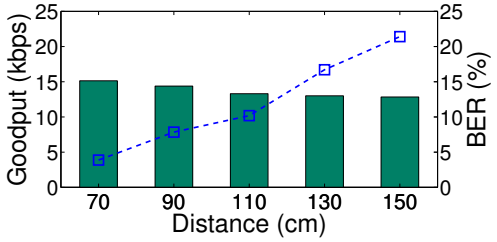


Fig. 11: BER and goodput vs. distance.

System	Capacity
HiLight	$\frac{frameRate*N}{6}$
InFrame++, τ = 4	$\frac{frameRate*N*bitsPerBlock}{8}$
TextureCode	$\frac{frameRate*N*encodedPercentage}{2}$

TABLE III: Comparison of Maximum Transmit Rate, which is normalized for the video display frame rate and the number of blocks per video frame. In InFrame++, *bitsPerBlock* is the number of bits per encoded block. In TextureCode, *encodedPercentage* is the percentage of encoded regions over the whole video frame.

C. Microbenchmarking

1) Communication Range: We examine the communication range of TextureCode by measuring the goodput and BER at increasing distance between the camera and the screen from 70 cm (the minimal distance at which only the screen pixels project onto the camera image) to 150 cm. We plot the average bit error rate (averaged over all videos) and goodput in Figure 11. The block size is 32×32in this experiment. As one can expect, the BER increases with distance as the camera-captured block size becomes smaller, resulting in higher inter-pixel interference [7]. We observe that TextureCode performs well when the distance is within 1 m, maintaining bit error rate less than 10% on average. One possible solution to improve performance at greater distances is to adaptively change encoded block sizes. We reserve such considerations for future work.

2) Maximum Transmit Rate: Table III shows the maximum transmit rates for HiLight, InFrame++, and TextureCode. A major improvement through TextureCode is that it can embed a bit for each block within every two frames of the carrier video, while HiLight requires 6 frames and InFrame++ requires 8 frames (including about 4 transitional frames) respectively. Although in TextureCode, the maximum transmit rate is reduced by a factor of *encodedPercentage*, we observed that this factor is about 30-40% from our experiments. As a result, the overall theoretical limit on transmit rate of TextureCode is almost of the order of HiLight and InFrame++.

VI. Related Work

Screen-camera communication. Screen-camera communication began with codes that are not embedded. PixNet [22] uses 2D OFDM to modulate high-throughput 2D barcode frame, and optimizes high-capacity LCD-camera communica-

tion. COBRA [14] is a color barcode system for real-time phone to phone transmission optimized for reducing decoding errors caused by motion blur. Another orthogonal class of work includes resolving the frame synchronization problem [15], extending the operational range [16], boosting the reliability and throughput of the screen-camera communication link [24]. More recent studies have focused on embedded screen-to-camera communications. Visual MIMO [26] is a real-time dynamic and invisible message transmission between screen and camera. VR Codes [25] is an invisible code to human eye, which uses high-frequency red and green light to transmit data to a smartphone's camera, where only the mixed colors are perceived by human eyes. HiLight [21] leverages pixel translucency channel to encode data into any screen. InFrame++ [23] uses complimentary frame composition, hierarchical frame structure and CDMA-like modulation to embed messages into videos. TextureCode differs in that it explores spatial coding, an orthogonal dimension to these previous works.

Video watermarking and steganography. Video watermarking and steganography also make the embedding into images and videos imperceptible [8] [17]. However, the techniques do not address real-world challenges in screen-to-camera communication channel as our system does. We made observations as to finding appropriate regions (in an image) for embedding inspired by the work in watermarking community and proposed a novel technique that addresses screen–camera channel distortions by encoding over spatio-temporal dimensions.

VII. Conclusion and Future Work

In this paper, we enable high-rate flicker-free embedded screen-camera communication. An examination of factors that affect flicker perception leads us to explore the spatial dimension of the design space and to combine it with more conventional temporal schemes. The resulting encoding scheme, TextureCode, is spatially adaptive based on texton and superpixel analysis. Experimental results show that this approach reduces flicker to unobservable levels while meeting or exceeding the goodput of existing schemes. These results show promise for significantly improving the performance of embedded screen-camera communications through techniques that jointly use multiple dimensions of embedding. This motivates future work to design such protocols and receivers that can automatically recognize the encoded regions in the transmitted video stream.

References

[1] http://ls.wim.uni-mannheim.de/de/pi4/research/projects/retargeting/test-sequences/.

[2] Elemental technologies 4k test sequences. http://www.elementaltechnologies.com/resources/4k-test-sequences.

[3] glvideoplayer. https://code.google.com/p/glvideoplayer/.

[4] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2274–2282, Nov 2012.

[5] Edward H Adelson. Lightness perception and lightness illusions. *New Cogn. Neurosci*, 339, 2000.

[6] Börje Andrén, Kun Wang, and Kjell Brunnström. Characterizations of 3d tv: active vs passive. In *SID symposium digest of technical papers*, volume 43, pages 137–140, 2012.

[7] A. Ashok, S. Jain, M. Gruteser, N. Mandayam, Wenjia Yuan, and K. Dana. Capacity of pervasive camera based communication under perspective distortions. In *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, pages 112–120, March 2014.

[8] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Review: Digital image steganography: Survey and analysis of current methods. *Signal Process.*, 90(3):727–752, March 2010.

[9] Charles Chubb, George Sperling, and Joshua A Solomon. Texture interactions determine perceived contrast. *Proceedings of the National Academy of Sciences*, 86(23):9631–9635, 1989.

[10] Tom Cornsweet. *Visual perception*. Academic press, 2012.

[11] Oana G Cula and Kristin J Dana. Recognition methods for 3d textured surfaces. In *Proceedings of SPIE conference on human vision and electronic imaging VI*, volume 4299, page 3, 2001.

[12] James Davis, Yi-Hsuan Hsieh, and Hung-Chi Lee. Humans perceive flicker artifacts at 500 [emsp14] hz. *Scientific reports*, 5, 2015.

[13] H DeLange. Experiments of flicker and some calculations of an electrical analogue of the foveal systems. *Physica*, 1952.

[14] Tian Hao, Ruogu Zhou, and Guoliang Xing. Cobra: Color barcode streaming for smartphone systems. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 85–98, New York, NY, USA, 2012. ACM.

[15] Wenjun Hu, Hao Gu, and Qifan Pu. Lightsync: Unsynchronized visual communication over screen-camera links. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 15–26, New York, NY, USA, 2013. ACM.

[16] Wenjun Hu, Jingshu Mao, Zihui Huang, Yiqing Xue, Junfeng She, Kaigui Bian, and Guobin Shen. Strata: Layered coding for scalable visual communication. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 79–90, New York, NY, USA, 2014. ACM.

[17] N.F. Johnson and S. Jajodia. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, Feb 1998.

[18] DH Kelly. Sine waves and flicker fusion. *Documenta Ophthalmologica*, 18(1):16–35, 1964.

[19] T. Leung and J. Malik. Recognizing surfaces using three-dimensional textons. 2:1010–1017, 1999.

[20] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. 43(1):29–44, 2001.

[21] Tianxing Li, Chuankai An, Xinran Xiao, Andrew T. Campbell, and Xia Zhou. Real-time screen-camera communication behind any scene. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 197–211, New York, NY, USA, 2015. ACM.

[22] Samuel David Perli, Nabeel Ahmed, and Dina Katabi. Pixnet: Interference-free wireless links using lcd-camera pairs. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, pages 137–148, New York, NY, USA, 2010. ACM.

[23] Anran Wang, Zhuoran Li, Chunyi Peng, Guobin Shen, Gan Fang, and Bing Zeng. Inframe++: Achieve simultaneous screen-human viewing and hidden screen-camera communication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 181–195, New York, NY, USA, 2015. ACM.

[24] Anran Wang, Shuai Ma, Chunming Hu, Jinpeng Huai, Chunyi Peng, and Guobin Shen. Enhancing reliability to boost the throughput over screen-camera links. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 41–52, New York, NY, USA, 2014. ACM.

[25] G. Woo, A. Lippman, and R. Raskar. Vrcodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 59–64, Nov 2012.

[26] Wenjia Yuan, Kristin Dana, Ashwin Ashok, Marco Gruteser, and Narayan Mandayam. Dynamic and invisible messaging for visual mimo. In *Proceedings of the 2012 IEEE Workshop on the Applications of Computer Vision*, WACV '12, pages 345–352, Washington, DC, USA, 2012. IEEE Computer Society.