Reading between Lines: High-rate, Non-intrusive Visual Codes within Regular Videos via ImplicitCode

Shuyu Shi
National Institute of Informatics, Tokyo, Japan
shi-sy@nii.ac.jp

Lin Chen **Wenjun Hu**
Yale Univerisity
New Haven, CT, USA
{lin.chen, wenjun.hu}@yale.edu

Marco Gruteser
WINLAB, Rutgers University
North Brunswick, NJ, USA
gruteser@winlab.rutgers.edu

ABSTRACT

Given the penetration of mobile devices equipped with cameras, there has been increasing interest in enabling user interaction via visual codes. Simple examples like QR Codes abound. Since many codes like QR Codes are visually intrusive, various mechanisms have been explored to design visual codes that can be hidden inside regular images or videos, though the capacity of these codes remains low to ensure invisibility. We argue, however, that high capacity while maintaining invisibility would enable a vast range of applications that embed rich contextual information in video screens.

To this end, we propose ImplicitCode, a high-rate visual codes that can be hidden inside regular videos. Our scheme combines existing techniques to achieve invisibility. However, we show that these techniques, when employed individually, are too constraining to deliver a high capacity. Experiment results show that ImplicitCode can deliver a significant capacity boost over two recent schemes, up to 12× that of HiLight [19] and 6× or 7× that of InFrame [32], while maintaining a similar or better level of invisibility.

Author Keywords

Screen-camera communication; Non-intrusive visual codes; Flicker fusion

ACM Classification Keywords

C.2.1 Network Architecture and Design: Wireless Communication

INTRODUCTION

The proliferation of mobile devices has generated renewed interest in enabling rich user interactions, both between the user and the device, and aiding the user to interact with the environment. Mobile phones are now equipped with assorted sensors, which can gather information about the environment and facilitate context-aware applications such as location based services.

In particular, phone cameras have been used to capture visual tags [28, 29] as authentication tokens since their early days. With the camera capability growing at a fast pace with each phone model revision, visual codes and their applications are also flourishing. For example, QR Codes [1], originally designed to tag cars for tracking in the automobile industry, are now seen everywhere (e.g., see Figure 1 of [16]), and often link the physical objects (or products and services) carrying these codes with their presence on the Internet. Product brochures (e.g., from IKEA) sometimes have dedicated prompts inviting the user to scan a page with their camera and obtain further product information or discount coupons.

We are also surrounded by an ever increasing number of electronic displays from TVs to public displays. To facilitate interaction across pervasive devices, it would be helpful for a mobile device to easily detect the display it is pointed at and the screen content shown. Adding visual codes to the displays could achieve this, similar to how printable visual codes have enhanced interaction with physical devices.

Since many codes like QR Codes are not designed to be aesthetically pleasing but occupy a prominent area of the display surface, they are visually intrusive yet incomprehensible to our eyes. There have been a long series of efforts designing visual codes that can be hidden inside regular images or videos. This dates back to early techniques of steganography and digital watermarking. More recently, a number of new schemes, VR Codes[34], HiLight[19], and InFrame[32], exploit new avenues for hiding the barcodes using existing commodity hardware.

HiLight modulates bits in slight changes in the pixel translucency level, which adjusts the color intensity of the host image in imperceptible ways while creating a channel for communication. VR Codes leverages the flicker fusion property of the human visual system when viewing videos played at more than 60 fps, i.e., adjacent frames are perceived to be mixed together, and therefore a video of two alternating frames can appear as a static image. InFrame builds on the insight from VR Codes to design a visual code displayed on a high-rate display.

Although previous schemes have explored various mechanisms for hiding, the capacity of these codes remain low in order to ensure invisibility.

We argue, however, that high capacity while maintaining invisibility would enable a vast range of applications that embed rich contexts in video screens, without having to resort to additional WiFi or cellular connectivity. Consider a movie screening scenario. Ideally, our eyes should just see the regular movie, but subtitles, foreign language tracks, background information about the ac-

UbiComp '15, September 7–11, 2015, Osaka, Japan.

http://dx.doi.org/10.1145/2750858.2805824

tors, and product purchase information could all be embedded into the movie and viewable with a Google-glass like device. Such visual codes require a significantly higher capacity than conventional visual tags encoding simple identification information of URL strings. Even for URL strings, we need a high throughput to enable low-latency detection. For example, detecting a 1000-bit URL within 100 ms of pointing the phone at the display requires a data rate of at least 10 kbps. The capacity potential can also be traded for a longer operating range, as shown by Strata [16].

A close examination of HiLight and InFrame suggest that the barcode hiding techniques, when employed individually, are too constraining to the capacity. This is because these permit only very subtle changes in the color of spatially and temporally adjacent code blocks to effectively hide the barcode from our eyes. These subtle changes provide very low signal-to-noise ratios and hence low capacity values. On the other hand, a prerequisite for a high-capacity code is sufficient distances between similar codewords, in this case the colors representing different bits.

That said, we find that combining these barcode hiding techniques in a simple way can be surprisingly effective. The key in our design is to smooth the color transition between adjacent blocks as efficiently as possible. We regulate the color change using limited translucency adjustment and then blend the colors of blocks adjacent both in space and in time. The resulting visual codes, ImplicitCode, deliver a high capacity while simultaneously remaining visually non-intrusive.

Although our scheme is simple, it outperforms competing schemes while offering more flexibility in the supported host images or videos. We implemented the ImplicitCode decoder as an iOS app as well as a desktop version for easy comparative performance evaluation. Experiment results show that ImplicitCode can deliver a significant capacity boost to two recent schemes, up to 12× that of HiLight [19] and 6× or 7× that of InFrame [32]. Compared to HiLight, we relax the permitted translucency change to boost the capacity. Compared to InFrame, the capacity gain comes from using fewer blocks to achieve a smooth color transition.

To summarize, we make the following contributions:

- We show that existing techniques using a single domain of non-intrusiveness achieves that at the expense of capacity, and may not eliminate flicker entirely.
- When combined, these techniques can simultaneously achieve invisibility and a high capacity.
- We design and implement ImplicitCode leveraging the above insight. ImplicitCode can deliver a significantly higher capacity than recent competing schemes.

BACKGROUND AND MOTIVATION

Related work

Hiding data and enabling communication implicitly in the environment is an important requirement for ubiquitous computing [33]. Even since phones were fitted with cameras, they have been leveraged to interact with the environment [28, 29].

Many visual codes have been designed for communication, from the more traditional 2D barcodes such as QRCodes [1] and Data Matrix [2] to videos of barcodes over display-camera links [24, 12, 15, 16, 31]. These are normally optimized for robustness, high capacity, or scalability with diverse operating conditions, and are visually intrusive. Instead, ImplicitCode aims to deliver a high capacity while being visually non-intrusive. We focus on the most relevant schemes below.

Data hiding via Steganography. Steganography and watermarking technologies have been traditionally employed to communicate secret data in images and videos [10, 21]. Such hidden data can be used for copyright protection or authenticity detection[13, 11, 8]. More recently, many printed brochures come with embedded codes[3, 25, 26, 27], often encoding URLs linking to more product information, and the reader is encouraged to scan those printed pages using their phones.

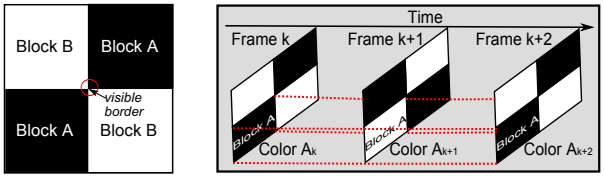
These schemes tend to be optimized for robustness, and embed static, short messages, which are easier to encode and decode. Instead, ImplicitCode joins other recent schemes to enable concurrent visual channels between display-camera links. These channels convey both explicit visual information for human eyes to consume and invisible, dynamic messages for the camera/decoder.

Unobtrusive display-camera communication. An earlier work proposed double-phase coding, such that the embedded information appeared as random background noise to the carrier image[18], although the scheme was only simulated. Bokode [23] is a novel 2D barcode which embeds data in the bokeh leveraging a special lens setup and captures it with an out-of-focus camera. AiD [35] is an LCD-based device which can alternately display between RGB backlight and near-infrared (NIR) backlight at 120 Hz. As human eyes cannot see the NIR spectrum, it can deliver information unobtrusively. Both Bokode and AiD require custom hardware. Other psychovisual features were also studied extensively to model human gaze [22, 17, 14]. Temporal psychovisual modulation (TPVM) [36] is a general paradigm that leverages the psychovisual redundancy of a signal display with a refresh rate over 60 Hz to present multiple visual percepts for different viewers.

A few recent schemes achieve invisible coding using commodity display and cameras on mobile devices, including VRCodes [34], HiLight [19] and InFrame [32]. These techniques individually can only offer limited benefit. However, when combined, these basic techniques can deliver a high capacity while remaining visually non-intrusive, as shown by the performance of ImplicitCode. We defer detailed descriptions of related coding schemes to later in this section.

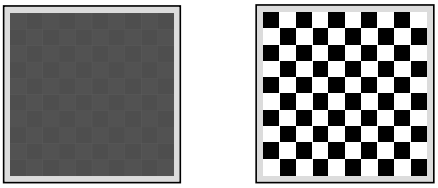
Cause of intrusiveness

Conventional barcodes use colored patterns to encode bits, either directly in the spatial domain, or in the fre-



(a) Spatial (b) Temporal

Figure 1: Intrusive block color transition.



(a) Low contrast (b) High contrast

Figure 2: Tradeoff between barcode visibility and ease of color detection.

quency domain. In order to recognize these patterns, their colors must be distinguishable, even in cases of low light, hardware noise, and ambient light. In other words, there must be a noticeable color transition between different patterns. Unfortunately, such differences are also prominent to human eyes, even though these barcode patterns are incomprehensible and hardly appealing visually.

Spatial transition is a defining feature of common barcode designs. Consider a QR Code, for example, which is composed of many black and white blocks. When any adjacent blocks have different colors, a clear border is visible between the blocks (Figure 1(a)).

Suppose we now want to display several different QR Codes in succession as a video. **Temporal transition** can also be an issue. This is seen between blocks at the same position in successive code images when they have different colors, as shown in Figure 1(b).

Further, a dedicated area of the screen or the print medium is needed to display the barcode. Ideally, we would like to embed a (sequence of) barcode(s) into a regular image or video to *hide* it from the human visual system.

Challenges in hiding barcodes

Fundamentally, the challenge to designing an invisible barcode is the tradeoff between visibility and ease of barcode detection and decoding by the camera/decoder. This tradeoff limits the capacity of these barcodes and dictates the decoding complexity.

Figure 2 shows a toy example of two barcodes with different levels of color contrast, with black for 0 and white/gray for 1. The color changes are less sharp on the left, and could potentially be superposed on a regular image. However, there is a higher probability of confusion between the black and gray blocks if the image is not captured perfectly.

Existing techniques for hiding barcodes

Roughly speaking, existing techniques for hiding information exploit *either* the spatial (pixel), temporal (inter-frame), *or* frequency domain properties of the carrier image or the video.

Frequency domain. The overall idea is to turn a host image to the frequency domain, and modulate some bits to the high-frequency components of the image. These components affect the image less than their low-frequency counterparts. When transformed back to the spatial (pixel) domain, the changes are spread over the entire image and less noticeable than direct changes to the pixels.

There are two main disadvantages. First, frequency domain coding by itself still alters the pixel domain patterns, and the embedded code is usually visible in smooth areas of the host image or video frame. Therefore, it is difficult to generalize this to *any* host image or video. Second, frequency domain processing is usually more computationally intensive, which incurs decoding latency on less powerful mobile devices.

Spatial. The pixel colors can be adjusted directly by tuning the *translucency* value, which effectively scales the intensity level of the overall image. If up to 1% of this value is changed, the resulting difference would be imperceptible to human eyes, as shown in HiLight [19]. This then creates an opportunity for modulating information. However, the small change is a rigid constraint on the channel capacity and does not permit simple, effective modulation directly in the pixel domain. HiLight encodes bits using a sequence of translucency changes over frames.

Temporal. Videos often exhibit temporal correlation due to mostly similar scenes. Such redundancy could be leveraged to modulate bits, for example, by using different motion vectors. However, this faces the same issue as seen in the frequency domain technique, in that the exact encoding is also dependent on the host video and hard to generalize.

A more recent technique exploits the inherent limitation of the human visual perception system. Our eyes can only distinguish individual video frames displayed up to 60 fps (or pulses up to 60 Hz). Beyond that rate, the frames will appear fused together to our eyes. For example, two alternating pure-color frames (so-called *complementary frames*) shown at 120 fps appears as a static image, whose color is approximately the average of the two actual colors. We can then generate a barcode image and its complement, add each to a host image, and make a video from alternating these two embedded images to hide the barcode image into the host image.

VRCodes [34] is the first system that exploits the effect of flicker fusion to design a novel visual code. InFrame [32] builds on VRCodes and uses a high-refresh-rate display to show a video embedded with barcodes.

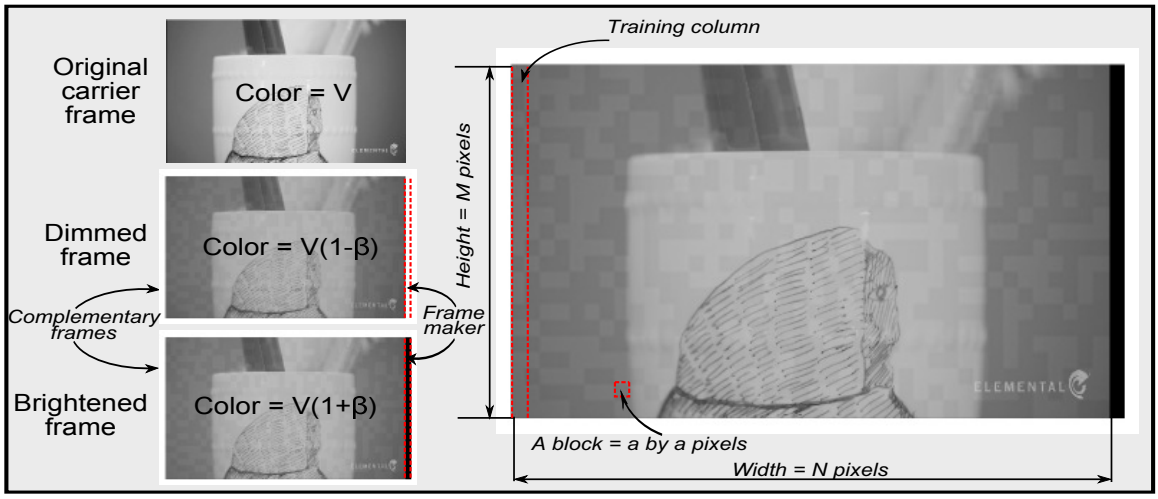


Figure 3: Frame layout of encoded videos using ImplicitCode.

Unfortunately, the border of any two complementary blocks of different colors remains visible in VRCodes (shown later in the paper in Figure 11). Although it can be mitigated by introducing dummy blocks of intermediate colors between the intended blocks in InFrame [32], this is a fundamental limitation of the color fusion behavior [4]. Overall, InFrame does not effectively address spatial block color transition and is not suited to videos with texture-rich natural scenes.

Addressing the limitations. Since any single technique above is limited, we explore combinations of them. Specifically, ImplicitCode combines the basic techniques in the spatial and temporal domain, along with edge smoothing between code blocks to mitigate any sharp transition. This is detailed in the next section.

SYSTEM DESIGN

Overview

Typically, the human visual perception system can only distinguish temporal information at a rate of up to 60 Hz[1]. If a video is played at a higher rate, the perceived frames will simply be mixtures of the original video frames[34]. This effect is analogous to recording a video at a lower capture rate than the display rate[24, 12, 15]. The overall idea of ImplicitCode is to display a video embedded with barcodes at a high rate (120 fps), such that the barcodes will be blended within the regular video when viewed at a low frame rate, but a high capture rate camera (at 240 fps) can still detect individual displayed frames, thereby decoding the barcodes.

The design of ImplicitCode answers three key questions:
(1) How to modulate bits by leveraging the perceived color fusion effects of the human visual system when viewing a video played at higher than 60 Hz?
(2) How to smooth the block transition spatially and temporally to effectively *hide* the code blocks in the carrier video?
(3) How to decode the hidden barcodes accurately?

[1]There are other factors (amplitude, spatial dimensions) at play that may allow flicker perception even above 60 Hz, though not significantly higher.

In the rest of the paper, we will refer to the original video without the barcodes as the *carrier video*. Once bits are encoded into the carrier video, we obtain a *barcode video*. Finally, we record the barcode video and try to decode those added bits from the *captured video*.

Modulating bits

Given a bit string, we first add error correction using Reed-Solomon coding of rate (80, 128) and then modulate the bits into the carrier video.

Figure 3 illustrates the modulation and the encoded frame layout. From a grayscale carrier video, we first duplicate each frame and divide them into square blocks. The corresponding blocks in each pair of duplicate frames then have their colors adjusted to encode bits. Specifically, for a pair of corresponding blocks, we represent 0 by using the blocks as they are from the original carrier video. To represent 1, the color of the first block is changed to $color_{original} \times (1-\beta)$, while the second block has its color turned to $color_{original} \times (1+\beta)$, where β is the change in the color translucency level, measured in percentage. The translucency basically controls the color intensity of the original video. Visually, the video can be dimmed or brightened by adjusting this parameter. We will discuss its value in the next subsection.

With bits added, the pair of 'duplicate frames' have become *complementary frames*. When two complementary frames are mixed together, the mixture appears visually indistinguishable from the original frame in the carrier video.

Frame layout. Once bits are modulated into the complementary frames, we add a few special patterns to help with decoding.

First, we add a *training column* of known colors to the left of each barcode video frame. These blocks will help us calibrate the threshold to classify the color of a barcode block to decode it to 0 or 1.

Second, we add a thin vertical bar, a *frame marker*, to the right of each barcode video frame to help identify a

Table 1: Adaptive translucency change

Average color intensity of block	Translucency β
≤ 79	12%
80 - 169	7%
≥ 170	5%

complete frame during capture. The bars in a pair of complementary frames are white and black respectively, such that they always mix to appear as a static gray bar to the human eye, again leveraging the frame mixing effects of the human visual system.

Finally, we surround each frame with a white border, similar to that of a QR code, to delimit the frame area against any background around the display.

Each barcode design typically comes with a specific frame pattern to aim frame detection. Our design partially follows the QR Code design. We have tried using the QR code corner patterns as well, but there was no noticeable improvement to the detection performance.

Smoothing code block transitions

Since we use color blocks to represent 0 and 1, any bit flip across blocks induces a change in the block color. In order for these bits to be hard to detect to the human eye, we need to smooth the transition between such color changes. The effects can be seen both temporally and spatially. In particular, previous work[34, 32] did not adequately address spatial transition.

Both types of transitions can be alleviated with a small translucency change threshold. However, if the threshold is too small, the colors to represent 1 and 0 will be almost indistinguishable. Therefore, we need to determine the optimal threshold.

Determining the translucency change β. We make pure color videos as the carrier videos, with their intensity varying from dark to bright using grayscale values 60, 140, and 220 respectively. We then vary the translucency change threshold β at 1% increments to generate more pure color videos. 15 subjects were asked to view these and report whether they appeared the same as the original three colors.

We found empirically that the darker the video, the larger β can be. For the three pure color clips, the threshold can be up to 12%, 7% and 5% respectively while the color changes remain imperceptible. For regular videos showing a range of intensity levels within each frame, we adapt the β value based on the average color intensity of a block, as shown in Table 1, using the results of the pure color clip tests.

Spatial transition. Unless spatially adjacent blocks in a pair of complementary frames always encode the same information, a sharp transition occurs if the corresponding bits flip between 0 and 1. To reduce this effect, we adopt a standard edge smoothing algorithm, such as the one in Matlab, to blend together adjacent blocks in each frame.

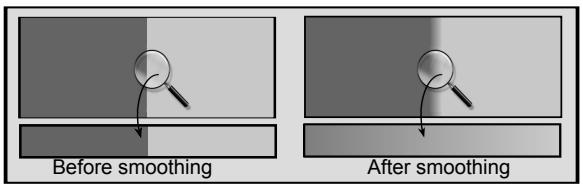


Figure 4: The border between the blocks before and after smoothing.

The effect is shown in Figure 4. In the right figure, the color transition between the two blocks is smooth and the border in between is less obvious.

Temporal transition. Once the translucency threshold is chosen, temporal transition is achieved automatically when the barcode video frames mix together.

Remarks. Although all techniques so far aim to reduce visual intrusiveness, they have a collateral benefit of increasing the capacity, because we achieve non-intrusiveness *using a minimum number of code blocks*. Thanks to this, ImplicitCode is able to modulate more bits than competing schemes using the same number of code blocks. As we will show later, competing schemes trade off capacity to ensure invisibility.

Decoding

There are two main challenges in the decoding process: extracting the barcode blocks and determining their color values.

Identifying complementary frames. Since the barcode video is recorded at twice the display rate, we expect to capture 4 frames for every pair of complementary frames. Due to the rolling shutter effect from the CMOS camera sensor[34, 15], 2 of the 4 recorded frames may be captured while the original video is transitioning from one frame to the next one. In other words, these recorded frames are mixtures of two frames[24, 12, 15], and should be discarded.

For the remaining two captured frames, the frame marker in one is brighter (in fact, it is brighter in this frame than in the other 3 of the 4 frames). This is identified as the first of the complementary frames. The remaining frame is the other complementary frame.

Once we bootstrap one pair of complementary frames, the rest can be identified similarly by examining successive groups of 4 captured frames at a time.

Extracting barcode blocks. For each pair of complementary frames, we *subtract* the first frame (the dimmer one) from the second (the brighter one), i.e., we perform $(\texttt{color}_{brighter\ frame}$ - $\texttt{color}_{dimmer\ frame})$ pixelwise. This leaves us with a *difference frame* per pair of complementary frames.

Determining code block color. Conventionally, the color of a block is determined by comparison to a threshold value. This value may be static or determined dynamically (as for QR Codes). Both approaches assume that the same few *absolute* colors are used consistently

throughout the barcode frames. This is reasonable, as traditional barcodes only use a few colors to modulate bits, such as QR Codes (using black and white) or HCCB[5] (using 4 colors).

Determining the threshold value is harder for ImplicitCode, however. Since we modulate bits as *relative* color change to the carrier video, we cannot use absolute threshold values. For example, the grayscale value of 100 might normally be considered black and representing 0 in QR Codes, but for ImplicitCode, it could be 1 if the carrier video block is even darker. In other words, we also need to determine the threshold between the colors representing 1 or 0 relative to the carrier video, even though we cannot assume any *a priori* knowledge of the carrier video. Therefore, we resort to linear SVM[9], a training based approach.

The **linear SVM**-based classification method needs n training blocks whose labels (0 or 1) are known beforehand. Then we scan the blocks representing bit 0 (or 1) to find the maximum (or minimum) grayscale value of the blocks. The average of the minimum grayscale value for 1 and the maximum for 0 is set as the threshold. The rest of the blocks in the frame are then classified as corresponding to 0 or 1 based on this threshold value. Ideally, we want the brightness of the carrier image at these training blocks to be representative in order to increase the successful classification rate.

Specifically, we take the following steps:

1. For each block in the training column, choose a quarter of the pixels around the center of the block to calculate the average color intensity of the block.

2. Since we know the bit value represented by each block, we find the minimum color intensity (`minBit1`) of all bit-1 blocks and the maximum color intensity (`maxBit0`) of all bit-0 blocks.

3. After scanning all blocks in the training column, we compute `threshold = (minBit1 + maxBit0)/2`.

We will show in the performance section that the complexity of this classification, a one-pass scan, is comparable to that of traditional approaches. Due to the temporal similarity between successive carrier frames, the same training results might suffice for several frames, although we currently perform the training once per frame. We leave the optimization to future work.

ImplicitCode IMPLEMENTATION

Encoder

The encoder is implemented in Matlab, using the image processing toolbox.

Each frame of the carrier video is loaded to Matlab or generated as a matrix of grayscale values, and subsequent operations are directly performed on these values. Given an input bit string for the barcode, the complementary frames are generated by adjusting the grayscale values as described in the previous section. Block edge smoothing is done using the built-in image-blending tools in Matlab[6]. Finally we put the encoded frames together into a 120-fps video in `.mkv` format (without compression) using `FFmpeg`. Quick experiments show that saving the 120 fps video in `.mp4` hardly affects the decoding accuracy, but the uncompressed format plays back more smoothly and minimizes player side artifacts for our experiments. This is then shown on a display with at least 120 Hz refresh rate and GPU hardware that is sufficiently powerful to drive the display.

Decoder

We implement the decoder both as a desktop emulation to be run offline and as an iOS app to be run online. The software libraries used are the same for both versions, except that the app is integrated with the iOS APIs.

(Barcode) video recording. We record the barcode video using the regular video recording functionality of the camera on iPhone 6 at 240 fps. We obtain a video in `.mp4` format this way.

Processing individual recorded frames. After we record the video, two threads are used to generate individual video frames and to decode the barcode images extracted from the recorded frames. This roughly corresponds to pipelining the decoding processing, with each threading corresponding to one stage of the pipeline.

The recorded video is only available in a compressed format, and therefore we need to effectively decode the video to pull out individual frames for further processing[2]. This is done in the first thread, using the `AVFoundation` framework in iOS. Once a frame is generated, we add it to a dispatch queue, a shared queue between threads.

The second thread then applies image cropping methods and perform perspective transformation to the frames in the queue. It then decodes the barcodes following the procedure described in the previous section.

With this implementation, generating individual video frame is the bottleneck. We have observed that the second thread always completes more quickly and the system is usually blocked on the first thread to generate individual video frames. This is a limitation from existing tools and APIs, and we cannot improve this at the moment. In contrast, decoding the frames can be accelerated by employing more concurrent threads, i.e., more copies of the second thread above.

PERFORMANCE

Overall setup

Test videos and default encoding parameters. We use 9 carrier videos. Three of them are the same pure color videos mentioned earlier. Each is essentially an image of a single color. The color intensity levels of the three vary from 60 to 220 in grayscale values. The other 6 videos are selected from the standard test video sequences [7]: *coast*, *news*, *mobile*, *foreman*, *cactus*, and

[2]In comparison, Android provides an API to return individual raw captured frames if the frame capture rate is below 30 fps. That would obviate the need for this first thread in our system.

(a) dark (b) medium (c) bright

(d) coast (e) news (f) mobile

(g) foreman (h) cactus (i) suzie

Figure 5: Snapshots of the carrier videos used.

suzie. They feature a range of color contrast levels, motion, and background texture. These videos are usually used to evaluate how various components of complex natural scenes affect the performance of a video coding algorithm. Similarly, these components also affect the performance of ImplicitCode. Figure 5 shows a snapshot of each sequence, and we embed barcodes, composed of random bit strings, in the entire sequence for each video. Unless otherwise specified, each carrier video frame is divided into code blocks of 32×32 pixels each. We play all barcode videos at 120 fps using the MPC-HC player on an ASUS VG278HE display, which is capable of a refresh rate up to 144 Hz.

Default capture conditions. In most experiments, the barcode videos are recorded using the camera of an iPhone 6, 40 cm from the screen at 240 fps, using all the auto settings, and the captured video resolution is 1280×720 pixels. We use the iOS decoder app to measure timing and the desktop decoder for comparison experiments, as it is easier to obtain logs from the desktop decoder and perform analysis. For each capture setting, we record the video 10 times and report the average results over the 10 captures.

Metrics. Recall that the design goal for ImplicitCode is to achieve a tradeoff between visual-intrusiveness and rate. Therefore, we evaluate ImplicitCode in terms of a subjective measure of visual-intrusiveness reported by subjects watching the videos and the achievable bit rate for the barcodes embedded. The latter is the combined effect of the barcode capacity due to our encoder design and the decoding accuracy (the percentage of correctly decoded bits out of all bits embedded). We will examine these in detail below.

Microbenchmarks

Decoding time. We measure how long it takes to decode each barcode frame using the iOS app. Table 2 shows the results.

Training column placement. In our design, the leftmost column of each video is used as the training column to aid decoding. We next show that the decoding performance is not sensitive to (1) the training column placement, or (2) the carrier video content. For each carrier video sequence, we randomly select 40 frames from different parts of the sequence, and use column p of the p^{th} frame (out of the 40) as the training column for that frame. Figure 6 shows that the decoding performance is quite stable regardless of the frame content or the training column placement. The videos shown have the highest/lowest decoding accuracy.

Table 2: Processing time breakdown

Operation	Time (ms)
Frame detection	17
Frame selection	1
Frame subtraction	10
SVM decoding	12

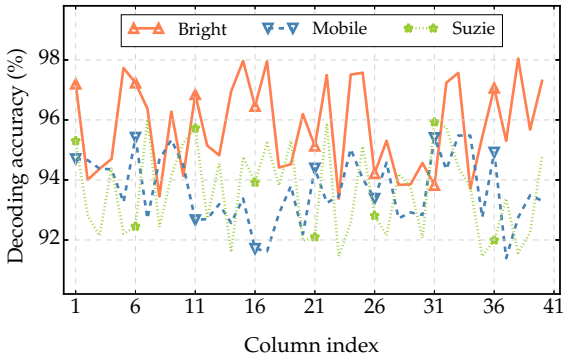


Figure 6: Training column placement and decoding accuracy.

Different thresholding techniques. As explained earlier, ImplicitCode cannot use a conventional static or dynamic thresholding approach, so we rely on linear SVM-based training. The time complexity for this method is N, including n units of time for training and $N-n$ units of time for classification. We next compare this approach to several alternatives, both analytically and experimentally.

Using **static thresholding**, we can define a prespecified grayscale value, say 127, as the dividing point between "white" and "black". For a frame with N blocks, the total time needed to determine all block colors is then N, which is the minimum necessary. However, static thresholding cannot adapt to the ambient brightness level or the dynamic range of the camera. For example, white blocks may be mis-classified as black in a dark environment.

Alternatively, we can scan all N blocks of a captured frame once to find the maximum and minimum grayscale values, and take the average of the two extremes as the **dynamic threshold**. Then we scan the whole frame for a second time to determine the color of each block. This way, the decoding process can adapt to the capture conditions. The total time needed for the whole color classification process is then $2N$, N for each scan. For ImplicitCode, however, the brightness

Table 3: Comparison of decoding thresholds

Method	Accuracy	Complexity
Static	53.2%	N
Dynamic	65.7%	$2N$
SVM	93.6%	N
k-NN	93.5%	$\frac{1}{2}(-n^2+n+N^2-N)$

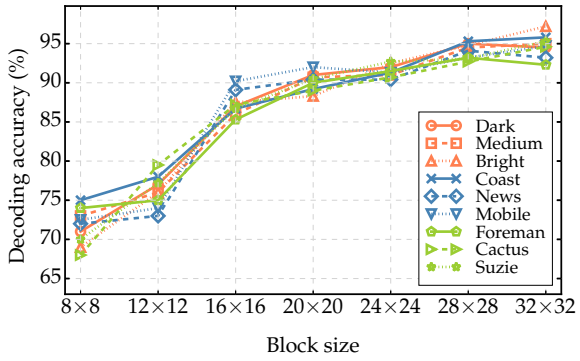


Figure 7: Decoding accuracy and block size.

distribution of the carrier image may skew the grayscale extremes and hence the threshold value calculated.

Similar to SVM, **k-NN**[9] also requires n training blocks with known labels. In contrast to the other methods, k-NN does not define an explicit threshold. It assigns labels to a block pending classification by a majority vote among k blocks with similar grayscale values that have already been assigned labels. The running time for k-NN is $\sum_{i=1}^{N-n} t(n+i-1,k)$, where $t(n+i-1,k)$ is the time needed to find k nearest neighbors among $n+i-1$ points, determined by the nearest neighbor search algorithm. If we use the naive linear search, the time complexity for k-NN will be $\frac{1}{2}\left(-n^2+n+N^2-N\right)$. Table 3 compares the decoding accuracy averaged over all frames and the time complexity using the four thresholding techniques discussed. N is the total number of blocks available per frame, and n is the number of blocks used for training. It is clear that neither static or dynamic thresholding delivers adequate decoding performance. SVM and k-NN can achieve comparable decoding accuracy, but k-NN incurs a higher time complexity. Therefore, we use SVM in ImplicitCode. This essentially bounds the decoding accuracy results in later experiments.

Further, the SVM-based approach automatically adapts to the ambient light conditions. Our experiments show that different lighting conditions have little effect on the decoding performance, and we omit detailed results.

Performance vs block size. The code block size affects the decoding accuracy in two ways. First, due to the edge smoothing, only the center part of each block can be used for decoding. If the block is too small, even the center pixels may need to have their colors adjusted in the edge blending operation. Second, in general a larger block affords more redundancy within the block to guard against pixel noise. On the other hand, more bits can be embedded into the carrier video using smaller blocks.

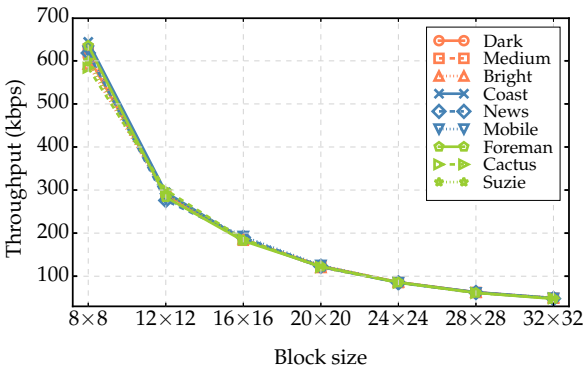


Figure 8: Throughput vs block size.

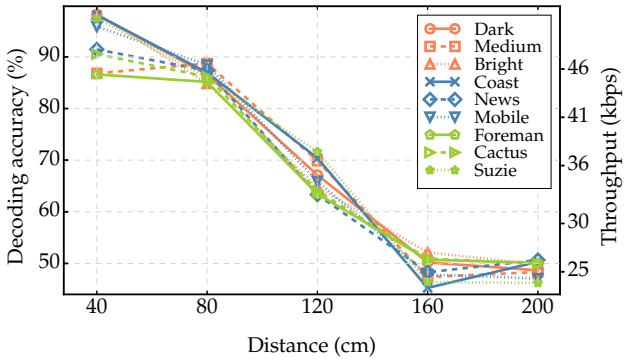


Figure 9: Decoding accuracy and throughput of ImplicitCode vs capture distance.

Figure 7 shows that the decoding accuracy roughly increases with the code block size. Figure 8 shows the throughput vs block size. There is a clear tradeoff between the supported bit capacity and the decoding accuracy. A block size of 8×8 yields the lowest decoding accuracy but the highest code capacity and throughput.

Capture distance. The decoding accuracy and throughput achieved at various capture distance is shown in Figure 9. ImplicitCode works effectively up to a range of 80 cm.

Performance comparison

As explained earlier, HiLight and InFrame are two recent schemes for non-intrusive barcodes, and we implement both alongside the desktop decoder for ImplicitCode to compare their performance. All three schemes modulate bits by changing the color intensity, while leveraging different properties to achieve invisibility. Since HiLight and InFrame each exploits *one* technique for non-intrusiveness, this set of comparison highlights the effect of combining these techniques as opposed to adopting them individually. We do not directly compare with VR Codes, since the coding scheme is not exactly comparable to ImplicitCode, and InFrame is already based on VR Codes.

HiLight. In each carrier frame, the translucency of a block is either 1 or 0.99. Over 24 frames, we obtain different sequences of 1 and 0.99 at various block positions. The bit 0 is represented by the sequence (0.99,

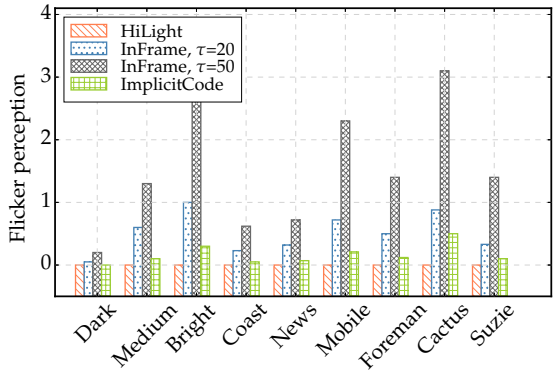


Figure 10: Perceived flicker of barcode videos.

Figure 11: Borders between blocks of different colors remain visible in VRCodes.

1, 0.99, 1, . . . , 0.99, 1), and the bit 1 by the sequence (0.99, 0.99, 1, 0.99, 0.99, 1, . . . , 0.99, 0.99, 1).

InFrame. InFrame merges contiguous p×p pixels in a frame to form a *SuperPixel*, and s×s SuperPixels to form a *Block*. We refer to each pair of Blocks at the same positions in two complementary frames as *complementary Blocks*.

To represent 0, the two complementary Blocks do not change their color values. To represent 1, the even-indexed SuperPixels within the Blocks retain their original colors; For the odd-indexed Superpixels, those within the *earlier* Block have their color values *subtracted* by a constant to become $(original - delta)$, and those within the *later* Block have their color values incremented by a constant to become $(original + delta)$.

Since the color change might be large enough to be visible, some dummy frames are also added in between the *actual* complementary frames to mitigate the color transition.

Perceived flicker. To evaluate the extent of visual intrusiveness to human eyes, we asked 16 subjects to watch the barcode video clips and score the quality of these videos. We use the *flicker perception* metric as in [32], which quantifies the flicker level using integers 0—4, ranging from "no difference" to "strong flicker and artifact". For each video, we averaged the flicker scores across the subjects and plot them in Figure 10.

Overall, ImplicitCode videos exhibit less flicker than InFrame videos. The τ value in InFrame indicates how much the color value changes between the corresponding blocks in consecutive frames. InFrame with $\tau = 50$ exhibits most flicker, as the color intensity change is abrupt, leading to noticeable block transitions and hence poor visual quality both spatially and temporally. This is in fact a fundamental limitation of the color fusion behavior inherited from VRCodes. Although two colors can appear fused, a border between blocks of different colors remains visible. The effect is shown implicitly in Figure 10 of the original VRCodes paper[34], and illustrated in Figure 11. Instead, ImplicitCode smoothes such transitions.

Table 4: Comparison of theoretical capacity

System	Capacity (bits)
HiLight	$\frac{1}{24} f\ N$
InFrame, $\tau = 12$	$\frac{1}{12} f\ N$
InFrame, $\tau = 14$	$\frac{1}{14} f\ N$
ImplicitCode	$\frac{1}{2} f\ (N - n)$

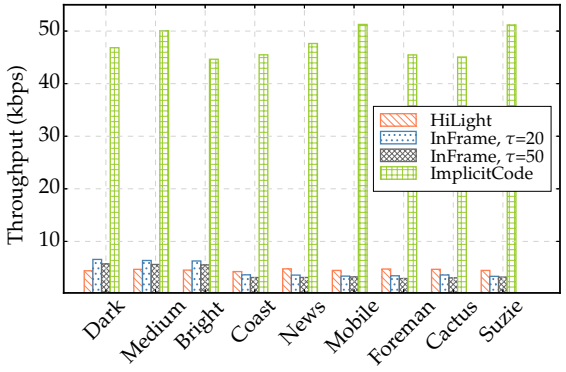


Figure 12: Measured goodput of ImplicitCode vs HiLight and InFrame.

Compared to HiLight, ImplicitCode shows more flicker, as we adopt a larger translucency change threshold in order to boost the barcode capacity. However, HiLight only supports static carrier images and constrains the barcodes more than ImplicitCode.

Capacity. We can calculate the barcode capacity based on the encoding schemes. Table 4 lists the theoretical capacity for HiLight, InFrame, and ImplicitCode, where $f = 120$ fps, and N and n are the total number of blocks per frame and per column respectively. Figure 12 compares the measured goodput, with decoding accuracy accounted for, of the three schemes from our experiments, using carrier videos of the same sequence length and the frame size[3].

As per our design goal, ImplicitCode can support a much higher capacity, 12× that of HiLight and 6 or 7× that of InFrame. Once normalized for frame rate and block size, ImplicitCode can embed a barcode within every two frames of the carrier video, while 24 frames are required for HiLight and 12 or 14 frames are needed for InFrame, depending on the τ value. This also means that ImplicitCode can start decoding after collecting 2 frames, while HiLight needs to wait for 24 frames and InFrame needs 6 or 7.

[3]For HiLight, the carrier video is simply a static image.

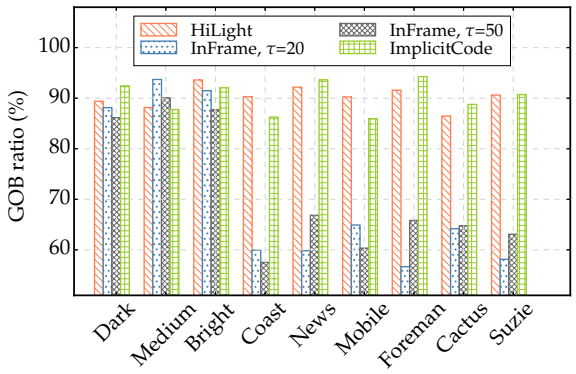


Figure 13: GOB ratio of the three schemes.

In comparison with HiLight, ImplicitCode gets a capacity boost from the larger translucency change threshold, which makes it possible to distinguish the color difference between regular blocks and dimmed blocks. We can then encode a bit in each pair of such blocks (as seen between the corresponding blocks in our complementary frames) and decode the bit by identifying the color difference. In HiLight, the translucency only changes by 1%, which is meant to be very close to the original color. While this effectively hides the change, it is not sufficient to encode a bit by itself. Therefore, HiLight employs a longer sequence of blocks, i.e., 24, over time to represent different bits so that they can be accurately distinguished from one another. Decoding requires an FFT over the 24 frames to recover the sequence pattern.

On the other hand, InFrame modulates each bit using 6 or 7 spatial blocks, and therefore also encodes less information than ImplicitCode. These 6 or 7 frames are needed to reduce the color transition gradient between the blocks. However, they are still not as effective as ImplicitCode's approach of blending block colors.

GOB ratio. InFrame considers each $m \times m$ Block a *Group of Block (GOB)*. If all of these Blocks are decoded successfully, this is an *available GOB*. The *GOB ratio* is the total GOBs divided by the number of available GOBs, effectively showing the decoding accuracy.

Figure 13 shows that both ImplicitCode and HiLight can achieve a GOB ratio accuracy of over 93%, regardless of the carrier video. For InFrame, however, once the carrier video contains texture, as is often seen in standard test video sequences, the GOB ratio precision drops dramatically. This is because, in its decoding process, InFrame assumes that the colors of every pixel within a Block in the carrier video frame are almost the same, which is not the case when the carrier image presents a natural scene instead of a plain color. Therefore, InFrame cannot be universally applied to all carrier videos.

CONCLUSION

In this paper, we explore how to design high-rate, non-intrusive visual codes with ImplicitCode. These codes are designed to be only visible to phone cameras but not to the human eye. ImplicitCode joins several recent efforts, especially HiLight [19], VRCodes [34], and InFrame [32], using commodity displays and cameras.

In particular, we show that the techniques adopted in previous work are limited when used individually. However, when combined, we can achieve a significant capacity boost of 12× over HiLight and 6× or 7× over InFrame. Further, InFrame-encoded videos still exhibit flicker, while ImplicitCode is hardly perceptible. More generally, our experiments using videos of natural scenes suggest that many complex factors affect the flicker perception, the embedding efficiency, and the decoding performance. It would be more effective to also incorporate design elements leveraging other psychovisual features and adapt to the video content. ImplicitCode shows that combining even a few techniques from multiple dimensions is promising, and takes us a step closer to practical barcodes that are both high-rate and non-intrusive.

Limitations and future work

Concurrent with our work, both HiLight and InFrame have been refined — HiLight now embeds into videos [20] and InFrame++ supports a higher theoretical capacity [30]. However, both incur higher decoding errors. We will study their performance in future.

The current ImplicitCode design is still limited to grayscale carrier video sequences and non-realtime decoding. However, we believe these can be addressed with a refined design and implementation.

Support for color videos. ImplicitCode expects the mixture of a pair of complementary frames to appear visually indistinguishable from the original frame. This holds for grayscale frames, because the average intensity of the two complementary block colors is roughly the same as the original. For color videos involving multiple color channels, the color mixture is more complex. Preliminary experiments suggest that we cannot simply apply the current technique per color channel to obtain the ideal color mix. Further, we may need to brighten and dim by different amounts when generating the complementary frames. We will investigate in future work.

Realtime decoding is currently bottlenecked by generating individual frames from the captured video. This is an artifact from the existing smartphone APIs rather than our design, and will similarly affect other designs requiring very high-rate video capture. Android phones and iPhone 6 can only support capturing individual raw frames up to 30 and 60 fps respectively. Only compressed formats are available beyond these rates, e.g., at 240 fps as required by ImplicitCode, which then necessitates the slow frame generation step. This issue would be resolved automatically if raw frame capture becomes available at 240 fps.

ACKNOWLEDGMENT

Marco Gruteser's participation in this work is supported by the US National Science Foundation (NSF) under grant CNS-1065463.

REFERENCES

1. ISO/IEC 18004:2006. Information technology. Automatic identification and data capture techniques. Bar code symbology. QR Code.
2. ISO/IEC 16022:2006 Information technology- Automatic identification and data capture techniques-Data Matrix bar code symbology specification.
3. `http://www.digimarc.com/products/discover/id-manager`.
4. Private communication with Grace Woo.
5. `http://research.microsoft.com/en-us/projects/hccb/`.
6. `http://www.mathworks.com/matlabcentral/fileexchange/30790-image-pyramidgaussian-and-laplacian/content/`.
7. `http://www.elementaltechnologies.com/resources/4k-test-sequences`.
8. M. Alghoniemy and A. Tewfik. Geometric invariance in image watermarking. *Image Processing, IEEE Transactions on*, 13(2):145–153, Feb 2004.
9. C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
10. A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt. Review: Digital Image Steganography: Survey and Analysis of Current Methods. *Signal Process.*, 90(3):727–752, Mar. 2010.
11. I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon. Secure Spread Spectrum Watermarking for Multimedia. *Trans. Img. Proc.*, 6(12):1673–1687, Dec. 1997.
12. T. Hao, R. Zhou, and G. 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. ACM, 2012.
13. F. Hartung and M. Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 87(7):1079–1107, Jul 1999.
14. X. Hou and L. Zhang. Saliency detection: A spectral residual approach. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007.
15. W. Hu, H. Gu, and Q. 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. ACM, 2013.
16. W. Hu, J. Mao, Z. Huang, Y. Xue, J. She, K. Bian, and G. 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. ACM, 2014.
17. L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.
18. S. Kishk and B. Javidi. Information hiding technique with double phase encoding. *Appl. Opt.*, 41(26):5462–5470, Sep 2002.
19. T. Li, C. An, A. Campbell, and X. Zhou. Hilight: Hiding bits in pixel translucency changes. In *Proceedings of the 1st ACM MobiCom Workshop on Visible Light Communication Systems*, VLCS '14, pages 45–50. ACM, 2014.
20. T. Li, C. An, X. Xiao, A. T. Campbell, and X. 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.
21. Q. Liu, A. H. Sung, and M. Qiao. Video steganalysis based on the expanded markov and joint distribution on the transform domains detecting msu stegovideo. In *Proceedings of the 2008 Seventh International Conference on Machine Learning and Applications*, ICMLA '08, pages 671–674. IEEE Computer Society, 2008.
22. O. L. Meur, P. L. Callet, and D. Barba. Predicting visual fixations on video based on low-level visual features. *Vision Research*, 47(19):2483 – 2498, 2007.
23. A. Mohan, G. Woo, S. Hiura, Q. Smithwick, and R. Raskar. Bokode: Imperceptible Visual Tags for Camera Based Interaction from a Distance. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 98:1–98:8. ACM, 2009.
24. S. D. Perli, N. Ahmed, and D. 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. ACM, 2010.
25. A. Pramila, A. Keskinarkaus, and M. Oulu. Camera based watermark extraction - problems and examples. In *In: Proceedings of the Finnish Signal Processing Symposium 2007*, 2007.
26. A. Pramila, A. Keskinarkaus, and T. Seppänen. Watermark robustness in the print-cam process. In *Proceedings of the Fifth IASTED International Conference on Signal Processing, Pattern Recognition and Applications*, SPPRA '08, pages 60–65. ACTA Press, 2008.
27. A. Pramila, A. Keskinarkaus, and T. Seppänen. Reading watermarks from printed binary images with a camera phone. In *Proceedings of the 8th International Workshop on Digital Watermarking*, IWDW '09, pages 227–240. Springer-Verlag, 2009.
28. M. Rohs. Real-world interaction with camera phones. In *Proceedings of the Second International Conference on Ubiquitous Computing Systems*, UCS'04, pages 74–89, Berlin, Heidelberg, 2004. Springer-Verlag.
29. P. Vartiainen, S. Chande, and K. Rämö. Mobile Visual Interaction: Enhancing Local Communication and Collaboration with Visual Interactions. In *Proceedings of the 5th International Conference on Mobile and Ubiquitous Multimedia*, MUM '06, New York, NY, USA, 2006. ACM.
30. A. Wang, Z. Li, C. Peng, G. Shen, G. Fang, and B. 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.
31. A. Wang, S. Ma, C. Hu, J. Huai, C. Peng, and G. 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. ACM, 2014.
32. A. Wang, C. Peng, O. Zhang, G. Shen, and B. Zeng. Inframe: Multiflexing full-frame visible communication channel for humans and devices. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 23:1–23:7. ACM, 2014.
33. M. Weiser. Creating the invisible interface: (invited talk). In *Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology*, UIST '94, pages 1–, New York, NY, USA, 1994. ACM.
34. G. Woo, A. Lippman, and R. Raskar. Vrcodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter. In *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, ISMAR '12, pages 59–64. IEEE Computer Society, 2012.
35. S. Wu and J. Xiao. Aid: Augmented information display. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 523–527. ACM, 2014.

36. X. Wu and G. Zhai. Temporal Psychovisual Modulation: A New Paradigm of Information Display [Exploratory DSP]. *Signal Processing Magazine, IEEE*, 30(1):136–141, Jan 2013.