# Screen Codes: Visual Hyperlinks for Displays

J. P. Collomosse and T. Kindberg

**Abstract**

We present ‘Screen codes’ - a space- and time-efficient, aesthetically compelling method for transferring data from a display to a camera-equipped mobile device. Screen codes encode data as a grid of luminosity fluctuations within an arbitrary image, displayed on the video screen and decoded on a mobile device. These ‘twinkling’ images are a form of ‘visual hyperlink’, by which users can move dynamically generated content to and from their mobile devices. They help bridge the ‘content divide’ between mobile and fixed computing.

## I. Introduction

In recent years, machine-readable visual tags and camera-equipped mobile devices have been combined to yield a new interaction method for mobile applications. Tags may be imaged using commodity devices (such as camera-phones or PDAs) to drive mobile information access, ticketing, or marketing applications. In this context, visual tags are referred to as ”mobile codes”, and conventionally take the form of 1D or 2D barcodes such as the ISO UPC[1], or Datamatrix[2] symbologies.

The position put forward in this paper is that we should concentrate more engineering and research effort on the important use case where mobile codes are displayed on screens rather than print, as dynamically generated ‘visual hyperlinks’ to content and services. The motivation is to break down the ‘content divide’ between mobile and fixed devices that afflicts users. Currently, moving content between the resource-rich fixed web and mobile devices is cumbersome, and involves at best some combination of Bluetooth or cable, and web upload or download. Visual hyperlinks remove that inconvenience. In one example, a screen in a store displays mobile codes linked to music tracks, which users download to their mobile phones to sample before purchase. In another example, a device such as a printer displays a mobile code that a second device such as a mobile phone reads in order to connect conveniently to the first device over Wi-Fi or Bluetooth and upload content. In these examples, visual hyperlinks enable the mobile device to act as a first-class citizen of the fixed web.

Conventional mobile code symbologies are appropriate for many applications, but can be unsuitable for applications involving displays because they are sometimes too large in relation to the available screen real-estate. In the example of the retail display of music samples, human-readable content on the screen may be dense, and space for codes consequently low. In the second example, devices such as printers tend to have small displays. The relatively poor resolution and optics of commodity mobile cameras places an upper limit on the data density of such codes; encoding as little as 100 bytes may require unacceptably large areas. Not only is there insufficient room, but existing mobile code symbologies can be aesthetically disruptive when relatively large.

We introduce a new form of mobile code, the “Screen code”, which relaxes these limitations and so broadens the gamut of potential applications for mobile codes. Screen codes provide a *robust* mechanism for trading off the space occupied by a code against decoding time, so enabling transmission of larger data volumes. We achieve this by creating time-varying codes on a video display such as a monitor, television, or public display. In contrast to existing codes, Screen codes are created from an arbitrary image, allowing the designer to easily customize appearance. Screen codes may therefore be smaller, and more visually appealing than conventional mobile codes containing the same data.

### A. Related Work

There have been previous attempts to meet our objective, of a space-efficient, aesthetically acceptable visual data channel from a display to a mobile device, without a back-channel. Serial (1-bit) channels

J. P. Collomosse is with the Department of Computer Science, University of Bath, U.K. {jpc@cs.bath.ac.uk}
T. Kindberg is with Hewlett-Packard Laboratories, Bristol, U.K. {timothy@hpl.hp.com}

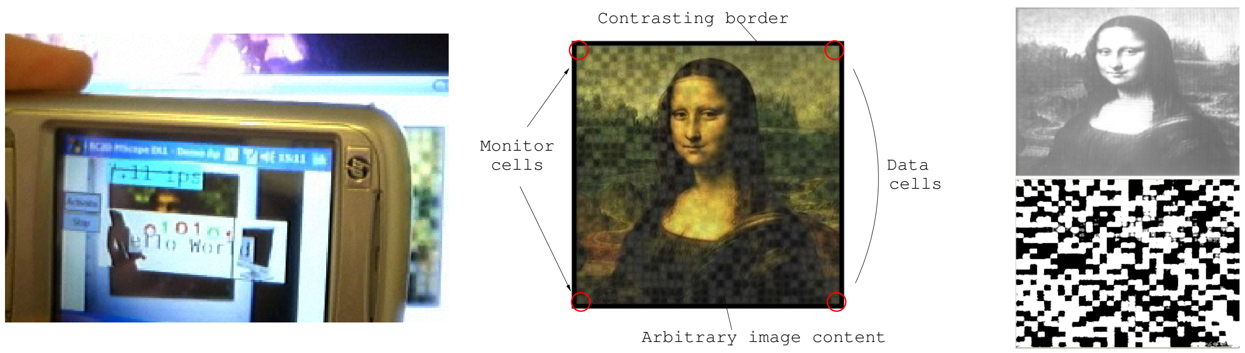


Fig. 1. Left: Typical use case for Screen codes; downloading data from a video display using a mobile camera. Middle: Single frame of a $32\times32$ Screen code, annotated to show key features. The image is divided into a regular grid; brightness fluctuations within each cell encode data. Four corner cells are reserved for error detection and synchronization. Right: A base-line image reconstructed over time (above), enabling recovery of luminance variation pattern (below).

have been implemented, both as a flickering screen region used to transmit data to a light sensor e.g. on Timex's Datalink watch[3] or Bloomberg's B-Unit, and by using a blinking LED to transmit data to a cameraphone [4]. However, 1-bit channels are too slow for most purposes: e.g. delaying the user by up to 48 seconds to reliably transmit a 30-character URL to a cameraphone running at 10 frames/sec.

We considered a naive approach to trading space for time, by distributing data among a repeating sequence of independent conventional mobile codes. However, most applications require code-reading to be quick and reliable, and the noisy 'screen to camera' transmission channel frustrates this. Data may be easily corrupted spatially, e.g. by adverse illumination and occlusions, or temporally due to sampling artefacts arising from differences in camera and display refresh rates. If a code in the sequence is missed, the time penalty is large: a single mis-read in a sequence of $n$ independent codes entails a delay of up to $n$ frames until the code is repeated at source. Saxena et al. [4] devised a way of transmitting data as a sequence of related frames. But they did not address the issue of dropped frames, and they achieved low data rates ($\sim 10$ bits/sec).

The foregoing techniques have limited aesthetic appeal. By contrast, algorithms for embedding data invisibly within visual media are common in steganography [5]. One such technique of note is the VEIL system[6] which encodes data in pairs of adjacent scan-lines within independent frames of video. VEIL has been used to embed information within television signals, e.g. in cartoons to trigger actions in toys. For large data volumes however, VEIL can be viewed as an adaptation of the aforementioned 'sequence of codes' approach, and similarly suffers from having no temporal error correction. Two further disadvantages are that users would not necessarily be able to distinguish ordinary media from that encoding data, and that data-carrying capacity is limited by scan rate.

## II. Overview of Screen Codes

A Screen code comprises a finite sequence of video frames, displayed in a continuous cycle. Data is encoded within an arbitrary image as a changing pattern of brightness fluctuations in a rectangular grid; this 'twinkling' makes the image recognisable to the user as a conveyer of electronic content. The fluctuations are passively observed by a camera-equipped mobile device over time, which is able to efficiently recover the data. Camera position and orientation may change during reading, as the user may move or shake the device. In view of the varying noise conditions in the visual channel, we have devised configurable error detection and correction schemes.

### A. *Anatomy of a Screen Code*

Each Screen code frame contains a user-selected source image, framed within a strongly contrasting 'quiet zone' (Figure 1, right). The source image is divided into a grid of 'cell' regions, whose dimensions remain constant over all frames. The luminosity of these cells are varied over time to encode data.

The raw data capacity of each code is therefore $N_c F_c \log_2(g)$ where $N_c$ and $F_c$ are the total data cell and frame counts, and $g$ specifies the number of discrete luminosity levels per cell. As we describe in subsection III-C, four 'monitor' cells per frame are reserved for the purpose of error detection; the remainder are used for encoding data ('data' cells). To aid exposition all our results here use $g = 2$; each cell is capable of encoding one bit. Larger values are possible, but in practice $g > 4$ levels are difficult to discriminate due to environmental noise. When $g = 2$, to encode a '1', the brightness of pixels in the cell is raised by addition of a constant luminosity component; to encode a '0', luminosity is lowered. The resulting discontinuities can be unduly emphasized by the human visual system ("Mach banding"). To guard against this we attenuate luminosity modulation towards the edges of each cell.

## III. Physical Layer – The Light Wire

By analogy with the OSI layered model for network protocols, we separate our exposition of Screen Codes into 'physical' and 'data' layers. The physical layer provides a mechanism to transmit raw bits (corresponding to cells) over the 'light wire' from display to camera. The data layer ensures robust transmission of identifiable units of data over this noisy channel. In this section we address the physical layer, describing the transmission process from the perspective of the receiving (camera) device.

### *A. Frame Registration and Baseline Image Recovery*

The Screen code is located by identifying the quadrilateral formed by the four strong edges of the quiet zone (Figure 1, middle). Contour-following algorithms [7] isolate connected edges in a dynamically thresholded [8] video frame. Intersecting edges of sufficient length are flagged as candidate Screen code corners, and permutations of these are searched to identify the largest quadrilateral covering the frame. We reduce search complexity by introducing constraints on corner proximity and orientation.

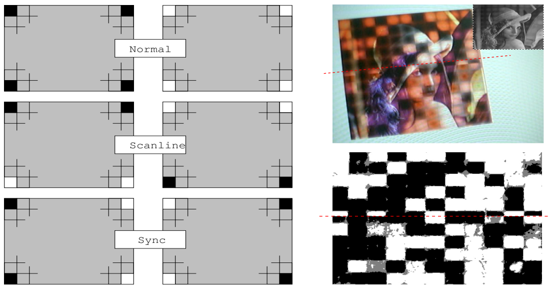
The region bounded by the quadrilateral is warped to a rectilinear basis of fixed size - so registering to a viewpoint invariant to camera position and distance, modulo rotation about the optical axis (Figure 1, right). To resolve this ambiguity we signal the presence of the top-left grid quadrant by boosting amplitude of the luminosity modulations. A compensatory rotation of the registered image is performed, if necessary, to preserve this property.

The initial few registered frames are analyzed to create a "base-line" image. The maximum and minimum values of each pixel over time are recorded and their mean value used to reconstruct a version of the Screen code image that exhibits no brightness fluctuations. This base-line image is subtracted from subsequent registered frames to yield a map of luminosity differences that encode data within a particular frame (Figure 1, right). The initial frames used to compute the baseline are not discarded, but are buffered and similarly processed against the baseline; so avoiding any 'preamble' delay.

### *B. Grid Sampling*

The periodic grid pattern creates strong peaks in frequency space, allowing the receiver to detect which of several preset grid configurations best describes the signal. For clear discrimination we found it convenient to preset grid sizes of $2^n \times 2^n$. After detecting grid resolution, we iterate over the baseline-subtracted image and extract a 2D grid of values from each cell. As discussed, under our two-state scheme a positive value indicates a '1'; negative a '0'. The value grid for each frame is stored in a buffer. If a grid is sampled that is near identical ($\pm 5\%$ error tolerance) of the latest frame in the buffer, it is ignored — this allows camera frame rates to exceed that of the display. Our decoding algorithm assumes that a cell will be observed at least once in each state (here '1' or '0'). To encourage this we add a pseudo-random pattern to data prior to transmission; later subtracted by the observer.

At this stage the buffer of sampled grids are ready for access by the 'Data Layer' (Section IV) for decoding. However these grids may be subject to spatial noise (corrupting values in the grid) or temporal noise (causing some grids to be dropped). Furthermore, no mechanism has yet been described to signal the start or end of the sequence. We address both issues next.



| Error Correction Configurations | | | | | |
|---|---|---|---|---|---|
| # | Spatial O/H | Occl. Tol. | Temporal O/H | F. Drop Tol. | Total O/H |
| 1 | 50% | $\sim$ 25% | (4,4,1)=0% | 0% | 50% |
| 2 | 40% | $\sim$ 20% | (6,4,2)=33% | $\sim$ 17% | 60% |
| 3 | 30% | $\sim$ 15% | (7,4,3)=43% | $\sim$ 29% | 60% |
| 4 | 10% | $\sim$ 5% | (8,4,4)=50% | $\sim$ 38% | 55% |

Fig. 2. Left: Monitor cell patterns, used to detect temporal sampling errors. Middle: Code imaged during a display refresh cycle (scan-line middle, in red). The grid is split into past/current states; the monitor cells indicate an error pattern. Right: Typical error correction configurations (#3); from strong temporal (#4) to strong spatial correction (#2). Columns 3 and 5 estimate tolerance to spatial (as % code area) and temporal (as % frames dropped) noise.

## *C. Error Detection and Synchronization*

Consider the transmission of a Screen Code animating at $C$ Hz on a display refreshing at $D$ Hz. The camera frame rate is $R$ Hz. The following error cases may arise:
**(a) Dropped Frames.** The Nyquist limits of the display and camera are $D/2$ and $R/2$ respectively. If $C > D/2$, $D > R/2$, or $C > R/2$ then the camera will likely fail to image a subset of frames in the Screen code, i.e. frames will be 'dropped'. Frames may also be dropped due to external factors such as mis-registration of a code, or the environment (e.g. obstructing the line of sight).
**(b) Partial Refresh.** Raster displays are refreshed at $D$ Hz from top to bottom by a scan-line sweep. Occasionally the camera samples the image during the display refresh; part of the display shows data from the previous code frame, the remainder from the current code frame (see Figure 2, middle).
**(c) Garbled frames.** Frames may become garbled due to the camera sampling in the 'idle' (dim) phase of the display refresh cycle, or environmental factors such as occlusion or specularities.

To detect these error cases, we adopt a signaling scheme using the four corner cells of the Screen Code which we dub 'monitor cells' (Figure 2, left). These cells also enable us to indicate the presence of the first and last frames in the code sequence; information useful later during decoding (subsection III-D).

With the exception of the first and last frames, the monitor cells in each frame are either set or reset as a group — toggling between these states on alternate frames (see Figure 2 'Normal'). If the top-left or top-right monitor cells do not match their bottom-left or right-counterparts, we deduce that a 'partial refresh' (case b) has been encountered and that the frame should be re-sampled (Figure 2, 'Scan-line'). To indicate the first or last frame, the 'Sync' patterns (Figure 2) are used. Again, mismatches between the top and bottom rows are used to detect case (b). The redundancy of signalling the sequence's start in both the final and initial frames robustifies that signal against corruption by noise.

Using the monitor cells we may detect case (a) when an odd number of frames are dropped; adjacent frames contain different data but identical monitor cell patterns. Unless the Nyquist limit is significantly exceeded (i.e. $C/R \gg 0.5$) it is likely that only single frames will be dropped due to temporal under-sampling. Although we cannot recover dropped frames we can appeal to the forward error correction mechanism of Section IV to reconstruct missing data. The monitor cells in the physical layer therefore enable us to detect (but not correct) data 'erasures' in the temporal domain. This knowledge almost doubles the data recovery capacity of the data layer (subsection IV-B). Undefined patterns of monitor cells are indicative of 'garbled frames' (case c), and also trigger frame re-sampling.

## *D. Frame Ordering*

By observing the monitor cells, and decoding the data cells within each frame, it is straightforward to deduce the length of the code sequence and to reorder frames in to their correct positions. Where dropped frames are detected, we record a temporal 'erasure'; information subsequently used for error correction. We do not spend bandwidth encoding frame sequencing numbers, or similar — these are rendered superfluous by our ability to detect both the sequence's start frame, and any dropped frames.

## IV. Data Layer — Spatiotemporal Error Correction

The physical layer provides us with a 'light wire' enabling the noisy transmission of a sequence of 2D grid patterns from display to camera. We interpret this data as a spatio-temporal volume ($x, y, t =$ time) with an additional flag at each $t$ denoting whether grid data is undefined due to temporal erasure (subsection III-C). The transmitting party must encode data robustly within this space-time cuboid. Conceptually this is achieved by accepting source data, applying some form of error correcting algorithm that appends parity bits to create a 'protected' bit sequence $E(i)$ (where $i$ is a bit index), and encoding that protected sequence within the space-time cuboid. That is, we require an invertible mapping of $E(i)$ into space $(x, y, t)$ via some transfer function $L$ such that $L(i) = (x, y, t)$. Conceptually, $L(.)$ is a space filling function of some form. In this section we describe our chosen $E(.)$ and $L(.)$ which are tailored to the particular noise characteristics of the 'light wire'. By distributing data over space and time, we reduce the likelihood of data corruption leading to retransmission. This in turn reduces the time to decode a Screen code (latency). Error correction overhead is configurable, allowing the transmitting party to trade robustness in space or time for bandwidth.

### A. *Our Error Correction Strategy*

In practical scenarios, the space-time cuboid exhibits much higher spatial resolution than temporal resolution. For example, each grid in the cuboid may have dimension $64 \times 64$ but the sequence may contain only 20 frames (e.g. 2 seconds at 10fps; a typical mobile camera frame rate). If environmental noise caused a portion of one frame to be mis-sampled, a small fraction of the raw bit stream may become corrupted. However, temporal noise causing one frame to be dropped by the physical layer would result in much greater data loss (here $\sim 5\%$). Mis-sampling in the spatial and temporal domain is caused by different phenomena and corrupts data to different degrees of magnitude. Our approach is therefore to specify a combined $E(.)$ and $L(.)$ that can cater separately for these different classes of noise — most importantly, mitigating against the large-scale data loss caused by dropped frames.

We divide each grid into a data and parity region, the former encoding data and the latter encoding error correction words that guard against data corruption. Reed-Solomon error correction is later used to generate (or verify) these parity words against data in the frame. The boundary between the data and parity regions is constant over time, thus the guarded data regions yield a spatio-temporal volume (channel), robust to spatial noise (e.g. occlusion), but not to temporal noise (e.g. dropped frames).

To encode data for transmission, bits in the data stream are first chunked into words; these words are transformed into the codewords of a linear binary code by dictionary look-up. Typically codewords are longer than the data they represent e.g. a 4 bit word might be represented by one of 16 transmission codewords, each 7 bits long. We can construct a 'dictionary' of such codewords, each codeword differing by a minimum Hamming distance of $n$ (in this example, codes can be devised up to $n = 3$ — a so-called (7,4,3) code). Such dictionaries are generated *a priori* by exhaustive search and shared between the sender and receiver. The codewords are striped across the guarded data volume in the temporal dimension. For example, in a 10 frame Screen code, if codewords are 7 bits long the first word might be sampled from (x,y,t) coordinates $(1, 1, 1-7)$, the second from $(1, 1, 8-10)$ then $(2, 1, 1-5)$ and so on to fill the spatio-temporal guarded data volume. Once the data volume is filled, we apply Reed-Solomon independently to each frame to compute its spatial error correction (parity) region. During decoding, spatial error correction is first applied to frames independently, and codewords then extracted from the spatio-temporal guarded volume and matched against the dictionary to look up the bit patterns they represent. These bit patterns are concatenated to recover the original data.

### B. *Error correction configuration*

We have used a linear binary code (LBC) to guard data in the temporal domain. LBCs are able to detect and correct up to $(k/2) - 1$ bits, where $k$ is the minimum Hamming distance between dictionary codewords. Due to our temporal sampling strategy a single bit error is analogous to a missing frame. However if we know the locations of such frames ('erasures') prior to decoding, we can correct for up to $k-1$ bits. Our 'monitor bit' temporal error detection technique (subsection III-C) provides us with exactly this information. Thus, in our above example of a $(7,4,3)$ binary code we are able to correct for 2 in every 7 frames (29% frame loss) without incurring increased decoding time due to data retransmission. Furthermore, within each frame we employ spatial error correction to compensate for partial misreads due to occlusion or noise. The levels of both temporal and spatial error correction can be independently set according to the deployment environment; Fig. 2 gives some example configurations we have found to work well. For example, in situations where fast data rates are desired, but spatial noise/occlusion is unlikely (e.g. the shop window or printer use cases of Section I), configuration #4 would be desirable. In the case of a public display imaged at a distance, spatial error correction would be more important (and depending on data capacity, may be tradeable for temporal error correction). Note also that our system degenerates to the naive 'sequence of 2D codes' example if all temporal error correction is removed (#1). A typical, balanced configuration is #3.

## V. Closing Discussion and Position

We have described Screen codes, a novel way of conveying data over a 'light wire' from a constrained display or region of a display, to a camera-equipped mobile device. The technique is robust to noisy channel conditions, and it is flexible with respect to trading off space for time, and in terms of configuring the level of error correction in each of these domains. A further trade-off of robustness vs. aesthetics can be made by controlling the amplitude of intensity fluctuations, or capacity vs. robustness by varying grid cell size. Screen codes have been implemented with on a laptop PC with a webcam (codes up to $48 \times 48$ sending data at $\sim 11kbit/sec$), and on an HP iPAQ rw6815 smartphone running Windows Mobile (codes up to $16 \times 16$ at $\sim 2.5kbit/sec$). We anticipate that the mobile data rate would improve with further code optimization (to improve frame rate), and improvements in optics (e.g. ability to focus, or increased video resolution). We have reported initial progress in evaluating the Screen code prototype under varying deployment conditions (Figure 2).

Screen codes are a novel form of 'visual hyperlink', and the position put forward by the authors is that, by placing visual hyperlinks on displays and not just in print, we have the potential to eradicate the disruptive 'content divide' that exists between fixed and mobile devices. Visual hyperlinks enable users to move content such as media samples, timetables and maps conveniently from a PC or public display to their mobile devices. Users could upload content acquired in their mobile lives back into the web on a PC, or to a device such as a printer, by reading a distinctive form of 'upload' visual hyperlink. The potential benefits of visual hyperlinks are widespread. Globally, there are an estimated $\sim 1.2$ billion camera phones in use (Lyra, 2007). These devices are evolving to provide the processing and imaging resources needed for visual hyperlinks. Continued engineering effort will be required, and more research into step-changing paradigms such as Screen codes will be necessary, to make these interactions truly smooth. But, as optics, resolution and algorithms improve, and with carefully designed feedback in the user interface, users will be able to achieve these interactions by casually pointing their camera phone broadly in the direction of the visual hyperlink and pressing 'go'.

## References

[1] "EAN/UPC bar code symbology specification," ISO/IEC Standard 15420:2000, 2000.

[2] "International symbology specification: Data matrix," ISO/IEC Standard 16022:2000, 2000.

[3] M. Jacobs and M. Insero, "Method and apparatus for downloading information from a controllable light source," US Patent 5,488,571, Jan. 1996.

[4] N. Saxena, J. Ekberg, K. Kostiainen, and N. Asokan, "Secure device pairing based on a visual channel," in *Proc. IEEE Symp. on Security and Privacy*, May 2006, pp. 1–6.

[5] E. T. Lin and E. J. Delp, "A review of data hiding in digital images," in *Proc. PICS'99*, Apr. 1999, pp. 274–278.

[6] D. Ciardullo, K. Kosbar, and C. Chupp, "Method for transmitting data on the viewable portion of a video signal," US Patent 6,094,228, July 2000.

[7] S. Suzuki and K. Abe, "Binary picture thinning by an iterative parallel implementation," *Pattern Recognition*, vol. 10, no. 3, pp. 297–307, 1987.

[8] R. Gonzalez and R. Woods, *Digital Image Processing, 2nd edn.*, Prentice-Hall, 2002, IBSN: 0201180758.