Subliminal: A System for Augmenting Images with Steganography

Juliet Bernstein **Josh Goodwin**

Abstract

Many methods exist for using a camera to capture digital information transmitted over the visible light spectrum. These methods often result in visually obvious patterns and codes that are displayed for a receiving camera to capture. We seek to encode information in images such that the code is distributed throughout the image and such that its presence is nearly invisible to the naked eye. In addition, we require our encoding scheme to be robust against common image degradations that are encountered when taking pictures with a mobile phone. An encoding and decoding scheme that achieves these properties is described. This scheme is tested via an Android application on the Nexus One phone and is shown to successfully decode the hidden information present in source images captured by the onboard camera.

Introduction

As smartphones with onboard cameras become pervasive in society, many opportunities arise for transmitting information using the smartphone's camera. The Google Android phone has a built-in barcode reader, and the Google Goggles application can read text as well as recognize some common objects.

Most common methods for visual information transport result in visually intrusive patterns. Familiar methods include one-dimensional barcodes and two-dimensional QR or Data Matrix codes. Newer techniques include Microsoft High Capacity Color Barcodes, and bokodes, which are constructed from an LED, a patterned mask, and a lens. All of these types of codes are localized in space and have the property of being "opaque"—that is, a specific area of the substrate must be devoted to displaying the code and nothing else.

An alternative approach to barcode-like systems is to distribute information throughout an image in a way that is "transparent" or "translucent" – either imperceptible or nonintrusive to the human eye. In addition, the encoding scheme should be robust to image degradations and transformations that are common when taking a picture of an object with a camera phone: cropping, rotation, translation, scaling, slight geometric distortion, occlusion, nonuniform lighting, color balance changes, imperfect display or printing, and image compression.

We select an encoding technique that is robust against these image degradations in addition to having the desired properties of being "translucent" and distributed. Information is encoded as a texture that is created by strategically placing delta functions in the Fourier domain. Then, in the spatial domain, the texture is modulated by a mask and added to the original image. Our encoding scheme is loosely based on the Acode technique by Joshua Smith.

Figure 1: An input image and an optional mask are used to create an encoded image.

Methods

Encoding

The inputs to our encoding system are an image to be encoded, a mask specifying where the encoded texture is to be added to the image, a string of bits to encode, and a set of frequency bands where the information is to be encoded. The image may be grayscale or color, and the mask may be binary or grayscale. The number of bits that it is possible to encode varies based on the modulation transfer function of the imaging system through which the encoded image will pass before it is decoded. The higher the frequency bands used to encode the bits, the more bits can be encoded and the less perceptible the encoded pattern will be to the human eye. In a test of our implementation using a laser printer and an Android camera phone, we encoded 24 bits in two frequency bands ($r_1 = 350$ and $r_2 = 450$ for a 1024 x 1024 image). However, we estimate that we could encode about 100 bits using this setup. Figure 1 shows an example of an input image and mask, as well as an output encoded image.

If the image to be encoded is in color (not grayscale or binary), it is first converted to HSV color space, and all operations on the image are done in on the value (lightness) channel. After the encoding process is complete, the encoded value channel it is recombined with the other color channels and the image is converted to the original color space.

The first step in the encoding process is to pre-filter the image to remove any spatial frequencies at the frequency bands were the delta functions (bits) are to be placed. Next, the encoded texture is created in the frequency domain. In our implementation, delta functions are placed at regular angular intervals at one of two frequencies: r_1 if the delta function is to represent a zero bit, r_2 if the delta function is to

represent a high bit. An important step in the encoding process is the inclusion of a beacon signal. The beacon allows the decoder to recover the degree of rotation of the recovered image, and therefore read out the bits starting at the correct location. Our beacon signal is comprised of a delta function at both the low and high radii, as well as a delta function placed halfway between those radii. After the encoded texture has been created in the frequency domain, it is transformed back to the spatial domain, modulated by the mask, and added to the image. The encoding process is illustrated in Figure 2.

This encoding technique is compatible with many methods for error correction and detection. However, there is one caveat about which bit strings may be encoded; if only two frequency bands are used for encoding, the decoder may have a difficult time distinguishing a bit string that is all 0s from a bit string that is all 1s. Therefore, in our implementation, we used an error correcting scheme that involved replicating each bit twice and inverting half of the resulting bits.

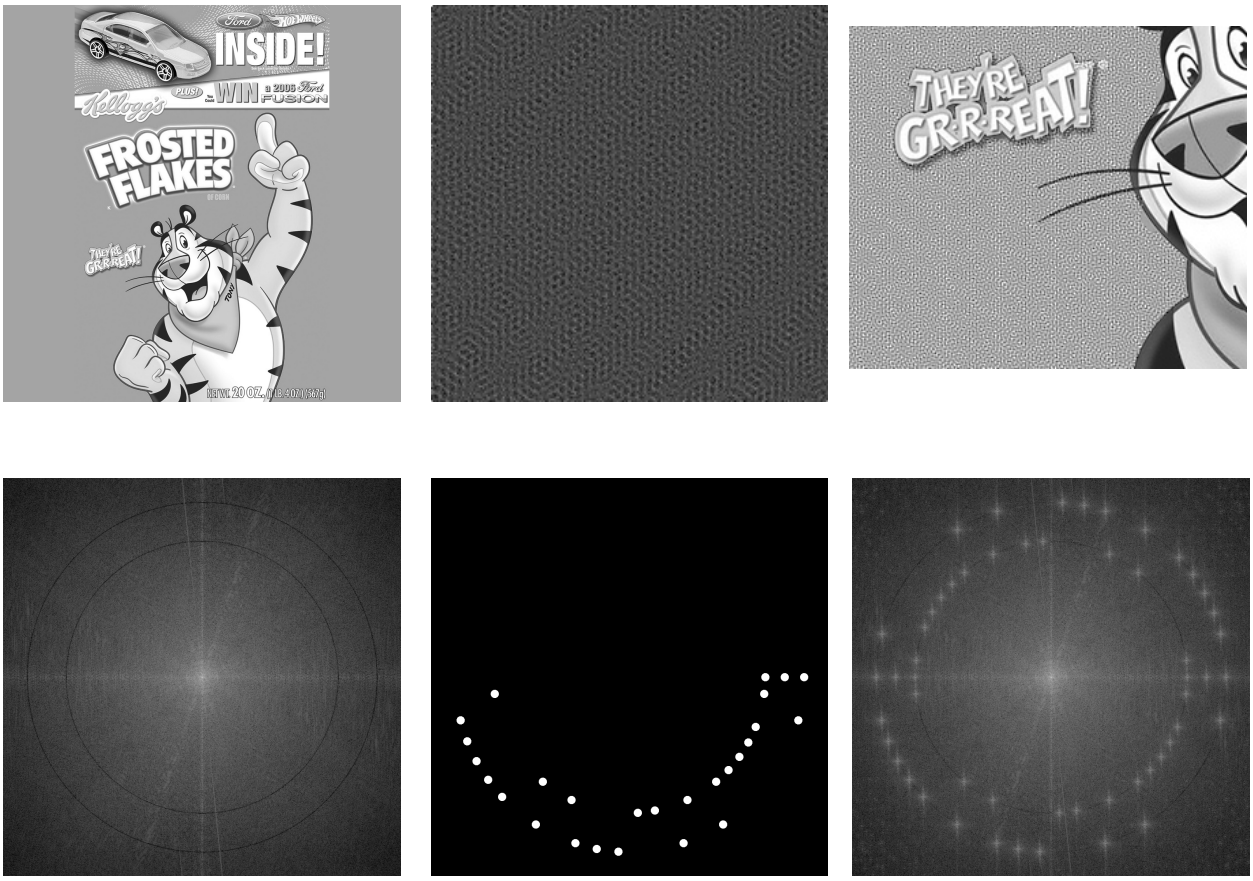

Figure 2: Top Row: Spatial Domain representations of the input image that has been pre-filtered, the encoded texture, and the encoded image (cropped). Bottom Row: Frequency Domain representations of the same images. The Fourier transform of the encoded texture has been enhanced to show detail.

Imaging System

The methods of image display and capture that take place between the encoding and decoding steps affect the range of spatial frequencies that may be used to encode information. In our implementation, we used a Xerox Phaser 6200 color laser printer to print images on white office paper, and a Nexus One Android phone camera to capture images. We used an r_2 value that was near the upper limit of the spatial frequencies that the printer could produce. The Nexus One camera has a 5 megapixel sensor and autofocus from 6 cm to infinity. Our system works if the camera is about 6-20 cm away from the printed paper when the image is captured. After image capture, the image is JPEG compressed with a quality factor of 50. Figure 3 shows example encoded images captured by the Nexus One camera.

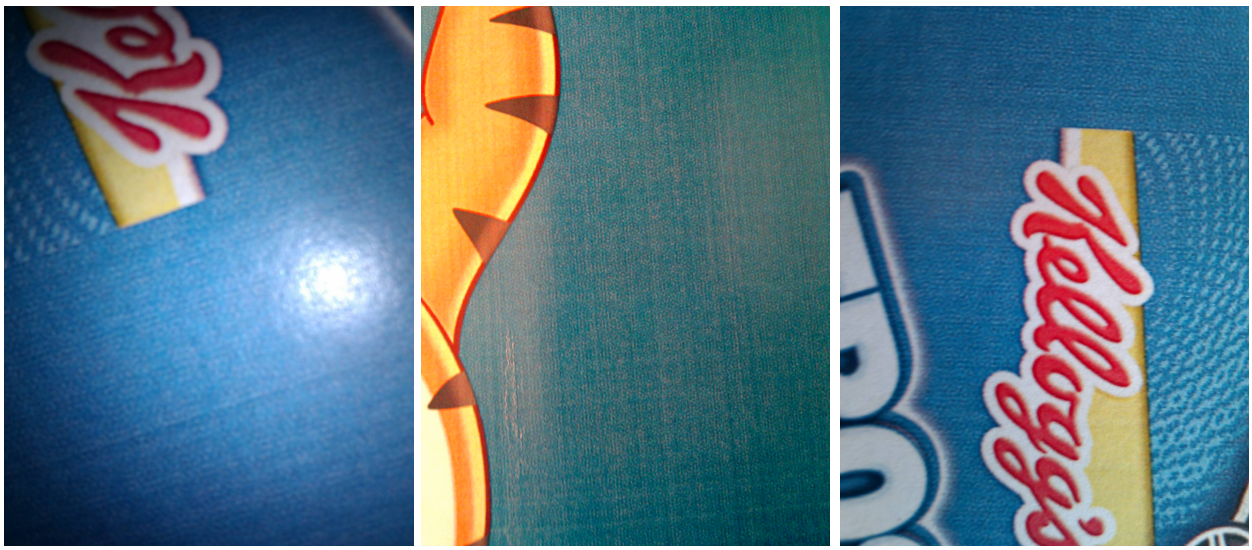

Figure 3: Example images taken by the Nexus One camera demonstrating rotation, translation, cropping, lighting non-uniformity, occlusion, imperfect printing, and color balance issues.

Decoding

The decoding system uses image processing and computer vision techniques to read the bits from the captured image. The captured image is converted to grayscale and cropped so that it is square. Next, a Hamming appodization window is applied to reduce the effect of hard edges in the image on the Fourier transform of the image. In the frequency domain, a mask is applied to remove very low and very high frequencies in the Fourier transform, and the transform is thresholded. These masks and thresholds must be set on a per-application basis. Finally, scale and rotation are detected. The algorithm maximizes overlap between the thresholded transform and two masks: one to detect scale, which is a set of band-pass filters which are resized until maximum overlap is found, and one to detect rotation, which masks out radial spokes and is rotated until maximum overlap is found. Bits can be read out by applying masks to select for each angle (wedge of the spectrum) and radius (frequency band), and comparing the total brightness of each of these regions. The beacon is detected as a wedge of the spectrum that contains the highest total brightness of all of the band pass masks multiplied together. Once the beacon is found,

the string of detected bits is shifted so that bits can me read out starting at the correct location. Figure 4 illustrates the decoding process, as well as the masks applied to detect rotation and scaling.

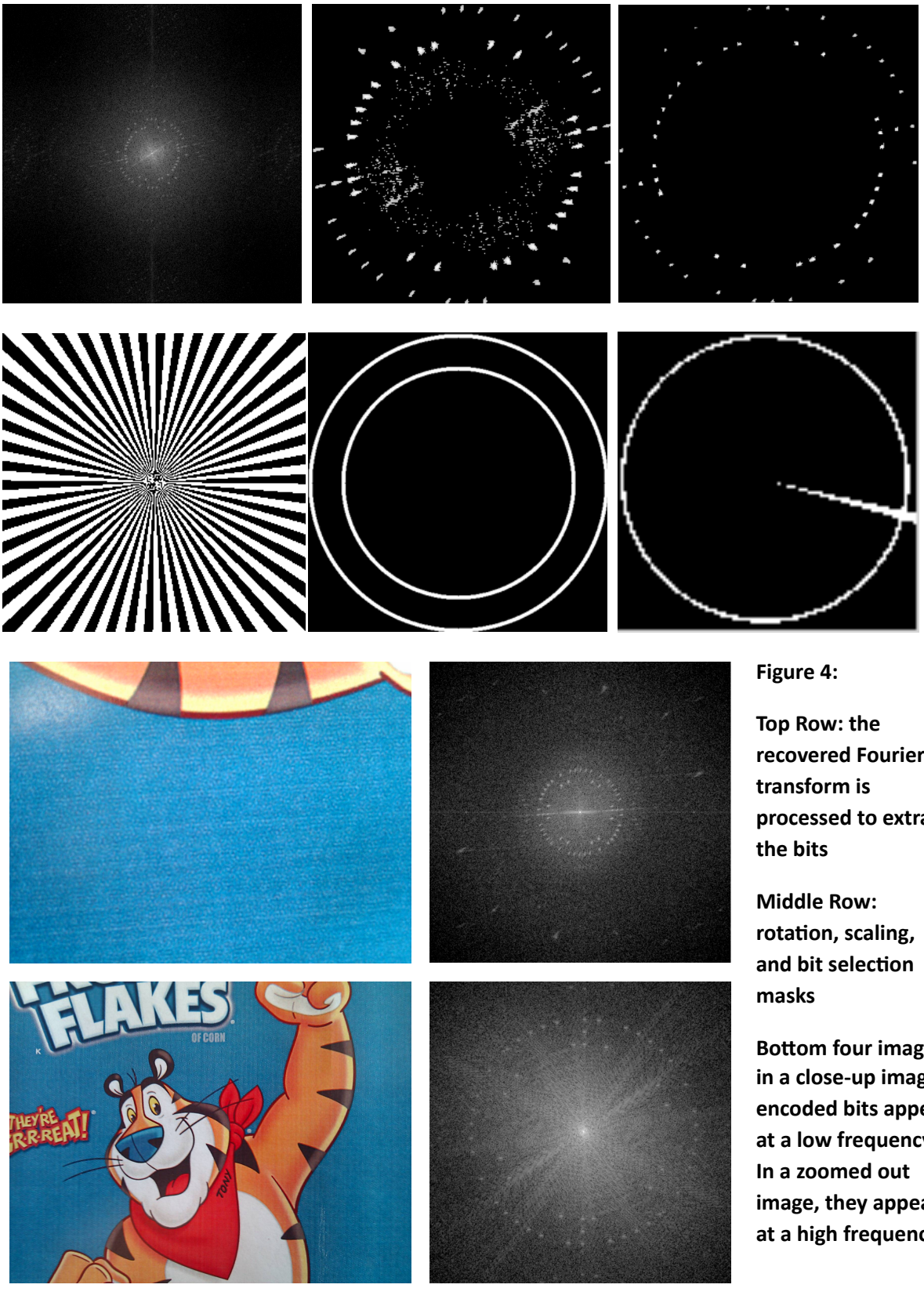

Figure 4:

Top Row: the recovered Fourier transform is processed to extract the bits

Middle Row: rotation, scaling, and bit selection masks

Bottom four images: in a close-up image, encoded bits appear at a low frequency. In a zoomed out image, they appear at a high frequency.

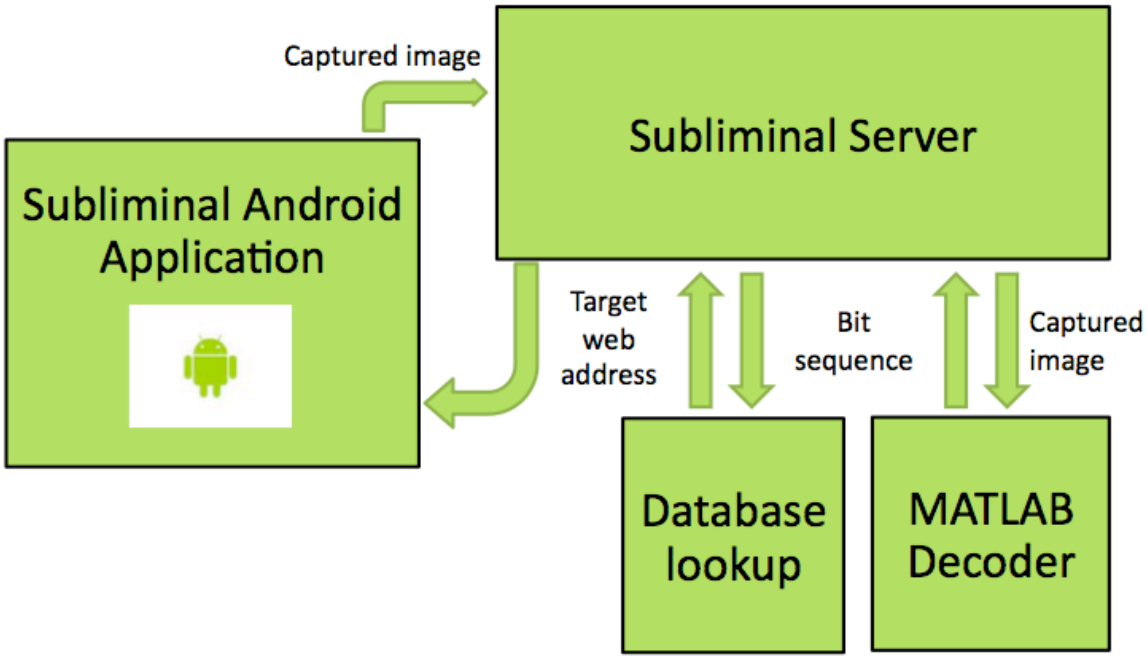

Figure 5: Subliminal system architecture

System Architecture

In order to test the application of this encoding scheme in a real-world scenario, a system was constructed based on using the Nexus One smart phone as a capture device. We named this system *Subliminal*. Smart phones with onboard cameras are becoming common place among much of society, and enabling Subliminal on such devices would permit the placement of hidden information anywhere a user could capture it with their smart phone. The resulting decoded information can then be immediately displayed to the user.

In order to accomplish this, several components were needed. First, we set up a remote server with which Subliminal clients would be able to communicate in order to decode source images. This server was created on the departmental cubist.cs.washington.edu host, using the Ruby on Rails development framework. In addition, a mySQL database was created and integrated with the server that provides maps from smaller integer keys to larger text values. Doing so enables Subliminal to encode a minimal number of bits in the source image without limiting the amount of resultant decoded information.

Second, the actual decoding MATLAB script was integrated with the server, so that an input source image could be decoded to a resulting integer value. Lastly, an Android application was developed using the Android SDK from Google. This application captures a source image and automatically uploads the image to the Subliminal server, which decodes the image into a key and returns the resulting value from the database. At the moment the resulting values are target web pages, at which point the Android application automatically launches a browser and loads the target address.

Results

Encoding and Decoding Results

Examples of encoded images and their recovered Fourier transforms are shown in Figure 4. The decoding algorithm is almost always able to interpret the bits correctly if the bits can be visually interpreted by a human from this Fourier transform. However, the parameters of the decoder must to be tuned for various operating ranges, including expected range of distances of the camera from the substrate.

The decoder currently takes about 2 seconds to process an image in MATLAB, but varies based on the size of the image and the number of bits encoded. This code is not at all optimized and could be made to run much faster.

System Implementation Results

While functional, the overall framework could be improved in many ways. First, capturing an in-focus image to decode is essential in recovering the correct encoded bits. In manual testing, the Android auto-focus feature used when taking pictures with the default camera application seemed to be rather robust in consistently acquiring in-focus images. However, the Subliminal calls to the camera API, though using the autofocus feature, seem to be less robust. We presume that greater experience with the android API would solve this, as the hardware seems to be very capable of consistently producing in-focus images.

In addition, the latency of the overall process is currently around 25 seconds. Much of this is due to inefficiency in connecting the various components of the backend system. For example, MATLAB is unfortunately not installed on cubist, but is available on abstract. As such, in the current implementation after receiving an image from a client, the server transfers that image *again* using scp to abstract, where the actual decoding script is run (re-loading and launching the MATLAB framework with each call) and the resulting key is passed back to the cubist server. The decoding script itself is also not optimized and could almost certainly be improved in regards to running time. Future implementations could certainly significantly decrease the latency of the complete capture-to-response time.

Applications

We believe that Subliminal and similar systems would be useful in a number of potential applications:

Aesthetically Pleasing Barcode Replacement: Subliminal could function as a barcode replacement in situations where its properties are desired over those of barcodes. For example, on small or artistically designed materials, such as book covers, a Subliminal code could replace the traditional barcode. In this situation, the Subliminal code would probably not be distributed over the entire material, but localized in the same way a barcode is. The goal in this case is a less visually intrusive barcode.

Improved Cash Register Checkout Speed: With traditional localized barcodes, a cashier must orient each product so that its barcode may be scanned. Using Subliminal, this process could be done automatically, since the products would not need to be in any particular physical orientation for the labels to be read.

Product Labels for Contests, Advertising, and URL Distribution: Companies could print hidden images in their labels, with a chance to win a prize if the user takes a picture of the label with a smartphone.

Improving Object Visibility for Computer Vision Systems: As computer vision and augmented reality systems become more popular in the coming years, a need will arise to tag common objects with meta-information for recognition.

Non-Human-Readable Barcode for Security: Barcodes, by nature, are readable by humans. For some secure applications, it might be useful to have a barcode that is not interpretable by humans.

Watermarking of Printed Materials: A system like Subliminal could be used to watermark valuable printed materials, such as currency, cheques, tickets, and stamps.

ID Card Scanner: Subliminal codes could be used to overlay data, such as a social security number, onto an image of a face on an ID card.

Localized Codes on Large Objects: Different codes could be embedded on different parts of a large object, such as a poster, assisting an augmented reality system in localization or orientation.

Future work

One potential area of future work would be to automatically decode video streams as well as still images. Each frame in a video stream could have hidden information, and a receiving camera could continuously capture frames until it successfully retrieved the hidden information. In addition, protocols could be implemented to transmit more bits than are encoded in a single frame as one message, enabling arbitrary length data transfers using hidden information.

Conclusions

Subliminal, while not completely invisible in the sense of comparing before encoding and after encoding images, still successfully encodes information that the naked eye would simply perceive as a (most likely intentional) background texture. We feel that this degree of "invisibility" is acceptable for a wide range of target applications, with product labels being one tested example. In addition, further work with encoding/decoding parameters and different display/capture scenarios could further refine the degree of invisibility attained while still retaining robust and correct decoding. Subliminal achieves its target goal, and does so while still indicating great room for improvement.

References

[1] J. Smith, "Data Encoding and Decoding Using Angular Symbology," WO Patent Application 03012727, Published Feb. 13 2003.

[2] C. Honsinger and M. Rabbani, "Data Embedding Using Phase Dispersion," Eastman Kodak Company, 2000.

[3] A. Mohan et al., "Bokode: Imperceptible Visual Tags for Camera Based Interaction from a Distance," SIGGRAPH 2009.

[4] Google Goggles: http://www.google.com/mobile/goggles/#text