Visual Code Marker Detection

David Varodayan and Kunal Ghosh
Department of Electrical Engineering
Stanford University, Stanford, CA 94305
Email: {varodayan, kghosh}@stanford.edu

***Abstract*— An image processing algorithm to detect visual code markers and extract the embedded data is presented. The algorithm uses an adaptive thresholding technique to produce a binary image with an accurate representation of each visual code marker from an input RGB image. Visual code marker detection is performed on the binary image using a series of tests that leverage prior knowledge of the distinctive features of a visual code marker, namely, the long and short guide bars and three cornerstones. Embedded data bits are subsequently extracted using a projective transformation matrix that maps the code plane to the image plane and prior knowledge of the data area in the code plane. The algorithm is able to detect visual code markers in an image, correctly identify the origin coordinates, and accurately extract the data in each visual code marker under challenging imaging conditions.**

I. Introduction

The emergence and rapid growth in consumer digital imaging systems, especially those found in mobile imaging devices such as cellphone cameras, are enabling a plethora of new applications. One such application is the detection of visual code markers and the subsequent processing of embedded data using conventional cellphone cameras. Visual code markers are 2D code tags placed on objects that can be used to embed object information, similar to barcodes but richer, thus acting as gateways between the consumer and a virtual world centered around the physical object. Assuming the availability of a CCD or CMOS image sensor for image capture followed by a DSP block integrated into a cellphone camera module, an image processing algorithm for such visual code marker detection and processing was proposed in [1]. In this paper, we use a similar approach to design and implement an efficient and robust visual code marker detection algorithm.

Our algorithm takes a color image acquired by a cellphone camera as input. Pre-processing of the image includes conversion of the RGB image to an intensity image, followed by conversion to binary using adaptive thresholding. The binary image is then used to detect the number of visual code markers and the location of each marker in the image plane. A projective transformation using the image coordinates provides a mapping between the image and code planes for each visual code marker that is used to read the embedded data. The outputs from our algorithm are the image coordinates of the origins and embedded data bits for each visual code marker detected in the input image.

We have tested our algorithm on images captured using different camera modules and under different imaging conditions. We find that the adaptive thresholding is tolerant to varying scene illumination and produces accurate binary representations of each visual code marker. Assuming that the code markers have a uniform white border, we accurately detect each code marker and its corner coordinates. Finally, we correctly read the binary data sequence embedded in each visual code marker.

II. Image Processing Algorithm

Our algorithm consists of three main modules: A) Pre-processing of the RGB image; B) Visual code marker detection; and C) Extraction of embedded data. Fig. 1 shows a typical input image which will be used for purposes of illustration as we discuss the algorithm, and the visual code marker definition and layout following that provided in [1].

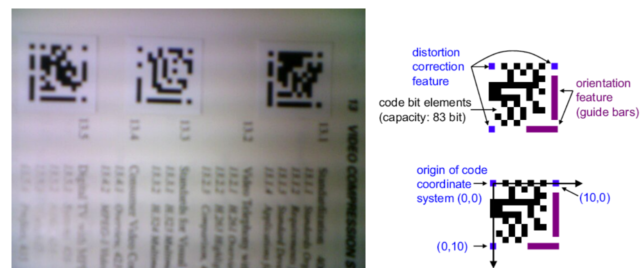

Fig. 1. Typical input image with three visual code markers (left) and visual code marker definition and layout (right)

A. Pre-processing

Our objective in the pre-processing steps is to ultimately produce a binary image from the input RGB image that has an accurate representation of each visual code marker in the image. The motivation to do so is two-fold. Firstly, a binary image allows representation of each image pixel using 1 bit, a dramatic reduction in data from the 24 bits (assuming 8 bit representations for each color channel) required for each pixel in the original image, thus decreasing computation time in the subsequent steps. Since the visual code markers are binary, we clearly do not lose any relevant information in the process as long as we do not incorrectly threshold. Secondly, binarization of the image allows us to design a simple, computationally efficient algorithm to read the embedded data bits in the final stage since the binary sequence can be readily determined by finding the logical value of pixels in the image plane corresponding to the data area in the code plane.

We use an adaptive thresholding technique for converting the luminance component of the RGB image (the grayscale image) to binary that compensates for uneven camera brightness (poor contrast), varying scene illumination and image noise. Adaptive thresholding is essentially a local thresholding algorithm that computes a separate threshold for each pixel based on the neighborhood of the pixel. Note that the approach is fundamentally different from a conventional global thresholding algorithm (e.g., Otsu's method) that uses a single threshold for all the pixels in the image. The challenge in global thresholding algorithms is in finding the correct threshold level for all pixels in the image that accurately turns the dark pixels to black, and the light pixels to white. Hence, global thresholding methods are prone to inaccuracies when images are underexposed or overexposed, illumination varies across the scene, or distinct histogram peaks are obscured by image noise. Since our input images are taken under varying illumination levels, and thresholding accuracy is important to ensure data bits in the visual code marker are not inadvertently flipped, we cannot rely on such global thresholding methods.

Our adaptive thresholding technique is a modified version of that presented in [2]. The idea of the algorithm presented in that work is to run through the image while calculating a moving average of the last n pixels, and when the value of a pixel is significantly lower than this average, it is thresholded to black (white otherwise). Our modified algorithm uses a 2D low-pass filter to compute the local average which we then use as the local thresholding level. That is, instead of scanning the image in a raster-like fashion to compute a moving average as proposed in [2], we simply compare the original grayscale image to the blurred grayscale image, where each pixel in the blurred image is a weighted local average of the pixel neighborhood. This modified approach is unbiased towards the orientation of illumination, which the raster-scanning approach often suffers from. In our implementation, we use a filter with exponentially decaying weights and spatial extent that is sufficient to compute local averages of regions with uniform illumination. We apply a thresholding factor of 0.9 to each pixel thresholding level for tighter accuracy during thresholding. Finally, we morphologically open and then close the thresholded image using a 3x3 pixel cross-shaped structuring element to remove small regions before subsequent region labeling. Fig. 2 shows the sample image in Fig. 1 after the above pre-processing steps.

B. Visual code marker identification

Identification of visual code markers relies on their common features: the two guide bars and three cornerstones shown in Fig. 1. As suggested in [1], we pursue a region-based approach to locate these features.

We now summarize our procedure, before delving into more detail. First the black pixels in the binarized image are labeled by region. From this set of regions, we create subsets of long and short guide bar candidates according to each region's size and shape. Then each pair of long and short guide bar candidates is evaluated based on relative size and position.

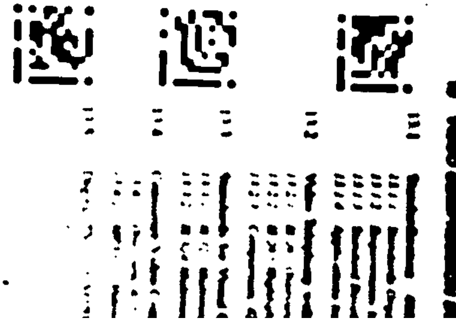

Fig. 2. Typical input image after thresholding

For those pairs of guide bars that remain potential visual code markers, we search for appropriately-sized cornerstone regions in the three expected corner positions. If all three cornerstones are found, a visual code marker has been identified. The coordinates of the centroids of the cornerstones and the short guide bar are calculated for use in the next part of the algorithm.

The initial labeling of black pixels according to region is performed by a two-pass region-labeling algorithm for 8-connected regions.

From among these regions, we find candidates for long and short guide bars as follows. Regions of less than 75 pixels are ignored. Whereas the algorithm in [1] fits an ellipse to each region, we instead apply the Hough transform to each of the remaining regions. We first find its orientation θ, which we define to be the angle along which the region is narrowest. The region's length l and width w are calculated as its maximum extent in directions parallel and perpendicular to θ, respectively. We also compute a parameter ρ for each region that defines the line $x \cos \theta + y \sin \theta = \rho$ through its centroid in the direction of θ. A region is designated a long or short guide bar candidate if its length-to-width ratio lies between 3 and 8 or between 2 and 6, respectively. Our cutoffs are quite loose to allow identification of guide bars distorted by perspective. Figs 3 and 4 show the long and short guide bar candidates for the sample image in Fig. 1, respectively.

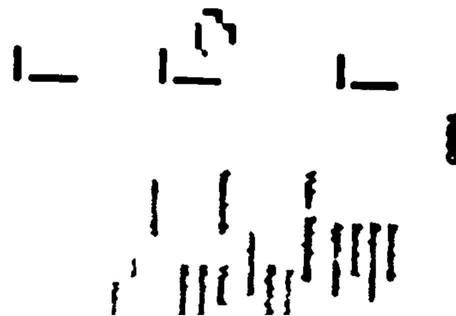

Fig. 3. Long guide bar candidates of typical input image

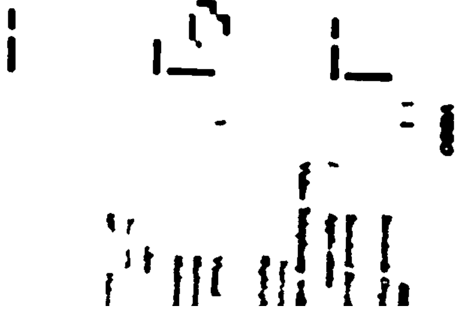

Fig. 4. Short guide bar candidates of typical input image

Fig. 5. Identified guide bars and cornerstones with marked centroids of a typical input image

Next we obtain pairs of long and short guide bars that form potential visual code markers by applying a sequence of tests. We reject pairs of guide bars whose orientations differ by less than $50°$ or more than $130°$, or if one guide bar is more than 50% wider than the other. We compute the point of intersection of the lines $x \cos\theta + y \sin\theta = \rho$ of the two guide bars and verify that it lies within 3 pixels horizontally or vertically of the short guide bar. We compare the point of intersection to the endpoints of the guide bars along their respective lines $x \cos\theta + y \sin\theta = \rho$. If the ratio of the distances from the point of intersection to the endpoints of the long guide bar is less than 3 or more than 10, the pair is rejected. As is the case if the ratio of the distances from point of intersection to the endpoints of the short guide bar is less than 5. This sequence of tests whittles down the candidates for visual code markers to pairs of guide bars with appropriate relative size and position. As before, the cutoffs are selected to be loose so that visual code markers distorted by perspective are not inadvertently rejected.

The prediction of cornerstone locations for each visual code marker candidate is based on the guide bars' orientations, lengths and their point of intersection. Starting at each of the three predictions, we begin a pixel-wise search for the respective cornerstone around a square spiral. The first region that is no more than 50% wider than the guide bars in the directions perpendicular to their orientations is denoted the cornerstone. If all three cornerstones are identified within 15 pixels of the original predicted locations, a visual code marker has been identified.

Finally, the centroids of the three cornerstones and the short guide bar are computed for use in extraction of the visual code marker's embedded data. Fig. 5 shows these regions with centroids marked as white points for the sample image in Fig. 1.

C. Extraction of embedded data

The inputs to the last stage of the algorithm, reading the embedded data bits, are four pairs of coordinates for each visual code marker detected. We follow the method outlined in [1] for extracting the embedded data due to its computational efficiency and robust results. The procedure essentially involves computing a projective transformation from the code plane to the image plane from the four pairs of image plane coordinates, and then running through the known data area in the code plane and testing for the logical state of the corresponding code plane pixel in the image plane. Since our image here is the thresholded image obtained using the steps detailed in section A above, we can directly test whether a corresponding code plane pixel in the image is black (bit 1) or white (bit 0) and generate the binary sequence for the 83 bits in the data area of each visual code marker.

A projective transformation matrix exists between the code plane and the image plane since the code elements are coplanar. The four correspondence points that we use in computing the projective transformation matrix are the centers of the three cornerstones and the center of the second guide bar. We reproduce below the derivation provided in [1] for the projective transformation matrix:

Code element	Image coordinate	Code coordinate
upper left cornerstone	(x_0, y_0)	$(0, 0)$
upper right cornerstone	(x_1, y_1)	$(10, 0)$
second guide bar	(x_2, y_2)	$(8, 10)$
lower left cornerstone	(x_3, y_3)	$(0, 10)$

Code coordinate (u, v) is mapped to image coordinate (x, y) with

$$x = \frac{au + bv + 10c}{gu + hv + 10}, \quad y = \frac{du + ev + 10f}{gu + hv + 10}.$$

The parameters a to h are calculated from the four reference points (x_i, y_i), $i \in \{0, \dots, 3\}$, as follows:

$$\Delta x_1 = x_1 - x_2, \quad \Delta y_1 = y_1 - y_2,$$
$$\Delta x_2 = x_3 - x_2, \quad \Delta y_2 = y_3 - y_2$$

$$\Sigma x = 0.8x_0 - 0.8x_1 + x_2 - x_3,$$
$$\Sigma y = 0.8y_0 - 0.8y_1 + y_2 - y_3$$

$$g = \frac{\Sigma x \Delta y_2 - \Sigma y \Delta x_2}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} \quad h = \frac{\Sigma y \Delta x_1 - \Sigma x \Delta y_1}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2}$$
$$a = x_1 - x_0 + gx_1 \quad d = y_1 - y_0 + gy_1$$
$$b = x_3 - x_0 + hx_3 \quad e = y_3 - y_0 + hy_3$$
$$c = x_0 \quad f = y_0$$

For each code plane element (u, v) in a detected visual code marker, we find the corresponding image plane pixel (x, y) using the above equations. Since we know the data area for each visual code marker as defined in Fig. 1, we are able to traverse through the code plane for each marker and find the corresponding pixel in the image for each code plane element. We then test each pixel for its logical state, black representing 1 and white representing 0, and read the data into an 83-element array.

III. Results

Our visual code marker detection and processing algorithm performs perfectly on the training image set provided in the problem statement. All bits from all visual code markers are read without error and there are no repeats or false positives. Each training image takes on average 5 seconds to process on a SCIEN lab machine.

We also performed tests on several images of our own creation such as those shown in Fig. 6 The algorithm proves robust to various illumination conditions and many visual code marker placement geometries. The only failures occur when a visual code marker is at too steep an angle to the image plane or when its white borders are cut too thin. In both cases, the cornerstones and/or guide bars lose their distinctness as individual regions by merging into surrounding regions. This impedes the region-based approach to identification of the visual code markers. A very different strategy would be needed to handle such difficult cases. We should bear in mind that these situations are perhaps unimportant for practical applications.

Fig. 6. Test images with varying scene illumination and complex visual code marker placement

IV. Conclusion

We have designed and implemented a visual code marker detection and processing algorithm that successfully detects visual code markers and correctly reads the embedded data bits in each training image given. The success of the visual code marker detection and data bit extraction depends on accurate thresholding of the input image. We find that our adaptive thresholding technique is tolerant to varying scene illumination levels and different camera modules and does not inadvertently flip visual code marker data bits. We are able to correctly detect code markers and their image coordinate locations assuming each code marker is surrounded by a white border, as per the training set. Once a visual code marker is detected and its corner coordinates are identified, we can always read the embedded data bits correctly from a binary image provided that the thresholding process is accurate. We have tested our algorithm on our own training set and have found it to be robust to challenging imaging conditions. Our image processing algorithm takes an average of 5 seconds for each input image running on a SCIEN lab machine.

V. Appendix

Distribution of work:

- Section A (pre-processing): Kunal and David
- Section B (visual code marker identification): David
- Section C (reading embedded bits): Kunal
- Generating new training set, testing, optimizing algorithm and writing report: Kunal and David

References

[1] M. Rohs, "Real-world interaction with camera-phones," in *2nd International Symposium on Ubiquitous Computing Systems (UCS 2004)*, Tokyo, Japan, 2004.

[2] P. D. Wellner, "Adaptive thresholding for the digitaldesk," *Xerox Technical Report*, vol. EPC-1993-110, 1993.