Detection of Visual Code Markers

Gabriel Takacs

***Abstract*—This paper presents a method for entering digital data into a cell-phone via a visual code marker. The code is presented as a two-dimensional array of black and white bits along with position and orientation features. The algorithm works by searching for properly shaped and spaced regions in an adaptively-thresholded image to locate the code data. The data are then read by computing the mapping between image coordinates and code coordinates. The method presented has a low error rate and has proven to be much faster than template matching methods.**

***Index Terms*—camera, cell-phone, code, detection**

I. Introduction

CELLULAR phones with built-in cameras are nearly ubiquitous in many societies around the world. These phones have wireless connectivity, visual and audio inputs, keypad entry, visual and audio outputs, and capable processors. Because of the prevalence and capability of such camera-phones they have the potential to be a powerful tool for linking the virtual-world and the real-world. Unfortunately, the only way that a user can enter digital data into the phone is via a small keypad. Visual code markers can be used as another quick and easy method of data entry by reading data presented to the camera's visual field. This paper describes one possible implementation of digital data entry through a low-quality camera.

II. Visual Code Markers

To extract digital data through an image the data must be presented to the camera in a structure that can be easily processed. Ease of processing, and density of data are both desirable features in the presentation format. To serve both goals simultaneously Michael Rohs has suggested a square marker layout with data in a two dimensional array.

Figure 1: Sample code marker (left), Sample code marker with non-datum elements shown in color.

Figure 1 illustrates the data marker layout. The code on the left shows the true color scheme of black and white. The code on the right has false colors on the elements which do not contain data. These non-data elements are used for finding the marker in a cluttered visual field, and then determining its orientation and position. The names by which these elements will be called in this paper are as follows: red is the top-left cornerstone; blue is the bottom-left cornerstone; magenta is the top-right cornerstone; green is the right guide-bar; and cyan element is the bottom-right guide-bar. Each non-data element has a one code-pixel white border between it and the data pixels. There are eighty-three bits of datum in marker.

III. Code Detection Algorithm

The method for reading visual code markers that is presented in this paper is based on the geometry of regions in a binary image; it will be referred to as region-geometry identification (RGI). The algorithm functions in three major steps, region extraction, marker location, and datum extraction. In region extraction a color image is processed to obtain an image with uniform background and labeled contiguous foreground regions. Marker location is the process of selecting groupings of regions which represent code markers. The last state, datum extraction, reads the data from the code marker. Each of these steps is explained in detail below. The algorithm is illustrated with a sample image.

A. Region Extraction

The region extraction step of the algorithm is applied globally to the entire image and is only applied once. This step culminates in a list of labeled regions which are indexed by code location step. Region extraction drastically reduce the dimensionality of the processing, thereby increasing the speed of the algorithm.

1) Grayscale conversion

The color information in the original image (Figure 2) is not useful for code detection since the white-balance of the image is unknown. An intensity image is calculated as suggested by Rohs as the average of the red and green channels. This method helps smooth the noise since the blue channel has the most noise. However, the RGI algorithm is not very sensitive to the method used to discard color information. Figure 3 shows the grayscale image.

Figure 2: Color input image

Figure 3: Average of red and green channels

2) Adaptive thresholding

Proper creation of a binary image is critical to the RGI algorithm. A global threshold will erase a code marker if the marker is in bright illumination or shadow relative to the mean image intensity. To resolve this problem the gray level is compared to a local threshold. The threshold for a given point is the average of the pixels in a window around the point. By selecting the size of the window properly gradual intensity variations are ignored, whereas local intensity changes are captured.

Figure 4: Adaptively thresholded binary image

Figure 4 shows the results of an adaptive threshold operation on Figure 3. The effects of the adaptive algorithm are not apparent as the sample image does not suffer from lighting variations.

3) Region Labeling

Each contiguous black region in the binary image must be individually labeled in order to calculate the properties of each region individually. Any algorithm which labels the regions will work, as the ordering of the labels is of no import to the RGI algorithm. This implementation uses the MATLAB `bwlabel` function.

Figure 5: Labeled regions, colors represent label number

The labels that result from this step are illustrated in false color in Figure 5. The colors are just for visualization, the label image consists of a single channel.

B. Code Location

The code location step is the key to the RGI algorithm. Once black regions are labeled, the algorithm need only iterate over the regions and not all pixels in the image. For each black region, a series of successive tests are performed to check if it is the right guide-bar of a code marker. The algorithm skips a region as soon as one of the tests fails. Regions that are very dissimilar from the true marker shape therefore require little computational effort to distinguish.

For each region the height, width, centroid, eccentricity, and number of pixels is computed. First the number of pixels in the region is counted, then the centroid is calculated by averaging the x and y coordinates of each pixel in the region. The centroid is then used in calculating the second moments of the region, as suggested by Rohs. The second moments are the defined in Equations 1.

$$
\begin{aligned}
&(\bar{x}, \bar{y}) = \text{centroid position} \\
&\mu_{xx} = \frac{1}{|R|} \sum_{(x,y)\in R} (x-\bar{x})^2 \\
&\mu_{yy} = \frac{1}{|R|} \sum_{(x,y)\in R} (y-\bar{y})^2 \\
&\mu_{xy} = \frac{1}{|R|} \sum_{(x,y)\in R} (x-\bar{x})(y-\bar{y})
\end{aligned}
\tag{1}
$$

Once the second moments are known an ellipse, E, can be fit to the region. The orientation and eccentricity of the region are then taken to be those of the region. The equations for the ellipse are defined in Equations 2. The orientation is given by angle θ. The eccentricity is the ratio of the smallest to the largest eigenvalue of B. The height and width of the region in pixels is the maximum difference between projections onto the principle axis of the ellipse. The principle axes of the ellipse are along the eigenvectors of B.

$$w = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$B = \begin{bmatrix} d & e \\ e & f \end{bmatrix} = \frac{1}{4\mu_{xx}\mu_{yy} - \mu_{xy}^2} \begin{bmatrix} \mu_{yy} & -\mu_{xy} \\ -\mu_{xy} & \mu_{xx} \end{bmatrix} \qquad (2)$$

$$E = \{ w \mid w^T B w = 1 \}$$

$$\theta = \frac{1}{2} \tan^{-1}\left(\frac{2e}{d-f} \right)$$

1) Possible right guide-bar candidates

The code markers are found by searching for the right guide-bar. Candidates for the guide-bar must have an eccentricity that is between 0.01 and 0.2, and greater than 100 pixels. Figure 6 shows the binary image with magenta stars marking the centroid of regions that meet these two criteria.

Figure 6: Regions with proper eccentricity

2) Estimate right corners

Given the centroid, orientation, and length of a possible guide-bar the algorithm then estimates the position of the top-right cornerstone and bottom right corner. Figure 7 shows the estimated positions with a red plus-sign.

Figure 7: Estimated position of right corners

3) Check existence and shape of right corners

The estimated right-corner positions are looked up in the labeled image to see if a distinct region lies at each of the corners. If so, then the eccentricity of the top corner must be greater than 0.4, and the eccentricity of the bottom right region must be between 0.01 and 0.2. The results of applying these criteria to the sample image are shown in Figure 8. In this sample case all but the correct marker region have been eliminated.

Figure 8: Regions with properly shaped right corner elements

4) Estimate left cornerstones

By this step the centroids of the top corner, right guide-bar and bottom guide-bar are all known. The distance between right corners and left corners is calculated using the length ratio and angle difference between the guide-bars. The positions of the left-corners relative to the right-corners are then estimated to be at the computed distance with the angle of the bottom guide-bar. These estimates are shown in Figure 9.

Figure 9: Estimated position of left cornerstones

5) Check existence of left cornerstones

The estimates of the left corner positions will not generally fall in the center of the cornerstone regions because all possible geometric distortions of the code plane are not compensated for in the estimate. Thus it is not sufficient to check for cornerstones at only the location of the estimate. This algorithm searches a square region around the estimate that grows until a cornerstone is found, or a maximum radius is reached. The maximum search distance is proportional to the length of the right guide-bar to account for marker scale.

If a region is found within the search area, its geometric properties are compared to those expected of a cornerstone. If all properties are correct, then the estimate is updated to the centroid of the region. At this point the code marker has been fully identified. The origin of the code marker is assumed to be the top-left cornerstone's centroid.

Figure 10: Centroid of cornerstones and guide-bars

C. *Datum Extraction*

The datum extraction step requires four points with their corresponding code plane and image plane coordinates. The code location step has yielded exactly this information. These points of correspondence are used to map known locations in the code plane to locations in the image plane. Once the image coordinates are known the bit value is processed from the grayscale image.

1) *Calculate coordinate transform*

For the coordinate transform it is assumed that the code lies in a plane and the image is in a plane and the code plane is projected onto the image plane. Under these conditions four points, and their corresponding coordinates in the image (x, y) and in the code (u, v), are needed to compute the correspondence at any other point. Equations 3 represent this coordinate transform. The four points that are used in the computation are listed in Table 1.

Table 1: Transform key points

(x_0, y_0)	Centroid of top-left cornerstone
(x_1, y_1)	Centroid of top-right cornerstone
(x_2, y_2)	Centroid of bottom-right guide-bar
(x_3, y_3)	Centroid of bottom-left cornerstone

$$
\begin{gathered}
x = \frac{au + bv + 10c}{gu + hv + 10}, \quad y = \frac{du + ev + 10f}{gu + hv + 10} \\
\Delta x_1 = x_1 - x_2, \quad \Delta x_2 = x_3 - x_2 \\
\Delta y_1 = y_1 - y_2, \quad \Delta y_2 = y_3 - y_2 \\
\Sigma x = 0.8x_0 - 0.8x_1 + x_2 - x_3 \\
\Sigma y = 0.8y_0 - 0.8y_1 + y_2 - y_3 \qquad (3) \\
g = \frac{\Sigma x \Delta y_2 - \Sigma y \Delta x_2}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2}, \quad h = \frac{\Sigma y \Delta x_1 - \Sigma x \Delta y_1}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} \\
a = x_1 - x_0 + gx_1, \quad d = x_1 - x_0 + gx_1, \\
b = x_3 - x_0 + hx_3, \quad e = x_3 - x_0 + hx_3, \\
c = x_0, \quad f = x_0,
\end{gathered}
$$

For each (u, v) that is within the data region of the code marker the closest corresponding (x, y) is computed. The data region is traversed column by column from left to right.

Figure 11: Image coordinates of code pixels shown in blue

2) *Convert gray-levels*

Figure 11 shows the location of code datum elements in image coordinates as blue dots. The pixel value of the grayscale image is measured at the center of each code element. The grayscale image is used since thresholding is a lossy operation, and may lead to bit errors.

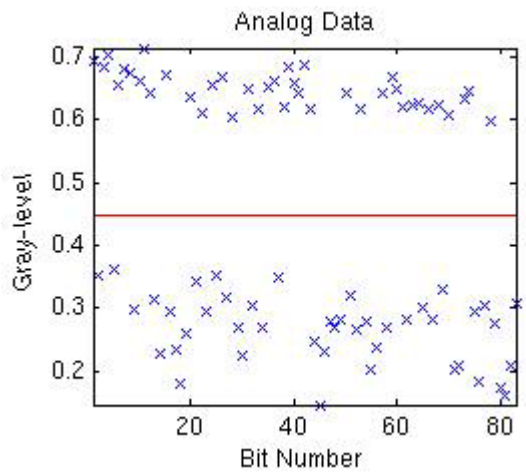

Figure 12: Gray level values of code pixels, mean value shown in read

All of the code bits are extracted from the image as gray levels prior to choosing a binary decision threshold. The threshold is chosen as the mean of the bits' gray values. This decision rule does not account for noise and signal statistics but is simple works well in practice. Figure 12 shows the gray values and threshold for the data in the sample image. After the thresholding operation all desired information is extracted from the visual code marker and the RGI algorithm is finished.

IV. Performance

An informal comparison the region-geometry identification (RGI) method to a normalized cross-correlation (NCC) method was conducted. Though RGI can only find patterns in a binary image and is limited in the type of shapes it can find, it is many times faster than NCC. NCC requires convolving a fairly large template through a large image matrix for each possible template shape. For applications where the template is fixed in size and orientation NCC performs quite well. However, the code markers in this application have unknown scale, orientation and perspective, which makes NCC nearly

infeasible, taking nearly ten minutes to match a code. Whereas, RGI excels at this type of application, taking approximately five seconds to match the same pattern.

This implementation of RGI does not account accurately for all of the possible geometric distortions of the code marker. Specifically, perspective squashing along code diagonals is not mathematically compensated. To overcome this effect the cornerstones have a position threshold. Because of this inaccuracy it is possible that a code marker viewed at an acute angle may not be detected.

Bit errors are possible with this implementation due to noise. After the transform between code and image coordinates is computed only a single pixel is used to estimate the code value. The noise can be mitigated by averaging a small number of pixels around the center. This is only necessary in very blurry or noisy images.

Though the algorithm can be improved in robustness, the existing implementation was able to correctly identify all the codes in the training images with zero bit-errors.

V. Conclusion

Region-geometry based code detection is fast and accurate method of reading data from the environment with a simple cell-phone camera. Because existing camera-phones have all of the hardware necessary to take advantage visual code markers this scheme can be deployed with little more than software modification.

References

[1] "Visual Code Marker Detection," 2006 EE368 project assignment, online at http://www.stanford.edu/class/ee368/project.html

[2] Michael Rohs, "Real-World Interaction with Camera-Phones," conference paper for UCS2004 online at http://www.vs.inf.ethz.ch/publ/papers/rohs2004-visualcodes.pdf