VISUAL CODE MARKER DETECTION ALGORITHM

An Binh Trong Ho
Stanford University, Department of Electrical Engineering

ABSTRACT

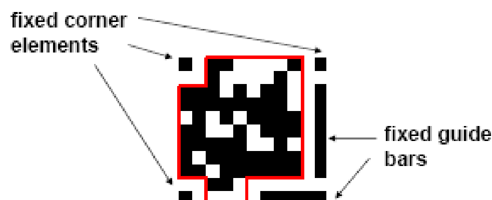
The convergence of digital photography and wireless internet access onto a mobile device has led to a growing number of applications for visual code markers. A need exists for a simple, accurate and robust algorithm to detect and decode these visual code markers taken with a cell-phone camera.

1. INTRODUCTION

The use of visual code markers to store information is becoming prevalent with the additions of cameras onto cell-phones, coupled with the ability of many phones to connect to the internet. Example applications include linking a document (newspaper, map etc.) to some related online content, automatic dialing of phone numbers stored in visual codes, or movement tracking. Ideally an algorithm that detects and decodes the code marker needs to be simple enough to run on a low power processor at a speed fast enough to be considered real time. It needs to be robust to different viewing angles, different lighting conditions, and low quality inputs that's often characteristic of images taken with a cell-phone camera.

This paper explains an algorithm that detects the presence of visual code markers in an image, finds the location of the marker within the image, and reads the data embedded in the marker. The algorithm accepts as input a color image with resolution of 480x640 pixels. Since the image is taken with a cell-phone camera, noise, distortions, and blur are to be expected. Test images contain 1 to 3 visual code markers. The algorithm returns the x-y coordinates of the upper left corner of each marker as well as the bits in each one.

Each visual code marker is an 11x11 array with either black or white elements. There are 3 fixed corner elements and two fixed guide bars, as illustrated in the sample below. The elements in the area enclosed by the red lines constitute 83 bits of data.



For a particular image, the performance of the algorithm is determined by a score computed using this formula:

(no. of correctly matched elements)
-83/2 x (no. of repeats + no. of false positives)

Repeat occurs when another detected code marker is closer to the true code marker location. False positive occurs when a detected code marker is too far from any true marker location. Negative scores will be counted as zero. Maximum run-time is 1 minute on a designated machine.

2. PRE-PROCESSING

The algorithm converts the rgb input image to grayscale using the Matlab function *rgb2gray()*. This reduces the number of pixels by a factor of 3 and thus makes the program run faster. Image enhancement techniques such as noise removal with wiener filters, contrast enhancement by stretching the dynamic range of the grayscale and Laplacian edge enhancement were tried. These tend to improve the visual appearance of the image but none of them actually improved the score from the *evaluate()* function. Some of them, like contrast enhancement, actually reduces the score. They are fairly costly operations so it's best to simply work with the grayscale version of the original input image.

To isolate the black from the white areas in the image, this algorithm calls the function *imextendedmin()*, which finds the regional minima. Depending on the threshold chosen as the parameter, the effect of this function is similar to a conversion to binary image, followed by an erosion of the black portions, followed by a not operation. An example is shown below. Note the 2 code markers in the image and the three corner elements in each marker.

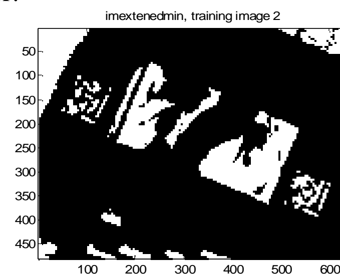
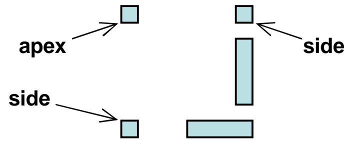


Figure: Isolating black and white areas

PDF created with pdfFactory trial version www.pdffactory.com

3. FINDING CODE MARKER LOCATIONS

The algorithm relies on the fact that there are 3 fixed corner elements in each visual code marker for detection purposes. The three corner elements form a right triangle. Let's call the element at the right angle the apex element and the other 2 elements the side elements.



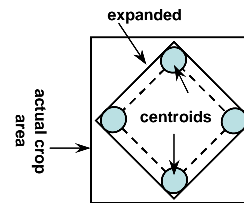
Interesting points in an image show up as small regions, which can be found using the built-in function *bwlabel()*. Their properties can by found by using the function *regionprops()*. The relevant properties are *area*, *centroid*, *major axis length*, and *minor axis length*. The *area* of an interesting point tends to be small so there is a maximum limit of 150 pixels. There is a minimum limit of 5 to remove noise specs from consideration.

Every point in the image is considered as a potential apex element and the algorithm searches for 2 side elements to pair up with a particular apex element. One apex element can have zero to multiple 2-element pairs and itself can be considered a side element for some other potential apex element. The search ultimately returns sets of 3-point groups that are potential candidates for corner elements in a visual code marker. A 3-point group is considered a candidate if certain requirements are met. First, the angle formed by the 2 side elements with the apex element must be around 90 degrees, with a possible variance of $\pm 30^{\circ}$. This variance is taking into account the shear to the square marker when the image is taken at some angle. Second, the Euclidean distance from the apex element to the side element must fall in the range from 40 to 200. Third, the Euclidean distance from the one side element to the apex cannot be more than twice as long (or less than ½ as long) as the distance from the other side element to the apex. Another word, an elongated rectangle cannot be a code marker. Finally, there are 2 possible permutations of the same 3-point group and thus the duplicate has to be removed. For example group [S1 Apex S2] is the same as group [S2 Apex S1].

The values of these parameters are found from experiments with the training images with the expectation that the testing images are similar in characteristics. The limits in the parameters do affect the robustness of the algorithm. However, with up to hundreds of interesting points to analyze, each point paired up with every other points, the number of 3-point groups to process must be reduced in consideration of the maximum run-time requirement.

4. EXTRACTING THE CODE MARKER

The *centroid* locations provide coordinates for the 3 points that are candidates for the 3 corner elements in a marker. If a candidate is confirmed as a code marker, the centroid location of the apex is returned as the location of the marker. The fourth point can be interpolated from simple geometry. The extraction coordinates are actually enlarged by 5 pixels in the x & y directions of each corner in order to capture the whole marker (see illustration). Note that the marker is extracted from the original full gray scale image, not the minimum extended image.



Because the image can be taken at an angle with the marker, a square marker can shear into a parallelogram. This is corrected by performing an affine transformation. Two functions in Matlab, *cp2tform()* and *imtransform()*, are used to implement this correction (see A1). The marker in the transformed image should be approximately square with horizontally and vertically aligned sides. It also resides in the middle of the transformed image and therefore can be extracted by an area of tighter fit. The orientation of the code marker can be found based on the location of the apex element (nw, ne, sw, se) relative to the side elements. Depending on the apex location, the extracted code marker can be re-oriented using some combination of horizontal flip, vertical flip, or matrix transpose. For example, if the apex is located on the north east corner, then the algorithm performs a horizontal flip on the extracted image followed by a matrix transpose operation. Section A2 contains images illustrating the process of image extraction and transformation.

5. DECODE THE DATA

By now, the algorithm possesses a grayscale image of the candidate for a visual code marker. It then attempts to decode the data and come up with an 11x11 matrix of logical values. The grayscale image is converted to binary. The threshold is taken from the graythresh() function, which uses Otsu's method [1] of threshold setting. The binary image is resized to dimensions of $(11k \times 11k)$, where k is an integer chosen such that $11k$ is the closest value to the maximum original dimension. From here, the resized image can be divided up evenly into 121 squares each of size (k x k). Each small square

PDF created with pdfFactory trial version www.pdffactory.com

represents an element in the code marker. The value for the element is determined by the majority pixels value in the square. If more than ½ the pixels are 1's than the value of the element is set to 1. Else the value of the element is set to 0.

At this point the algorithm returns an 11x11 matrix of binary values. Because of the large variance allowed for the parameters described in section 3, a lot of the candidates are false positives. Many are simply disparate points forming an angle from 60 to 120 degrees. This is where the fixed guide bars are used to help determine whether or not a candidate is a true code marker. The last 2 columns of the matrix must contain a predetermined pattern of bits. This is true for the last 2 rows are well (except for 6 elements). Therefore, a false positive will deviate from the pattern and thus detected and discarded. It is, however, still possible to deviate from the pattern even for a correct hit. This stems mostly from possible mis-interpolated 4th coordinates that led to parts of the south east corner missing and throws off estimate of the element values. It can also stem from miss alignment, where the guide bars are not vertically and horizontally aligned as they should. For these reasons, the algorithm allows for 6 deviations from the fixed pattern. Another word, if less than six elements are different from the pattern, than the candidate is considered a true code marker and the data is considered valid. Again, this parameter is chosen from experimentation with the training images.

6. RESULTS

Based on testing with the training images:

score_total = 1890 (max: 1909)
execution_time_total = 53.6170
num_correct_bits =
83 166 247 82 242 80 165 83 247 248 83 164
num_false_alarms =
0 0 0 0 0 0 0 0 0 0 0 0
num_repeats =
0 0 0 0 0 0 0 0 0 0 0 0
execution_time =
1.3620 0.8610 9.6340 0.6410 7.6010 1.1110 1.2120 0.7410 11.3270 14.6310 0.5910 3.9050

7.REFERENCES

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, No. 1, 1979, pp. 62-66.

[2] Department of Computer Science, ETH Zurich, "Visual Code Recognition for Camera Equipped Mobile Phones," http://www.vs.inf.ethz.ch/res/proj/visualcodes/.

8. APPENDIX

A1.

The function *cp2tform()* in Matlab infers a spatial transformation structure from pairs of control points. The 4 original centroid locations were used as input points and 4 interpolated points in a rectangle fitting the adjusted marker were used as base points.

With the transformation structure, the function *imtransform()* implements the transformation. Bicubic interpolation is chosen as the interpolation method.

A2.

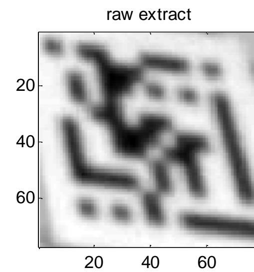


Figure 1: Extracted marker from grayscale image

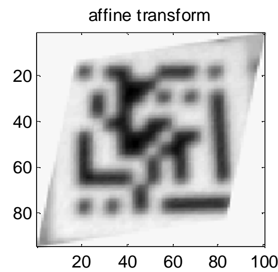


Figure 2: Extracted marker after affine transformation

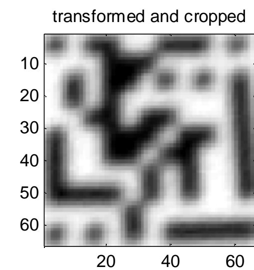


Figure 3: Cropped marker image

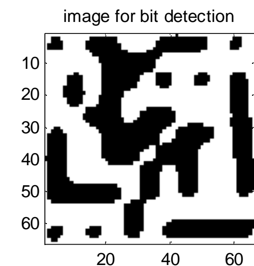


Figure 4: Image use to detect bits

PDF created with pdfFactory trial version www.pdffactory.com

Filename: ee368group30.doc
Directory: D:\Binh\School\Stanford\Spring 06 - EE368
Template: C:\Documents and Settings\Binh Ho\Application Data\Microsoft\Templates\Normal.dot
Title: Author Guidelines for 8
Subject:
Author: foobar
Keywords:
Comments:
Creation Date: 6/2/2006 3:48:00 AM
Change Number: 15
Last Saved On: 6/2/2006 6:52:00 PM
Last Saved By: Binh
Total Editing Time: 894 Minutes
Last Printed On: 6/2/2006 6:52:00 PM
As of Last Complete Printing
Number of Pages: 3
Number of Words: 1,716 (approx.)
Number of Characters: 9,783 (approx.)

PDF created with pdfFactory trial version www.pdffactory.com