Visual Code Marker Detection

Xu, Qiang

Stanford University, Department of Electrical Engineering, Stanford, CA, United States, qiangxu@stanford.edu
(National University of Singapore, Department of Electrical and Computer Engineering, Singapore, qxu@nus.edu.sg)

***Abstract* — a 2-dimensional visual code marker carries as much as 83 bits information. Thus it is a useful code marker system for storing and extracting information. This paper presents a digital image processing algorithm which automatically locates the marker on a cell-phone camera captured image. With the exact location of the code marker, detailed bits information of the visual code marker is returned. This algorithm takes into account of the rotation and tilt angle of the visual code marker. It is able to reconstruct a standard 83 bits map from the might-be distorted images, thus extracting the information stored in the visual code marker.**

I. Introduction

In this paper, we present a digital image processing algorithm to detect visual code marker. The goal of the detection algorithm is to look for visual code marker on a 480x640 image captured by a camera phone.

The visual code marker that we consider is a 2-dimensional array. The array consists of 11x11 elements and each element is either black or white. As shown in Figure 1, the feature of the code marker is fix with three of the cornerstones in black, one vertical guide bar (7 elements long) and one horizontal guide bar (5 elements long). The immediate neighbors of the corner elements and the guide bar elements are fixed to be white. This leaves us with 83 elements which can be either black or white.

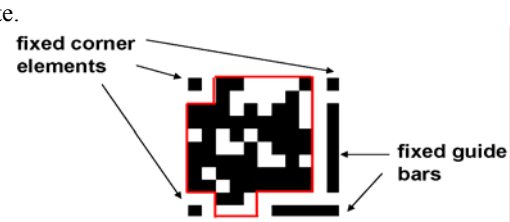

Figure 1, Sample Visual Code Marker

The idea is to first find the location of the code marker, then extract the embedded bits information. Due to be limitations on the luminance and color from the camera phone, these images are of low resolution and subject to sometimes severe color distortions. All these factors contribute to difficulties in locating the visual marker on the image, and also extracting its embedded information.

The proposed algorithm segment and lock the position of the code marker using its fixed features, such as the two guide bars and cornerstones. Together with a novel coordinate transform algorithm, the embedded bits information can be easily extracted.

II. Visual Code Marker features in Detail

From Figure 1, it is critical to investigate on the characteristics that the visual code markers possess, in order to aid our detection to a large extend.

A. Maximum contrast level

It is obviously seen that the code marker is designed in black and white color so as to achieve the maximum contrast level between the background and the fixed elements, as well as the embedded bits.

B. Two Orthogonal Guide bars

The two guide bars which are very noticeable in the images should be utilized to look for the position of the code marker in the image. In addition to that, to eliminate false positive findings, the relative angle between the two bars should be investigated. Under the condition of the tilting, the two bars might not be perpendicular to each other, but they will certainly lie in a range near ninety degrees.

C. Three cornerstones

In addition to the guide bars, there are three fixed elements on the three corners of the marker respectively. These cornerstones mark the boundary of the marker and thus can be used to ensure a visual code marker has been found, and eliminate false positives.

D. Structural information

Moreover, the position of the guide bars and cornerstones provides important structural information for us to derive a transform between image coordinates and standard marker plane.

III. Code Marker detection

Given the set of features described above for the code marker, the detection algorithm is depicted in the flow char as shown in figure 2. The 640x480 image is used to extract the Region of Interest (ROI) first. Then each ROI image is searched thoroughly. If time permits, the algorithm searches for the marker on the entire image for the markers.

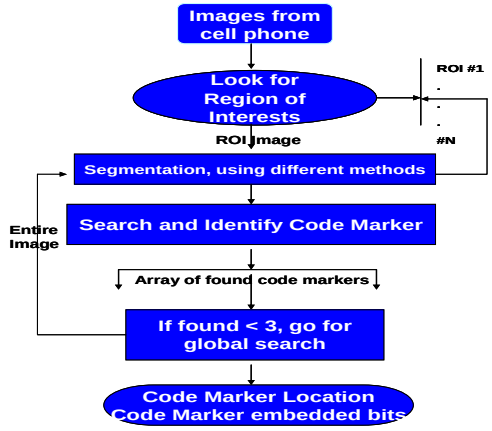

Figure 2, the overview flow chart for the whole process

1. *Extract Regions of Interest*

In order to find the visual code marker, the image has to be searched thoroughly for suspicious regions which might contain a marker. However, due to the noisy nature of camera phone captured images, it is a challenging task to segment the markers from various background conditions. The typical phone camera generates low to medium quality color images of 640 × 480 pixels. In the evaluation and testing, it is decided not to use color as a code feature because the color can change due to different lighting conditions etc.

In order to find the regions of interest (ROI), the abrupt change of luminance of the marker is used. It is observed that a code marker has much stronger edges than most background noisy objects. To further enhance the feature, the image is smoothed with a disk shaped kernel over the entire image. This is followed by canny edge detector with the intention to pick up the edge of the code markers only. A certain high threshold [-0.5 0.5] of the edge detector is set to eliminate other objects with softer edges.

From the experiment, best result was observed by converting the color image to NTSC format grayscale images so as to conserve the luminance of the image and forgo the chrominance. For most of the training images, only the edges of the code marker and a few noisy objects were obtained in the edge image. This enabled us to go on the search for interested regions.

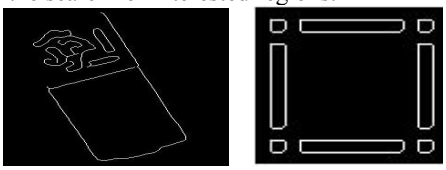

Figure 3, edge image for training_1.jpg and template

In figure 3, one can tell that the edge of the marker is pretty obvious and the image is clear of other noisy objects. With the edge image, template matching was performed with the edge image of a standard code marker in various orientations. The orientations were obtained by performing Hough Transform on the edge image and the top five most possible angles were picked for the matching. The template was carefully modified from the standard code marker, with edges of bars and corners on every side and tip of the image. By doing so, it only need to make sure the visual marker is in orientation angle off multiples of 90 degrees from its standard orientation.

The matched values of the five orientations are thus summed up and peaks above a certain threshold were kept as candidates for regions of interest. From the maximum peak, a region with a certain window size is captured and saved as a region of interest in an array. Regions too near to another found region is excluded to avoid repeated regions of interest and save processing speed.

From trials with the training images and proper parameter fine tuning, this method successfully returns regions of interest which covers all the markers in a complete and rapid fashion. In other words, the big image was then divided into small regions which might contain a real visual code marker. This set of small images is passed to the next step to look for the marker in details.

2. *Segmentation*

Segmentation is a critical task in this entire detection algorithm. The success or failure of the algorithm depends on whether the visual code marker can be properly and completely segmented.

With the output from last step, each region of interest image is then passed on to obtain a binary image. In order to make sure no visual marker is missed, an image is segmented under different segmentation schemes and parameters.

The first segmentation algorithm used an *adaptive thresholding method* [3]. Due to the inconsistent lighting and luminance of the camera, certain areas on the image can be darker than the others whereas same colors might have totally different luminance.

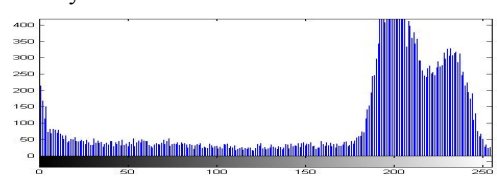

Figure 4, histogram for the markers from training images

The combined histogram for the twenty three gray-scaled markers was shown in figure 4. Despite the large portion of the pixels with high pixel value represent the white area. There isn't a clear boundary to separate the black pixels. Therefore, a fixed threshold value for all the images is not possible to be obtained.

We adopted the algorithm in the paper to traverse through the image, from top to bottom, from left to right on every odd rows and right to left on every even rows. This special traverse route makes sure the average window is close to the current pixel.

The window average $g_s(n)$ is set according to

$$g_s(n) = g_s(n-1)\cdot(1-\frac{1}{S}) + p_n$$

with P_n denoting the gray value of the current pixel and ***S*** *is the width of the average window*. The problem here is how to set the value of the initial $g_s(0)$. It is *initialized to* ***½*CS***, where ***C*** *is the maximum possible gray value*. The pixel value of the threshold pixel ***T(n)*** is obtained by comparing to the value of

$$\frac{g_s(n)}{s} \cdot \frac{100-t}{100}$$

If P_n is larger than this value, the pixel is set to one, otherwise to zero. The value of **t** plays a role here of setting the threshold adaptively.

With experimentations, it is noticed that for certain images with bigger visual code markers, the segmentations might fail to separate the bars from the inside code because the threshold for t was set too small. However, for certain images with small visual code markers, the algorithm sometimes failed to segment the cornerstones due to their small sizes. This requires us to use different set of S and T for images of different sized visual code markers. After trials, three set of S and T values were derived to run on every image. The algorithm attempts these three different sets of parameters until a marker is located or with the conclusion that no marker is present in this ROI image. The three sets of S and T are

Value/set	Set 1	Set 2	Set 3
S	1/16*Width	1/40*Width	1/32*Width
T	32	8	15

The second segmentation algorithm that was implemented was *Otsu's method* [6]. Otsu's method basically selects and sets an appropriate global threshold value for the entire image by looking at the histogram of the grayscale image. The threshold is selected at run time, thus the threshold is adaptive to each image. In practical observations, Ostu's method returns a much cleaner binary image for images with small-size visual code markers.

From the twelve training images, these four segmentation schemes successfully segment all visual code markers completely, with guide bars and cornerstones. However, due to the sequence of performing these four schemes, final score might vary because the information bits inside the code marker could not be perfectly segmented and some errors were observed.

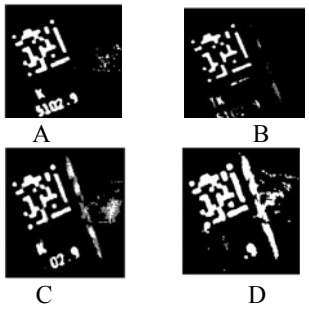

Figure 5, binary images using different segmentation scheme, A: Otsu; B: Adaptive-1; C: Adaptive-2;D: Adatpive-3

From figure 5, it can be observed in this case, the Otsu's method and adaptive thresholding with parameters set 1 gives the best result. However, Otsu's method amplified the noise below the marker. Therefore, each segmentation scheme has it own advantages and drawbacks.

3. Detection and Identification

After each ROI image is properly segmented, it is passed to the detection and identification module to look for a code marker and return the detailed location in image coordinates. The intrinsic features and compulsory components of the visual code marker are explored to locate and ensure a visual code marker has been found.

Step1: Look for long Guide Bars

There are two guide bars in each code marker. The longer one in size has one dot and the shorter bar on its two ends respectively. Based on this important feature, the binary image is labeled using connected-component region labeling algorithm. From the group with biggest size to the smallest, each labeled object is investigated to look for bars first. The eccentricity and major-axis length of the object is investigated to test on a set of criteria to determine whether the object is a bar. The properties of each region are obtained through a Matlab built-in function called "regionprops". The basic principle behind is the object's eigenvectors [7] need to be found first. Then from the second moment of this object, one can obtain its eccentricity, a line should have an eccentricity near to one, whereas a dot has eccentricity near to zero. In this case, an object with eccentricity larger than 0.9 and major axis length greater than 20 pixels is considered as a bar. These criteria and parameters are obtained from the training images.

Step2: Look for short Guide Bar and cornerstone

From the long guide bar, local search from the two ends are performed in aim to find the short guide bar and cornerstone. The search is started at certain length away from the centroid of the long bar. The distance is chosen based on the length of the long bar.

Similarly, the eccentricity and major-axis length is used to judge whether the object found near the ends is a short bar or a cornerstone. For cornerstone, the eccentricity has to be smaller than a certain threshold and so is the length. For the bar, additionally, the relative angle between the longer bar and the shorter one is used to test whether the two bars are perpendicular to each other. Due to the tilting effect, this criterion is set to be a certain range around ninety degrees.

In the case of any or both of the short guide bar and cornerstone wasn't found, the long bar was discarded and the loop goes on to another labeled region.

Step3: Look for the other two cornerstones

From the two bars and one cornerstone, it is easy to start another local search to look for the two cornerstones left, based on the relative position and orientations of the cornerstones and bars. Similar criteria are considered to

test the two cornerstones and the locations of the cornerstones are estimated based on the known position of the two bars and one cornerstone.

Figure 6, marker locations found

Figure 6 shows the marker with marks on the three cornerstones and the shorter bar.

Step4: Store information and look for second one

After all the bars and cornerstones are located, the locations are stored in an array. All the labeled regions of the bars and cornerstones are thus marked as zeros so that they would not be considered for further searches. After this, the loop goes on to look for another code marker until all three markers are found in one ROI image or other regions larger than a certain area.

At the end, the module returns an array of the stored locations for each of these markers found.

4. *Global search for visual code markers*

After all the regions of interest images are examined by the detection and identification module, repeated markers are discarded by checking the relative Euclidean distance between the left-upper corner and the shorter bar locations.

However, if the total number of markers found is less than three, which is the maximum number of markers can exist in an image. A global search for markers is performed to make sure that no markers were missed in the step generating regions of interest images. Because the regions of interest search method is extremely fast, a normal image can be processed within 10 seconds using Matlab, and therefore we can afford to perform another global search on the image to ensure that nothing is missed at the ROI generation step. The segmentation scheme used for the global search is the adaptive thresholding algorithm and Otsu's method. Time was constantly monitored in the program, if the ROI/detection process takes longer than 30 seconds; the global search method is never started. However, if time allows, a second global search using Otsu's segmentation method will be performed on the entire image again, if total markers found is less than three.

5. *Mapping between image coordinate with code marker coordinates*

To map the image coordinate with code marker coordinates, the transform method based on [1] is adopted. With the set of found markers coordinates, there always exists a one-to-one mapping between the image code marker coordinate system and the standard code marker coordinate system.

From figure 7, it shows the coordinate system in the standard image coordinate system. With the pixel positions of the shorter bar and three cornerstones, we are going to derive the transform between them.

Code element	Image coordinate	Code coordinate
Upper left cornerstone	($x0$, $y0$)	(0, 0)
upper right cornerstone	($x1$, $y1$)	(10, 0)
second guide bar	($x2$, $y2$)	(8, 10)
lower left cornerstone	($x3$, $y3$)	(0, 10)

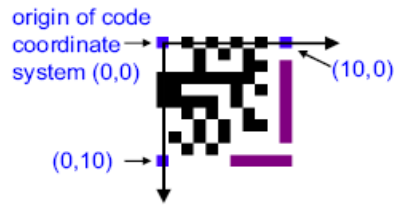

Figure 7, code marker coordinate system[1]

Based on the ideas for texture mapping [2] the mapping derived for the coordinate system is thus, with u and v denotes the marker coordinates:

$$x = \frac{au + bv + 10c}{gu + hv + 10}, y = \frac{du + ev + 10f}{gu + hv + 10} \text{ [2]}$$

$$\Delta x1 = x1 - x2,\ \Delta y1 = y1 - y2,$$
$$\Delta x2 = x3 - x2,\ \Delta y2 = y3 - y2$$
$$\Sigma x = 0.8x0 - 0.8x1 + x2 - x3,$$
$$\Sigma y = 0.8y0 - 0.8y1 + y2 - y3$$

$$g = \frac{\Sigma x \Delta y2 - \Sigma y \Delta x2}{\Sigma x \Delta y2 - \Sigma y \Delta x2}$$

$$h = \frac{\Sigma y \Delta x1 - \Sigma x \Delta y1}{\Delta x1 \Delta y2 - \Delta y1 \Delta x}$$

$$a = x1 - x0 + gx1;\ d = y1 - y0 + gy1$$
$$b = x3 - x0 + hx3;\ e = y3 - y0 + hy3$$
$$c = x0 \quad ; \quad f = y0$$

Figure 8, code marker transform to code coordinates

From this relationship, we have located the marker and the transform is established, the gray-scaled marker image is mapped into a square marker image as shown in the middle of *Figure 8*. This square marker image is then binarized and down-sampled into 11x11 image to extract the embedded bits information. By performing segmentation on the mapped marker image again

[1] Image adopted from Michael Rohs paper "Real-World interactions with camera phones"

provides us with higher accuracy in segmenting the detailed bits information, rather than work on the binary image of the ROI or the entire image.

After this final step, each visual code marker has been located and the embedded information is thus returned.

IV. Discussions on different algorithms

The first challenge encountered was the selection of the correct features to crop regions of interest. Template matching was attempted on the binary image. The regions with the peak values are extracted as regions of interest. However, due to noisy background, the performance was very poor. The algorithm returns too many regions of interest; and, it often misses the true region which contains the visual code marker. This is because in the background, many objects are of similar luminance as the code marker. Often, code markers are poorly segmented with missing bars or cornerstones. Therefore, binary information is not a reliable method to distinguish visual code markers with the interference from the background noise.

After observing the edge image, it is surprising that the edge of the visual code marker is well preserved after close observation. One idea is to use the edge to locate the rough position of the code markers. This reduces the false positives dramatically and the hit rate was almost 100% in the sense that each ROI image contains one different code marker.

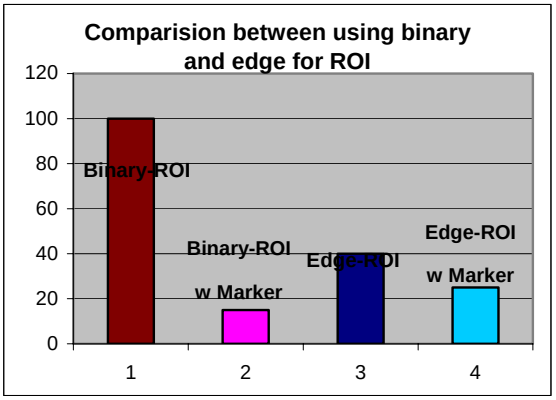

Figure 9, performance between binary image and edge image

From figure 9, it is shown that while using edge image for template matching, the rate of region of interest that includes complete code marker is much higher than using binary image. Thus, the high contrast of the edge for the code marker is an important feature to start with to locate the rough position of the markers.

In the second step, each segmentation scheme has its own advantage in looking for different types of markers. The adaptive thresholding method is very time-consuming but it adaptively set the threshold for each and every one pixel based on its neighboring pixels. However its sensitivity makes it very sensitive to the scale of the marker. However, in the Otsu's method, a global threshold was set for all the pixels based on the histogram and the grayscale value distribution. It is not suitable to globally segment the marker out of the entire image but might be a fast and effective solution for ROI images, due to the small size and less noisy characteristics. The drawback of the Otsu's method is that it often failed to segment the image in a detailed manner as much as the adaptive thresholding method does.

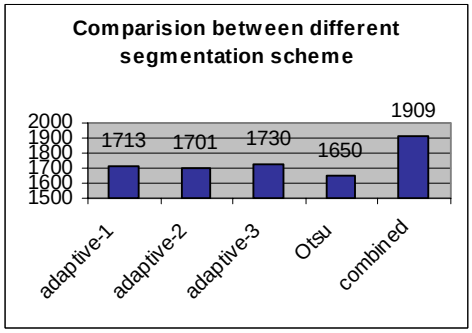

Figure 10, comparison between different segmentation methods

Figure 10 shows that the adaptive thresholding method generally performs better than the Otsu method. Therefore, in the algorithm, the adaptive thresholding method was performed first; Otsu method is performed last if only no marker was segmented by the previous methods. This combined approach of different segmentation methods improved the score dramatically and achieved 1909 points out of the total of 1909 points with the 12 training image, 23 markers.

V. Figures

	Marker found	Time/sec	Evaluation Score
Image 1	1	20	83/83
Image 2	2	36	166/166
Image 3	3	7	249/249
Image 4	1	50.3	83/83
Image 5	3	7.7	249/249
Image 6	1	25	83/83
Image 7	2	23.1	166/166
Image 8	1	21.6	83/83
Image 9	3	8.92	249/249
Image 10	3	7.68	249/249
Image 11	1	40	83/83
Image 12	2	25.9	166/166
Total	**23**	**287**	**1909/1909**

Table 1: Results for 12 training images

The above table shows the performance and score for each training image. The image which has three visual code markers runs much faster than the others because the algorithm runs the entire image through the search and identification module if less than 3 markers were found based on the ROI images. However, the markers for every image are detected within the allowed time limit with very high accuracy. By segmenting the transformed grayscale marker image again allows us to

choose the threshold on a much smaller region thus improves the performance.

The algorithm has also been tested on scaled, rotated and intensity varied training images. Results show that the algorithm can still keep its high accuracy with a scale factor between 80%-120%, rotations angle of N*x*90 (N=Integer) degrees and intensity level between 90%-110% on every training image.

In addition, 40 training images were generated by me using cell phones and webcams. Some of the images are much harder than the training images provided, with larger tilting angle and shadow noise on the marker etc. However, high accuracy and consistent performance of the algorithm was still observed. Figure below shows sample training images produced by myself.

Figure 11, sample images

VI. Conclusion

In this paper, I have presented the detection and identification algorithm for the 2-dimensional visual code markers. This algorithm adaptively sets threshold for segmentation and filters the noise. The search algorithm intelligently searches for correctly shaped objects. The novel transform on the mapping technique between code coordinates and image coordinates enabled us to ignore the effect of rotation and tilting. This saves a lot of work in terms of adjusting the code marker to the right orientation and the correct perspective angle.

In the whole algorithm, the most critical part is the segmentation. Segmentation has long been the most discussed topic and it is still very difficult to perform intelligent segmentation automatically and achieve the best result. In this presented detection algorithm, two best tested segmentation methods were combined. However, with advantages and drawbacks for these segmentation schemes, the result might fluctuate a little bit for different image sets.

The algorithm is fully tested both on a DELL Latitude D510 laptop and SCIEN LAB workstations with total execution time of 287 seconds and best score of 1909, over the 12 provided training images.

References

[1] Rohs, M.: Real-World Interaction with Camera-Phones. In: 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Tokyo, Japan (2004)

[2] Paul S. Heckbert: Fundamentals of Texture Mapping and Image Warping. Master's Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.

[3] Pierre D.Wellner: Adaptive Thresholding for the DigitalDesk. Technical Report EPC-93-110, Rank Xerox Research Centre, Cambridge, UK, 1993.

[4] Takeshi et al: A Coded Visual Marker for Video Tracking System Based on Structured Image Analysis. Proceedings of the second IEEE and ACM international Symposium on Mixed and Augmented Reality 2003.

[5] Xiang Zhang: Visual Marker Detection and Decoding in AR Systems: A Comparative Study. Proceedings of the International Symposium on Mixed and Augmented Reality 2002.

[6] Ku Chin Lin: Fast image thresholding by finding the zero(s) of the first derivative of between-class variance. Machine Vision and Applications, Archive volume 13.

[7] Sotirios Gyftakis et al: Data structures, computational geometry: Image-based change detection of areal objects using differential snakes. Proceedings of the 13th annual ACM international workshop on Geographic information systems GIS '05