Visual Code Marker Detection

Meng Yu
mengyu@stanford.edu

1 Introduction

This paper presents an image processing system that detects 2 dimensional square visual codes in images taken by mobile phone cameras. The algorithm applies edge detection, shape detection, grayscale thresholding, linear transformation, and other techniques in effort of obtaining high performance. There are 3 major parts in the system: producing binary image production, locating visual codes, and evaluating code values.

2. Binary Image Production

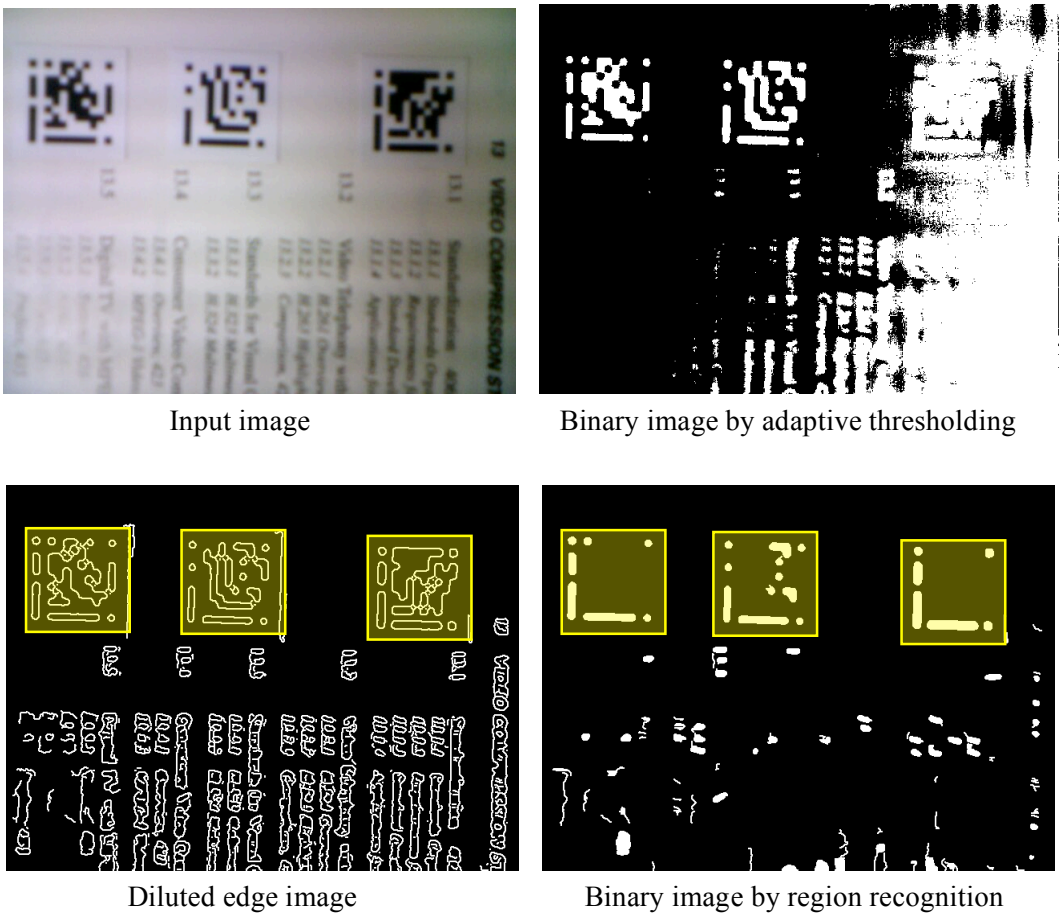
The input image is a 24bit RGB image. Since the features of visual codes are monotone patterns, it is efficient to remove the colors and highlights the possible candidates of visual codes in a binary image. Because the rest of the algorithm heavily relies on the binary image having all visual codes, it is important that the binary image contains representation of all the present visual codes in the input image.

My first attempt was to use gray scaling and thresholding. In order to remove the varying illumination in the image, I converted the image from RGB to YCbCr color space, and used the luminance values as the grayscale image. I tried both global threshold and adaptive threshold computation described in [1]. The best global thresholding for the training set is 100. Both algorithms work well most of the images, but fail occasionally. The intensity of the pixels varies over the image. The global thresholding only locates the global minimum in the intensity distribution. The algorithm fails, when there're a number of local minimums. The adaptive thresholding attempts to fix this problem by thresholding images differently across the pixels. The task of determining the thresholding regions is challenging. Wellner scans the image in a zig-zag pattern and defines the neighboring pixels as *n* previous scanned pixels, where *n* is user defined. The algorithm takes into account the affects of neighboring pixels, but it's very directional. If the image were scanned vertically, the resulted binary image would be different. Moreover, it's difficult to decide a global value of region size *n*, especially in this system, where the relative sizes of visual codes vary. Finally, the execution time of adaptive thresholding is expensive. Another reason that it is difficult to reach high performance using thresholding is the cell phone images have low qualities, that the noises can affectively confuse the computation.

After numerous trials of combination of different parameter values, I gave up on the gray thresholding. I noticed that because the shapes of the visual code elements are well defined as filled squares or rectangles, the edge detection can accurately pick up the edges of connected elements. This is especially true for the fixed guide bars and corner elements, where the surrounding regions are always white. The binary image can be constructed by detecting the enclosed areas in the edge image. The system uses Canny

edge detection. The edges are always 1 pixel wide. In order to recognize the connected edges, it dilutes the edges by a 2x2 pixel square, and labels the 4-connected components. For each connected edge, it checks the enclosed region by scanning each row that contains edge pixels and finding the enter and exit points of the region in each row. Allowing only one enter and exit points ensures the convex shapes of the regions. Figure 1 shows the results from a training image. The image on the bottom left is diluted image of the result of canny detection, and the image on the bottom right is the binary image created from detecting the enclosed areas and shapes. The visual code is highlighted by yellow squares. The two guide bars and 3 corner elements are preserved in well-defined shapes. This is an example that the thresholding technique struggles, because the most right visual code is darker than the left two. The result is shown in the upper right image of figure 1. The detection algorithm in this step is conservative, meaning it possesses high probability of including guide markers and but may include false positives.

Because the input image has low quality, I sharpened the image by filtering the image with a high pass filter at the beginning so that the edges are more dominant.



Input image | Binary image by adaptive thresholding

Diluted edge image | Binary image by region recognition

Figure 1. Examples of using different techniques of producing binary images that contains visual marker features.

3. Guide bar and corner code recognition

The guide bars and corner elements are the keys to locate visual codes. These features are the invariants for all visual codes. The pseudo code of guide code recogition is presented in figure 4. The corner elements are referred as *Corner1*, *Corner2*, and *Corner4* from origin to left and bottom most corner. Figure 2 illustrates the name convention. The element positions in image space refer to the centers of the connected regions that represent the element.

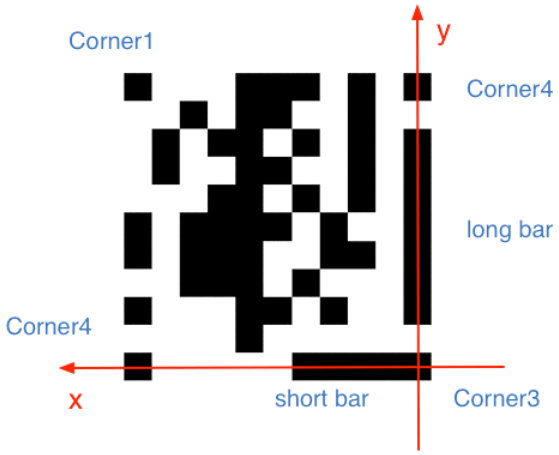


Figure 2. The representation of visual code and notations.

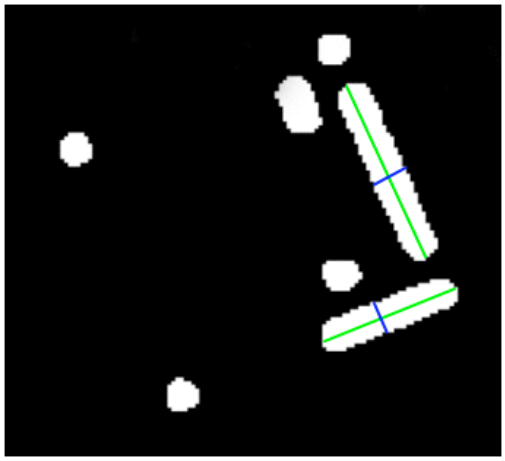


Figure 3. ellipse axes of connected regions.

To identify the guide bars, I used calculation of region shapes described in [2]. Because the guide bars are rectangles whose ratios of width to height are high, the system treats each connected region as an ellipse and computes the ratio of major and minor axis lengths. The bigger the ratio is, the longer the region is, and thus, it is more likely to be the long guide bar. Figure 3 shows the axes of the connected region, where green lines are the major axes and blue lines are the minor axes.

After selecting candidates of the long guide bar by passing the axis ratio threshold, the system locates the candidates of the short guide bar for each long bar candidates. This is done by computing the areas and comparing it with 5/7 of the area of long bar candidates. Though the orientations of the two bars may show some hints of the relationship between the candidate pair, it's not reliable, because the visual code may be titled and the perspective projection causes the two bars to be no longer perpendicular in the image space. To locate the 3 corner elements, I calculated the code-image transformation matrix that transforms code coordinates to image coordinates. All spatial transformations (i.e. rotation, scaling, and projection) are linear, except translation. One solution is to use homogenous coordinate system and rise to 3D space, the other is to apply translation independent of the matrix. I defined the code origin as the corner formed by the two bars (figure 2), and subtracted the origin values when calculating the matrix parameters. The centers of the two bars are used to calculate the 4 transformation matrix parameters. Once the system has the matrix, the 3 corner locations can be found by applying the transformation. For instance, the expected center position of Corner1 is

[10 10]**tform* + center(Corner3), where tform is the transformation matrix. During the process, candidates that do not pass the thresholds are rejected. Figure 5 shows result of located guide codes, marked by the triangles, on the binary image. Each color represents one set of markers.

Pseudo code Locating guide markers

```
Input: Binary image
Label the binary image.
Compute axis ratios for connect regions.
Select candidates of guide bars: axis ratio > ratio threshold.
Compute region areas of candidates.

For each candidates
   Short guide bar candidates: area ~= (5/7) of current candidate area.

Compute orientation of candidates.
For each pair of short and long bars
   Estimate the size of a single visual code element.
   Compute the angles between two bars.
   Compute the expected distance between centers of two bars, and make
      sure the actual distance is close to the expected value.
   Compute a linear transformation that maps the bars from code space to
      image space.
   Compute the expected positions of corner2 and corner4.
   Compute candidates of corner elements: size ~= single element size AND
      axis ratio ~= 1.
   Select corner2 and corner4 candidates who are corner candidates and
      close the expected distance.
   Locate origin corner element in similar way as calculating other two
      corners.

* ~= approximately equal
```

Figure 4. The pseudo code of locating guide markers.

Input image binary image with marked guide codes

Figure 5. Binary image with marked guide codes.

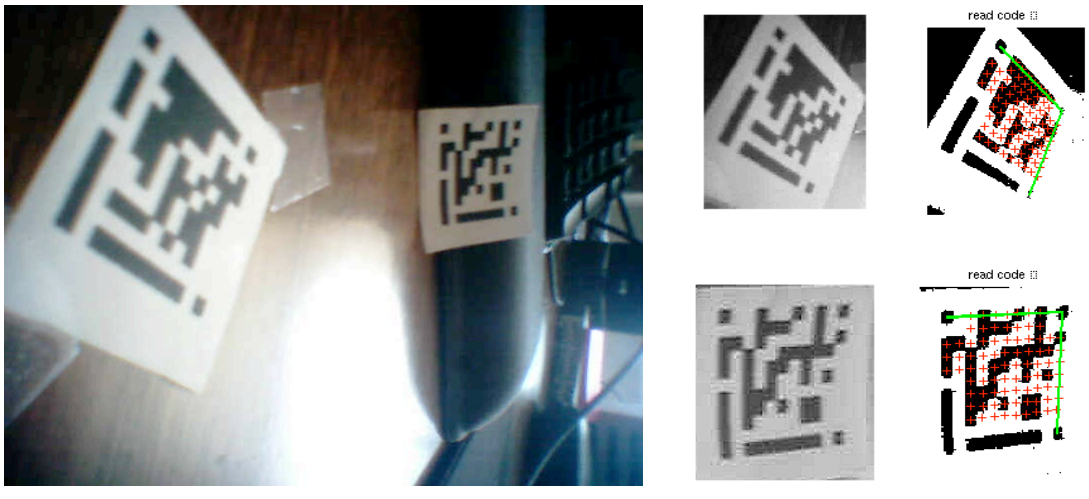
3. Visual code evaluation

The pseudo code of the code evaluation is shown in figure 6. For each detected code image, the system creates a sub image that only contains the visual code portion of the grayscale image, and applies gray thresholding to the sub image. Because the sub images are small, global thresholding is accurate enough to separate the code from the background. The produced binary image is used to read the code values. The transformation matrix is recomputed with the actual positions of detected corner elements from previous step. Comparing different weighting and averaging functions for computing the element values, I found that simply using the center of the element as the value of the corresponding code is the most efficient and accurate. Figure 7 shows the cropped images and their corresponding codes mapping. Notice the upper image is highly perceptively distorted; the system can still recognize and evaluate the visual code correctly, because the code-image transformation includes the projective matrix.

Pseudo code of visual code evaluation

Input: A set of 4 detected corners.
For each located visual code
    Crop the grayscale image, and create a sub image with only the visual code portion of the original image.
    Apply global thresholding to separate visual code from the background.
    Recomputed the code-image transformation matrix.
    Compute the centers of 83 elements and read the values from the sub binary image.

Figure 6. The pseudo code of evaluating detected visual codes.

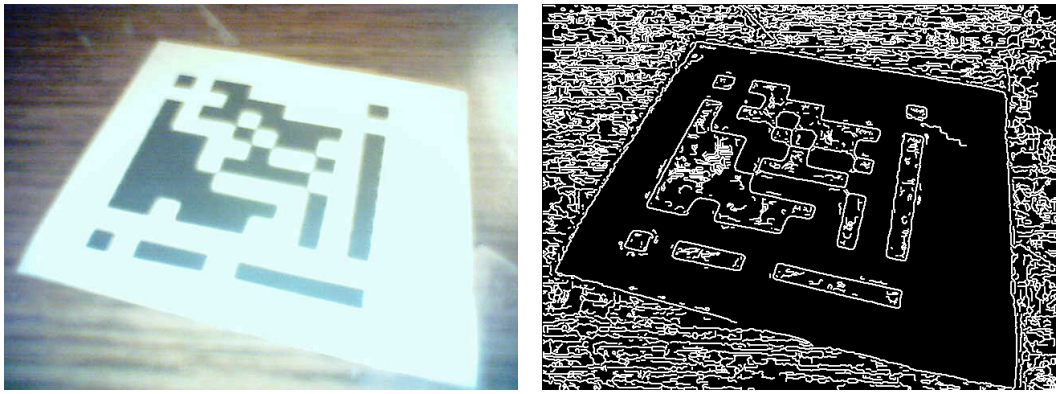


a b c

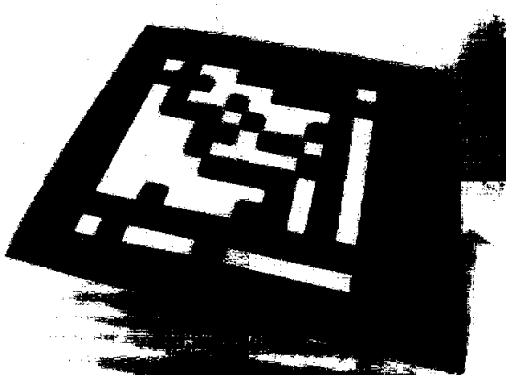
Figure 7. Result of code evaluation. a) input image. b) cropped grayscale visual codes c) binary images with marked code elements (green lines are two axes and red crosses are the center of elements).

4. Backup system

It is possible that edge detection strategy fails to detect visual codes. For instance, figure 8 shows a close-up shot of one visual code. Because of the low quality the image is, the edge image shown in figure 8 contains a big amount of false positives, such as the short false edges in the guide bars. The enclosed region detection would be distracted by the false edges and failed to detected the bar region. On the other hand, the gray thresholding works well (figure 8c). If edge detection didn't find any visual code, the system makes second attempt using gray adaptive thresholding.



a b



c

Figure 8. a) Input image. b) Diluted edge image.
c) Binary image by adaptive thresholding

5. Results

The system successfully detected all visual codes in the training set correctly with no false alarms and repeats. The details are in table 1. I also created a set of extra test images that reveal extreme cases. Some of them are shown in this report. The system performs well, too.

	1	2	3	4	5	6	7	8	9	10	11	12	total
correct bits	83	166	249	83	249	83	166	83	249	249	83	166	1909
false alarms	0	0	0	0	0	0	0	0	0	0	0	0	0
repeats	0	0	0	0	0	0	0	0	0	0	0	0	0
exec time(s)	7	9.57	8.16	11.42	7.97	10.21	10.40	8.76	9.2	8.58	8.97	10.79	111.04

Table 1. Performance result of training set.

5. Reference

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

[2] Michael Rohs. *Real-World Interaction with Camera-Phones*. 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Tokyo, Japan, November 2004.