EE368 Project: Visual Code Marker Detection

Doe Hyun Yoon, Sang Hui Ahn

Abstract — A process for marker detection which involves adaptive thresholding, region-modeling and mapping has a challenge of recognizing marker in noisy space. We propose accurate and computationally efficient algorithm in detecting any number of marker within moderate tilt in camera image.

I. Introduction

Visual code marker, taken with mobile phone camera, is prone to have noise in pictures with its limited resolution. A marker in a picture can also be arbitrarily rotated and tilted. Hence the challenge is how to recognize the marker with accuracy despite poor image quality, varying illumination, markers' size and orientation. Speed in obtaining data from a marker is significant as well because a marker should quickly load the information a user would like to obtain.

In this paper we discuss a systematic sequence of procedures in detecting marker, followed by other attempts in solving the problem. Fundamental flow of algorithm was mainly adopted from [1].

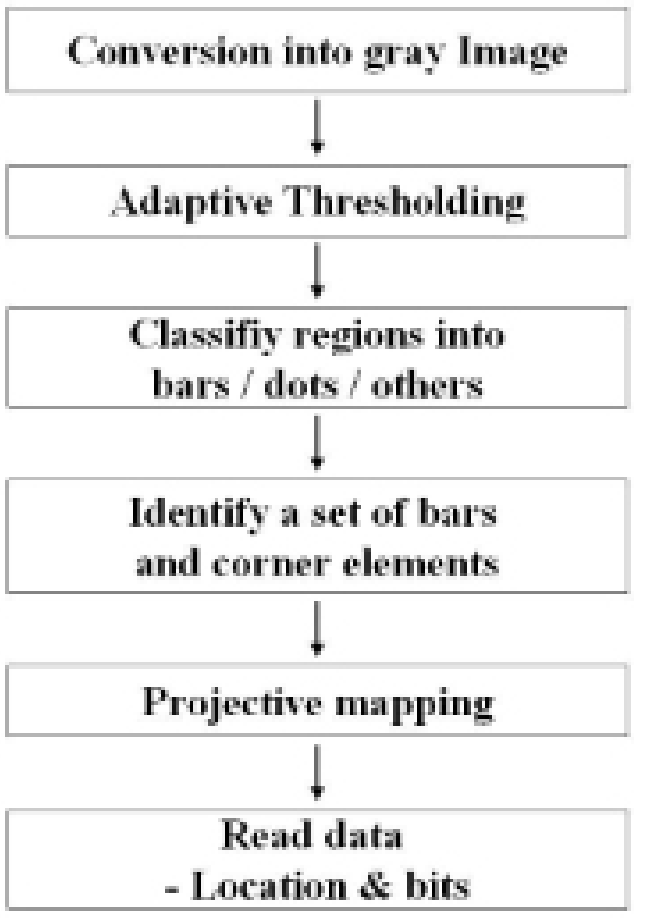

Fig. 1 Marker detection procedure flow chart

II. Approach

The flow chart used for detection algorithm is shown in (Fig.1).

Here is the outline of m files with their functions in detail.

- *detect_code.m* : It is the main function that implements the overall process and returns data and points of the upper left corner element.
- *adaptive_thresholding.m* : It applies adaptive thresholding technique and returns binary image separated by threshold value.
- *fit_ellipse.m* : It fits the region into ellipse and provides rotation angle, length of axis, center points. .
- *draw_ellipse.m* : It is used for debugging mode to plot ellipse to be fit inside guide bars from ellipse information.
- *draw_feature.m* : It draws and plots circles based on the information of long guide bar with certain search range for three corner elements.
- *find_corner_stone.m* : It finds a corner element from a center point within certain range and returns the label of region that contains the binary pixel.
- *mapping_image.m* : It performs projective mapping from code to image that returns 11 by 11 positions inside the marker.
- *read_data.m* : It reads data from positions obtained through projective mapping.
- *bwlabel_mod.m* : It upgrades speed as to examining region of the same label. This is a modified version of m file using bwlabel2 function inside bwlabel.m function.
- *ycbcr_marker.m* : It converts RGB image to YCbCr image and classify the region into two according to the similarity of its CbCr components with the marker.
- *obtaining_ycbcr.m* : It is a separate function from the procedure that calculates the mean and standard deviation of color components CbCr.

III. Algorithm

A. Gray scaling and adaptive thresholding

We first convert an RGB image to a grayscale image by using rgb2gray Matlab function. Using this function turns out to be less noisy than applying the coefficients in ITU-standardized

Fig. 2 (From training_5 image) Example of binary image created after adaptive thresholding.

formula. Then adaptive thresholding method described in [1] adapted from [2] is employed to separate foreground from background with nonuniform illumination. Constant threshold value does not work in detecting markers because camera images have varying brightness over the image.

Calculating a moving average of the last $s = \frac{1}{8} width = \frac{1}{8} 480 = 60$ pixels, the output binary image T(n) with (t = 15) percent darkness is :

$$T(n) = 1 \quad \text{if} \quad p_n < h(n) \cdot \left[\frac{100 - t}{100 - s}\right]$$

$$T(n) = 0 \quad \text{otherwise}$$

where p_n is the current gray value, $h(n)$ is the average of $g_s(n)$ and $g_s(n - width)$, and $g_s(n)$ is an approximate average of the last s pixels at current point as in [1].

Thus the final binary image denotes our target of interest as 1 and 0 otherwise (Fig. 2).

B. *Classifying regions into bars / dots / others*

Guide bars (GB hereafter) and corner elements (CE hereafter) are indicators of a marker, thus the aim is to obtain position of two bars and three dots before reading data from a marker.

1) Finding bars: Since the size of the marker varies, we made use of the fact that two fixed GBs do not change in the ratio of its width and height. After labeling the regions of which the number of pixels satisfies above the threshold of 20 pixels, we modeled those regions into ellipses. Regions that modeled into ellipses are then good candidates for GBs. Fitting into ellipse provides us with some merits in that we can measure the rotation of the ellipse, and the ratio of width to height is retained the same regardless of its angle of rotation. We adopted the code for fitting into ellipse from [5]. The length of long axis, short axis, and its center provides critical information in later processes where calculation about relation between the bars and dots is involved.

More specifically, within the regions identified with the same label, we set the region as a bar of which the ratio of long axis to short axis lies in the range of 2.7 to 10. True ratio for long and short guide bar is 7/1 and 5/1 respectively, but we allowed margins considering the distortion from noise in an image.

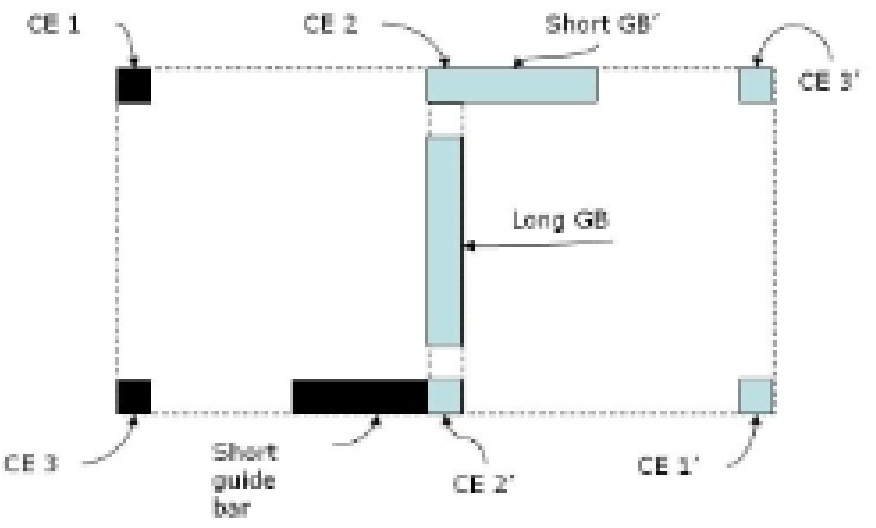

Fig. 3 Two markers are conceivable in each direction.

2) Finding dots: With the region that is not bar, the potential CEs are every dot that has the width to height ratio of approximately 1/1. Since dots can be seen as a circle, without the course of modeling into ellipse, the length of the region is measured in horizontal and vertical direction and if two lengths are within some tolerance it is defined as a dot.

C. *Identifying a set of bars and corner elements*

One marker is recognized with two GBs aligned perpendicular to each other and three CEs: one CE that stretch from two GBs respectively and the other one that lies on the vertex determined by the parallelogram defined by three points- two CEs and one point in the short GB. (Fig. 3)

First step is to start by finding a set of two bars perpendicular to each other. Among the bars defined in the above (B), assuming that this is the long GB, we search for short GB in two directions. For a fixed long GB, two marker positions are conceivable.(Fig 3) If a bar which lies within 90 degrees (with some tolerance) is found, next step is to inspect if the potential bar's short axis and long axis are within the range compared to long GB: in terms of short axis, a short GB's should be in 60% to 140% of the long GB's short axis (true ratio is 1 because they both occupy 1 pixel width); in terms of long axis, a short GB's width to height ratio should be within 3/7 to 6.9/7 ratio with long GB. (cf. true ratio is 5/7 without margin)

With the information of long axis of two GBs , angle between them and direction, we set the range so that each of three CEs lie inside. We treated points in image plane as vectors from the origin. CE2 can be found by extending a peak point w.r.t. center point of the long axis to 5/3.5 with the direction specified: i.e. CE 2 is p1 in (Fig. 4). To detect CE3, we extended another peak point as we did in p1, which is p2 in (Fig. 4). Then CE3 is located by rotating p1 w.r.t p2 about the angle between bars. CE1 is formed by rotated from p2 w.r.t p1 about the angle between bars deducted from 180 degrees. We set 5% margin to the angle between bars to the outward direction since with

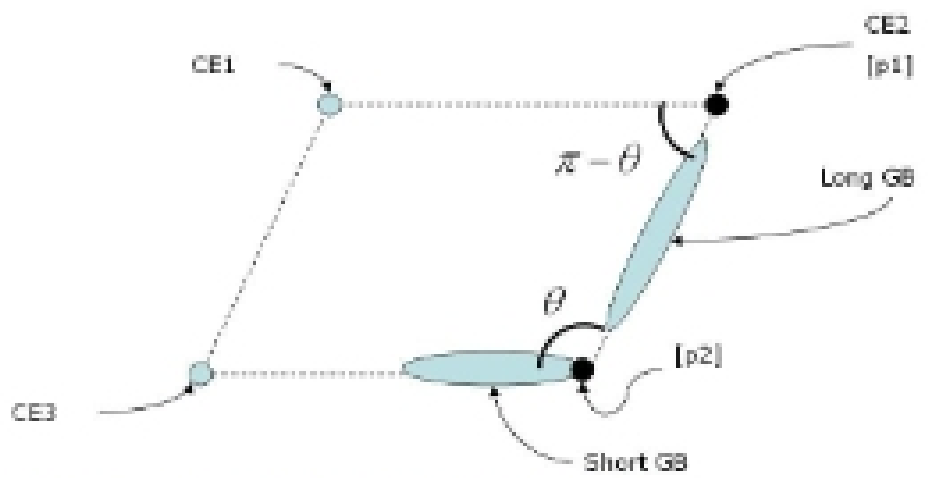

Fig. 4 Finding three CEs in parallelogram.

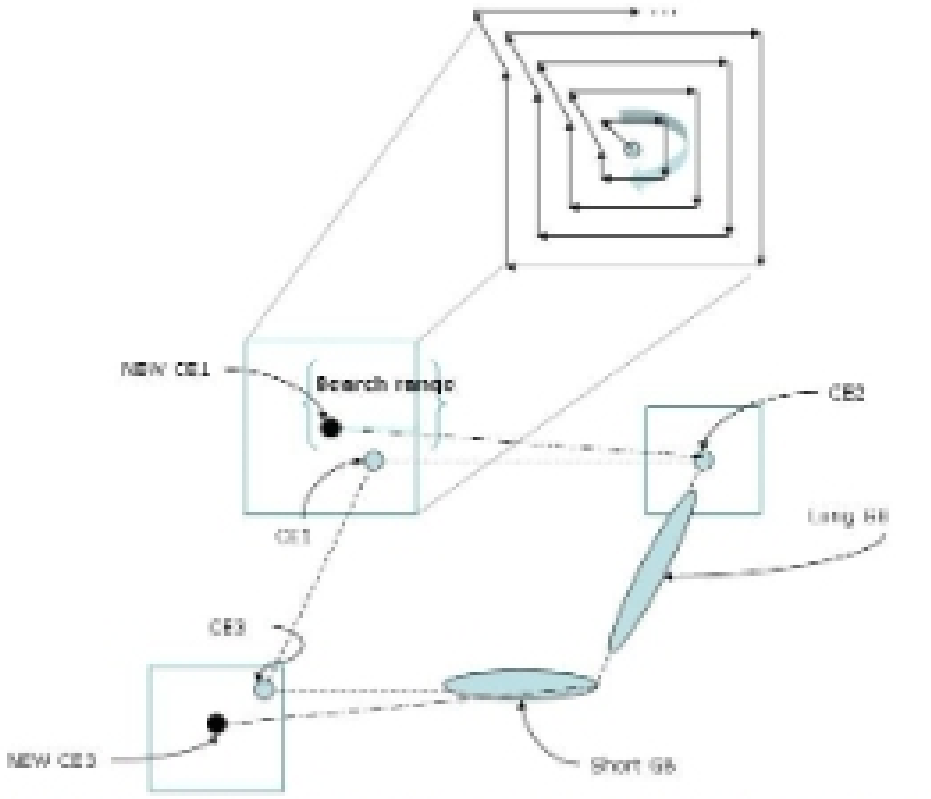

Fig. 5 Illustration of searching: center points of search are set with New CE1 and New CE3 directing a bit outward. The search is done following a spiral cord shape (increasing in x direction first, then in y direction, decreasing in x direction, then in y direction, and repeating this process until hit with value 1).

varying tilting along the marker it is reasonable to find from outside of the parallelogram. (Fig. 4)

In searching for the position of the CEs, we set search range for each CE. Each search range was decided according to its uncertainty: for CE1, it is 3 times the short axis of a GB, CE2 being 1 times the short axis of a GB, and CE3 being 1.2 times the short axis of a GB. Since CE2 is directly obtained on the line of its long axis of the long GB, it has the least uncertainty within the range whereas CE1 is most distant from any other certain points thus acquire the biggest search range.

The search is conducted in such a manner to follow spiral curve trajectory. It investigates the binary pixel starting from the center points (New CE1, CE2, New CE3) within its search range until it hits the pixel value of 1 in the binary image. (Fig. 5)

A set of marker after searching result is shown in (Fig. 6): two ellipses indicate long(green ellipse) and short(blue ellipse) GBs and three circles indicate the range of plausible CEs.

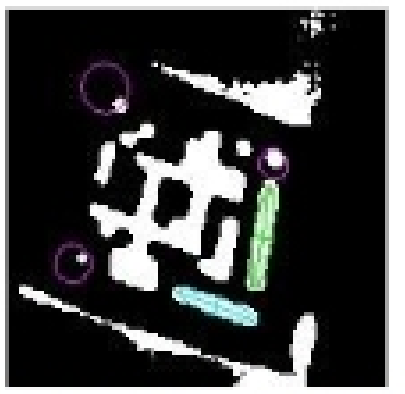

Fig. 6 (From training_6 image) A set of marker with two ellipses(green one is long GB and blue one is short GB) and corresponding three CEs with its search range symbolized as corresponding radius in each circle(in pink circles).

D. Projective mapping

One needs at least 4 points to describe a rectangle. We assumed that 11 points in any line of the marker is equally spaced and the lines are affine. Thus idea of projective mapping from code coordinate to image parallelogram is employed from [4]. After three CEs search regions being met with binary value 1, calculating the center point, i.e the mean value, is done within the region which includes that point.

With three vertices of the marker being found, it is better to assign another indicator point as the center of the short GB than calculating one of two peak points of the short GB (Fig. 7).

By adjusting coefficients in projective mapping method in [4], we obtain set of equations that map (u(m),v(n)) in known 11 by 11 marker coordinate into image coordinate, (x(m), y(n)),

$$x(m) = \frac{a\ u(m) + b\ v(n) + c}{g\ u(m) + h\ u(n) + 1}$$

$$y(n) = \frac{d\ u(m) + e\ v(n) + f}{g\ u(m) + h\ u(n) + 1}$$

with a to f coefficients:

$$g = \det\begin{bmatrix} \sum x & \Delta x2 \\ \sum y & \Delta y2 \end{bmatrix} / \det\begin{bmatrix} 0.8\,\Delta x1 & \Delta x2 \\ 0.8\,\Delta y1 & \Delta y2 \end{bmatrix}$$

$$h = \det\begin{bmatrix} 0.8\,\Delta x1 & \sum x \\ 0.8\,\Delta y1 & \sum y \end{bmatrix} / \det\begin{bmatrix} 0.8\,\Delta x1 & \Delta x2 \\ 0.8\,\Delta y1 & \Delta y2 \end{bmatrix}$$

$a = x1 - x0 + g\,x1$ $\quad d = y1 - y0 + g\,y1$

$b = x3 - x0 + h\,x1$ $\quad d = y3 - y0 + h\,y1$

$c = x0$ $\quad f = y0$

$$\sum x = 0.8\,x0 - 0.8\,x1 + x2 - x3$$

$$\sum y = 0.8\,y0 - 0.8\,y1 + y2 - y3$$

$\Delta x1 = x1 - x2,$ $\quad \Delta y1 = y1 - y2$

$\Delta x2 = x3 - x2,$ $\quad \Delta y1 = y3 - y2$

Through this process, 11 by 11 points in a marker are mapped into corresponding points in a marker regardless of its orientation. More specifically, the mapping performs well for