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].

II. Approach

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

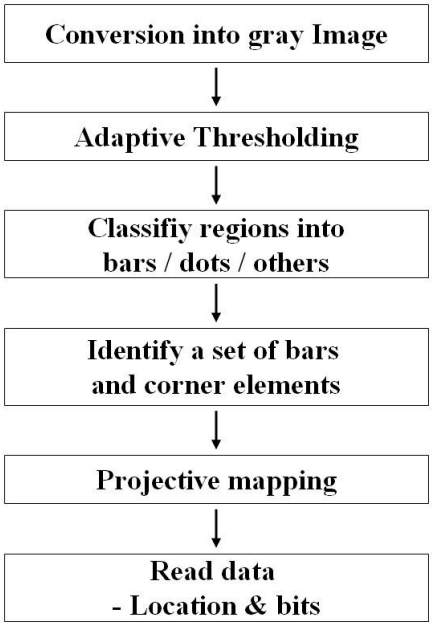

Fig. 1 Marker detection procedure flow chart

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 :

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

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

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.

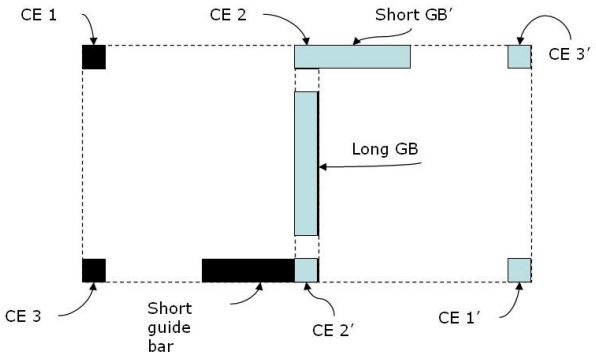

Fig. 3 Two markers are conceivable in each direction.

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

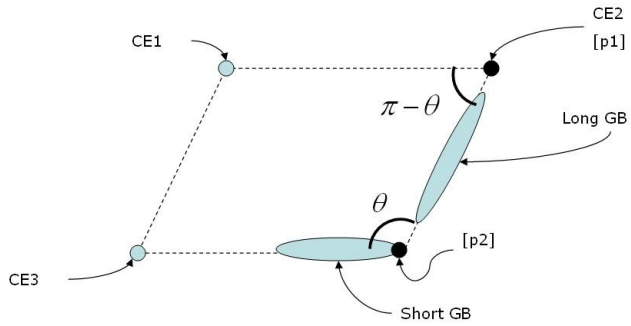

Fig. 4 Finding three CEs in parallelogram.

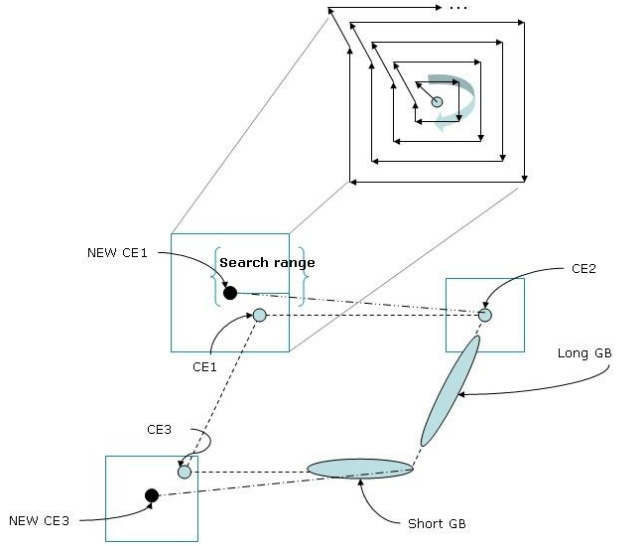

Fig. 5 Illustration of searching: center points of search are set with New CE1 and New CE3 directing a bit outward. The search is done following a spiral cord shape (increasing in x direction first, then in y direction, decreasing in x direction, then in y direction, and repeating this process until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

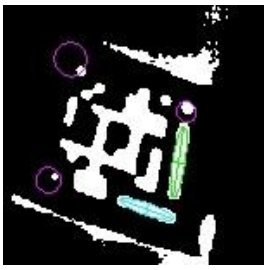

Fig. 6 (From training_6 image) A set of marker with two ellipses(green one is long GB and blue one is short GB) and corresponding three CEs with its search range symbolized as corresponding radius in each circle(in pink circles).

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:

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

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

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

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

$$c = x0 \qquad f = y0$$

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

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

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

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

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

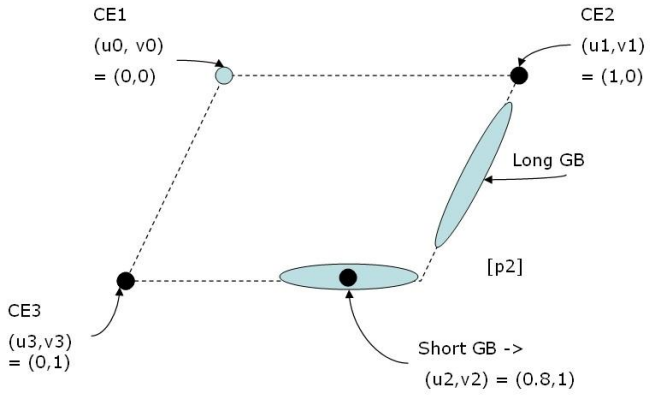

Fig. 7 Criteria points (u, v) in code domain to be mapped into parallelogram space.

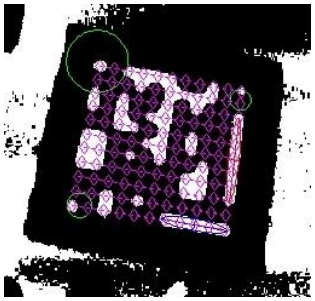

Fig. 8 (From training_12.jpg) This displays every 11 by 11 points in the image. Points inside the marker are mapped into image, represented by pink diamond symbols.

the parallelogram that is gone through 3 types of transformation — translation, scaling, and shearing.

The mapping result is shown in (Fig 8).

E. Read data- location and bits information

The data of a marker, consisting of 83 bits, are read from the result of the 11 by 11 positions via projective mapping. The positions from which we read data value are rounded off to the nearest integer. This process stores data moving from top to bottom, from left to right.

Origin is defined by center point of CE1.

IV. OTHER ATTEMPTS

After the initial implementation, we found that the processing time is directly proportional to the number of connected regions in binary image. Therefore, it is worthwhile to investigate some pre-filtering methods to reduce and eliminate regions out of interest. Such processing is expected to provide not only faster operation, but also enhancement to the side of robustness if non-marker regions can be successfully removed.

Those attempts are 1) median filter with added edge scheme and 2) color filtering. Here are two prominent attempts and why this is unhelpful to the algorithm described in III.

1) median filter with edge: Applying median filter to noisy mobile picture seems to be reasonable approach because it would reduce noise and preserve edges. The attempt is helpful in reduction of computational effort since it reduces the number of labels thus the process of identifying dots within the same label takes little time compared to the one without median filtering. Though it brings improvement in speed, however, it failed to catch a small CE2 point in training_10 image.

To compensate for the loss in detection, we added edge of grayscale image to binary image of inspection. As a result, applying sobel or prewitt edge with median filter captures all data in training images. Sobel and Prewitt were applied by using Matlab function and these methods display less noise and are faster than other edge operations such as Laplacian or Canny.

In terms of speed, however, edge detection process made the implementation slower. As seen in (Table 1), comparing the case 1 and case 3, this filtering process takes about 29.6sec in 12 training images whereas the code without filtering takes 10.6sec. All images display increase of execution time, which is especially most noticeable in training_10 image, it increased from about 1.3sec to 10.3sec, which is about 10 times slower. We can conclude that in certain picture, the performance is largely hindered by filtering thus should be avoided. Since those filtering procedure added no additional merit as a whole, it is better to skip the filtering process.

2) Color filtering: Inspired by the idea of color segmentation in face detection, it is expected that we can improve performance by discriminating similar color tone of a marker, or at least we can save time by skipping region with color far from our target, black. Compared to the significance of color segmentation methodology in face detection field, this case the issue is rather suppressing color that is outside of our interest, since the color of a marker is not rare like skin color.

Segmenting regions in YCbCr color space gives us merit because it separates luminance factor from chrominance information. From samples taken from 4 images, since Cb, Cr components are distributed according to Gaussian, the points whose color is similar to a marker have range specified by the mean and standard deviation of Cb and Cr, i.e at position n in YCbCr converted image by [3]:

$$T(n) = 1 \quad \text{if } Cb(n) \subseteq mean(Cb(marker)) \pm std(Cb(marker)) \\ \& \quad Cr(n) \subseteq mean(Cr(marker)) \pm std(Cr(marker))$$
$$T(n) = 0 \quad \text{otherwise}$$

That is, if the pixel value scaled in YCbCr domain lie between the range, the binary value is 1 and 0 otherwise. Thus the bar shapes in (Fig. 9) which originally have red color are no longer counted as candidates for bars.

Color filtering is then more tailored in order not to have any influence on the marker's pixel value. Thus the scaling factor is usually 5 to 10. Since the color filter introduces point spread noise, smoother using open and close morphological operations is used to first remove small 1-regions and then remove small

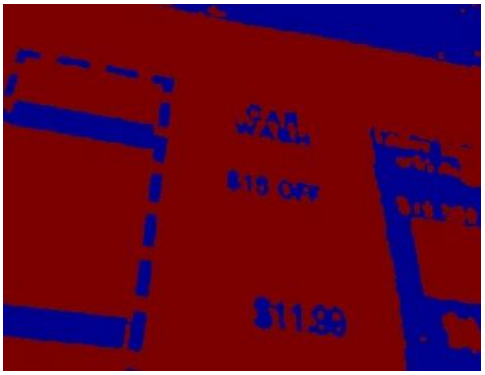

Fig. 9 (From training_5.jpg) Red color has value 1; blue color has value 0. The red and yellow color has been filtered out by a marker's CbCr sample range.

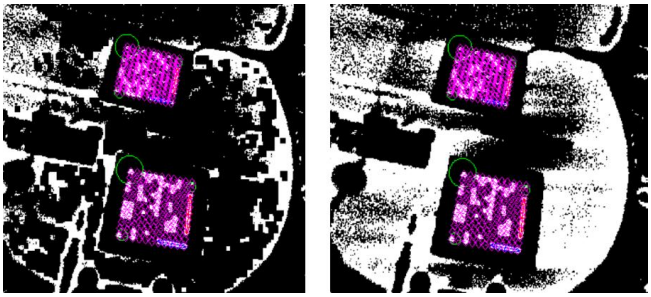

Fig. 10 Comparison of the images with or without color filter: one through color filter(left), and no color filter(right). This is the only case where color filtering increased speed performance.

0-regions inside the marker generated by white color in the marker.

However, the color filter does not improve the execution speed if not hinders the detection performance. Comparison of case1 with case 4 suggests that it improved speed only in one picture, training_12.jpg (Fig. 10). The reason is that the total number of labels is reduced from 1235 to 888 by decreasing noise in the binary image. However, in all the other cases imperfect characteristic of color filter increased the number of noise hence the number of labels increased; or remained the same when there is no red tone present. Since the training_2 and training_3 images have no red tone, use of color filter is only time consuming. In both images, color filtering decreased the speed about 0.52 sec.

Since there is approximately 50% increase in total execution time when activating color filtering, and since regions with non-marker color are sparse, there is no motive to put efforts into achieving higher separability. Hence for general application, it is better not to utilize color filtering process for now.

V. Results

The algorithms described above works accurately with all twelve training images. (Fig. 11) shows one example of final demonstration. Total 5 choices of algorithms involving main procedure and two kinds of attempts are tried as shown in (Table 1). Since adding median filtering or color filtering increased execution time, the main procedure will only be activated to detect markers.

TABLE I – Execution time of Visual Code Marker Detection

Training Images	Execution Time				
	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5
1	**1.844**	3.064	5.800	2.697	4.867
2	**0.585**	1.290	0.867	1.100	1.402
3	**0.394**	0.786	0.626	0.914	1.159
4	**0.976**	2.249	1.118	1.437	1.607
5	**0.800**	1.190	1.100	1.162	1.504
6	**0.525**	1.065	0.794	1.041	1.310
7	**0.734**	1.184	1.038	1.155	1.452
8	**0.512**	0.722	2.014	1.044	1.472
9	**0.660**	1.144	1.905	1.225	1.574
10	**1.278**	1.171	10.28	1.706	8.165
11	**0.353**	1.056	0.614	0.862	1.122
12	**1.961**	3.698	3.414	1.583	1.850
Total	**10.622**	18.619	29.579	15.924	27.485

Cases for table 1

	Fast mode	Median filter with edge	Color filter
CASE 1	O	X	X
CASE 2	X	X	X
CASE 3	O	O	X
CASE 4	O	X	O
CASE 5	O	O	O

Table 1 Comparisons between methods

The main procedure can be further improved by modifying labeling process, '[y x] = find(labeledImg == k);' in the code, since profile of the final code(described in III) demonstrates that 63% of the time is consumed in finding regions with specific label number. (cf. 11.4% consumed in adaptive thresholding, 4.8% consumed in the part where fitting ellipse is taking place.) Modified version of labeling function scans every pixel that is inside regions, and it stores (x,y) positions separately: for each row, it contains certain label from starting column to ending column. By this way, the execution time is reduced by 43% which is reduced from 18.619 sec (case 2 in Table 1) to 10.622 sec (case 1 in Table 1).

In comparing images by images, some image like training_10 scores the highest variance, whereas for several images (training_2, training_3, training_5, training_6, etc.) each takes less than 2 sec on all cases. Reducing execution time on special image like training_10 requires application-specific adjustment to better recognize markers.

More pictures that are taken by digital camera were examined.

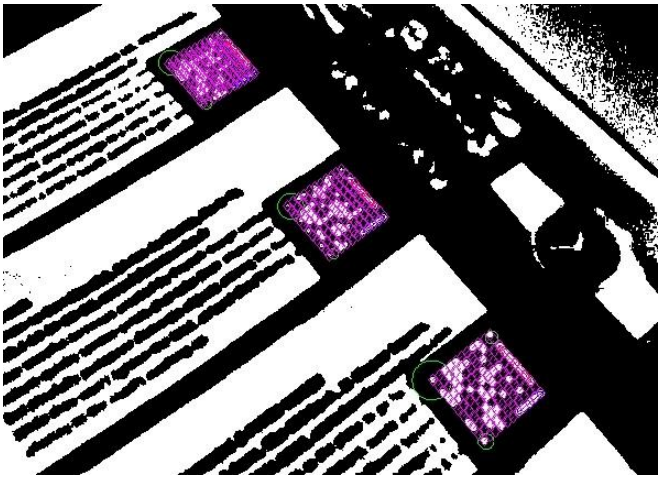

Fig. 11 (From training_10.jpg) The final result: three markers are detected. The upper most marker is the smallest marker of all training images.

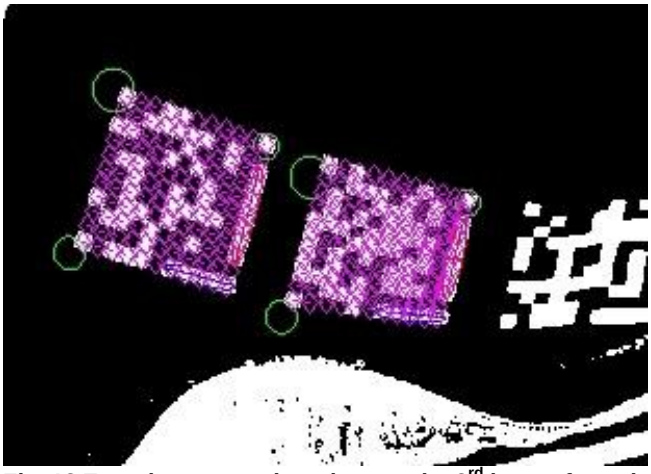

Fig. 12 Experiment on other picture : the 3rd image from the left is not detected because the marker is tilted in a way that projective mapping does not work.

The algorithm successfully detects markers of different markers sizes (within recognizable size) and of moderate tilt. (Fig 12) shows the result that failed in catching one marker which is tilted to the extent where parabolic edges are to be seen. To solve this issue, an algorithm needs to examine more parameters in measuring the amount of tilting using higher order terms with its warping model.

VI. Conclusion

Our detection algorithm performs with accuracy around 10.6 sec for the total of 12 given training images. Use of filtering in our algorithm proved to be inefficient in execution time's perspective.

Further work can be done on detecting markers with tilting which requires complex warping model.

Appendix

Division of work.

- Doe Hyun Yoon : Adaptive thresholding, Classifying regions, locating GBs & CEs, Reading data, speed optimization
- Sang Hui Ahn : Prefilter, Color filter, Fiding CEs, Projective mapping, ellipse fitting, testing other images

References

[1] M. Rohs, Real-world interaction with camera-phones, 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), 39-48.

[2] Wellner P. Adaptive thresholding on the DigitalDesk. EuroPARC Technical Report EPC-93-110, 1993.

[3] S.L. Phung, A. Bouzerdoum, D. Chai, A Novel Skin Color Model In YCbCr Color Space And Its Application To Human Face Detection, International Conference on Image Processing 2002.

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

[5] Ohad Gal, fit_ellipse.m file, Matlab Central [Online], Available: http://www.mathworks.com/matlabcentral/files/3215/fit_ellipse.m