An Algorithm for Visual Code Marker Extraction and Processing

Ahmet Altay, Emre Oto
Department of Electrical Engineering
Stanford University, Stanford, CA 94305

Abstract— **We present the design, implementation and performance evaluation of an algorithm devised for detection, extraction, and processing of visual code markers in images obtained via low-resolution cameras. The algorithm employs a myriad of techniques to locate, extract, and process the visual code markers for embedded binary data. The system performance has been assessed on a set of training images, and the algorithm has been observed to demonstrate high coherence in code marker detection and low bit-error-rate in binary data extraction. The algorithm also exhibits a significantly low execution time, which hints at its feasibility for implementation in low-power mobile devices.**

I. INTRODUCTION

Visual code markers have evolved as a bridge connecting the physical world to the cyber-world, as a tool that contributes to stronger human-computer interaction and ubiquitous computing [1]. Significance of code markers is constrained by camera phones. Thus any algorithm destined to process visual code markers must be capable of rapid processing of low-resolution, tilted, cluttered images under power and speed constraints [2].

In this paper, we present an algorithm for processing of low-resolution images acquired by a CCD camera containing visual code markers. The processing of the visual code marker entails the extraction of the binary vector embedded in the pattern. The algorithm takes a color image as input and returns the location of and the binary data in the visual code markers within the image.

The next section describes the morphology of the visual code markers subject to this paper. The mechanics of the proposed algorithm are presented in the following section. The performance of the algorithm on training set images is assessed, and the feasibility of implementation of this algorithm for real world applications is discussed in the remainder of this work.

II. BACKGROUND

The visual code marker that is processed by the proposed algorithm has been devised by Rohs and is described in [3]. The visual code marker is comprised of three salient parts, which are the fixed guide bars, the fixed corner elements, and the data area, as illustrated in Figure 1. The purposes of the fixed elements as envisioned in [3] are also indicated on the figure.

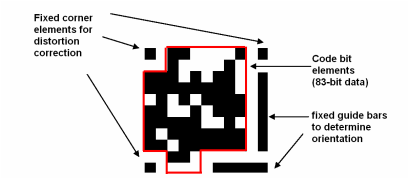

Fig 1. A typical visual code marker and salient features.

The vertical and horizontal guide bars have been intended for localizing and inferring the orientation of the code. The three squares at the corners are meant for detecting any distortion in the image. The data area encompasses the visual code pertaining to the actual code bits.

The visual code marker used in this study is an array of 11x11 elements, where an element is defined as a black or white square of the size of a fixed corner element. The vertical guide bar is 7 elements long and the horizontal guide bar is 5 elements long. The contiguous elements of the corner elements and the guide bar elements are also fixed and white.

III. METHOD

A. *Algorithm Overview*

The algorithm takes as an input a 24-bit colored image and performs the following operations on the image:

Marker extraction: The portions of the image that contain the visual code markers are detected and are isolated from the rest of the image.

Marker characterization: The parameters that characterize the stance of the visual code marker within the original image are extracted at this stage. These parameters are the rotation angle of the code marker and the position of its four corners.

Marker analysis: Given the rotation angle and the location of the four corners, the location of the code marker in the original image can be found and the binary data within can be extracted.

A system-level representation of the algorithm is presented in figure 2.

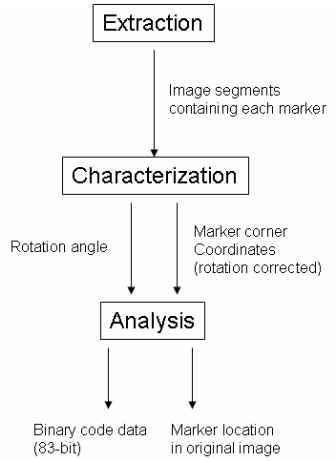

Fig. 2. Overview of the algorithm

We probe further into the mechanics of each operation separately in the following sections.

B. Marker Extraction

A flowchart illustrating the operation of the marker extraction algorithm is given in Figure 3.

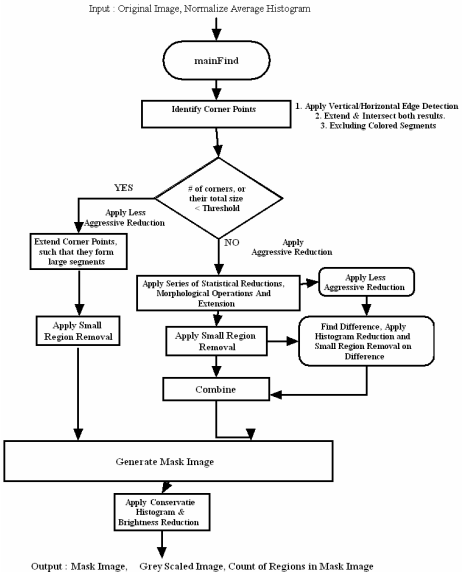

Fig.3. Flowchart describing the marker extraction algorithm.

As a first line of processing, the color images are grayscaled according to the scale-by-max algorithm, which scales each channel by its maximum component, as demonstrated in Figure 4. The generic idea in identifying the regions containing the visual code markers is corner detection. The primary motivation in applying this technique is that the visual code marker contains small white and black squares and is expected to contain a prominent number of corner points.

Fig. 4 a) Original Image, b) Image after Scale By Max Applied and c)Image after gray scaling

To find the corners within the image, horizontal and vertical edge detection are applied separately and the intersection points of the resulting edges are found. Because of the non idealities at this stage extra processing is required to get 1 pixel thick corner points. Figure 5 illustrates a dilated version of these corner points.

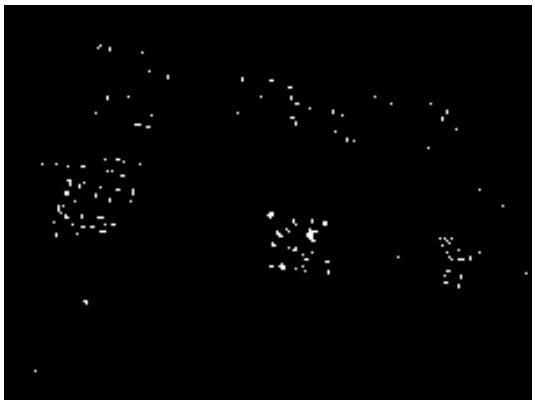

Fig. 5 Initial Corner Points

To attain 1-pixel thick corner points, the thickness of the detected lines are increased by dilation and then the intersection points are logically AND'ed.

It was observed that color could be utilized for a better estimate of corners; since the visual code markers are in black and white and any corners identified on the colored portions of the image can be discarded. However, the inherent chromatic noise makes color identification difficult. Algorithm 1 was utilized for the detection of colored regions:

Algorithm 1:

Grayscale the image by averaging 3 channels to get image 1

Grayscale the image by picking maximum of 3 channels to get image 2

For any pixel within image 1 and image 2
If (image 1) – (image 2) > threshold **then**
identify the pixel as a color pixel.
Else
identify the pixel as a black-and-white pixel.

If a pixel has a much higher channel value for any of its channels with respect to the averages of these channels, then one of the color channels dominates the other 2 channels, and the pixel is a colored pixel. A threshold of 40 was selected and was observed to be a satisfactory intensity separation. The colored regions identified on the example image of Figure 4 are shown in Figure 6. The corner points which corresponded to color pixels are logically subtracted from the image. (See figure 7.)

Fig. 6 Identified color pixels

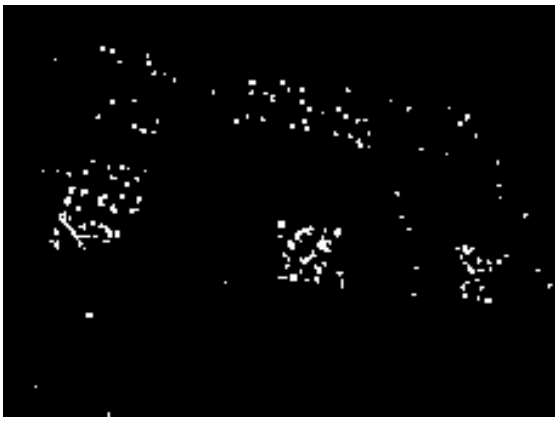

Fig. 7 Final Corner Points after dilation and colored region subtraction.

The corner points are next expanded until they combine and form larger regions. Once this is achieved, one of the two different strategies is chosen to identify which of these large regions the code markers are, and to expand these regions: For images that contain a small number of corner points or that have small regions after dilation a less aggressive algorithm is chosen. Otherwise, a more aggressive strategy is pursued.

The less aggressive strategy applies small region removal to eliminate the regions that contain a small number of corners and thus do not qualify as code markers. A final dilation can now be applied in confidence to combine the regions containing portions of a single visual code marker and to obtain a mask image. The output of the less aggressive strategy for the example image is illustrated in Figure 8.

Fig. 8 Output of less aggressive strategy

The more aggressive strategy first runs the less aggressive routine described above. Next, it runs a statistical reduction algorithm to identify the lines that do not pass through the code marker and then removes those lines. After some processing and another small region removal routine a mask image is obtained similar to that of the less aggressive strategy. Some further elimination of spurious regions is performed by taking the difference between the two masks and applying small-region removal. The output of the more aggressive strategy is shown in Figure 9.

Fig. 9 Output of less aggressive strategy

To further investigate if the identified regions correspond to a code marker, a histogram checking algorithm is employed. Prior to histogram checking, an average reference histogram pertaining to a generic code marker is needed. . The average histogram conveys a sense of the intensity distribution a typical code marker would have. This histogram is formed by processing the training set images and taking the average of their intensity histograms in grayscale. A typical code marker histogram and the average histogram obtained from the training set are illustrated in Figure 10. The histogram of every region falling into the mask in the original grayscale image is compared with the reference histogram to measure histogram distance.

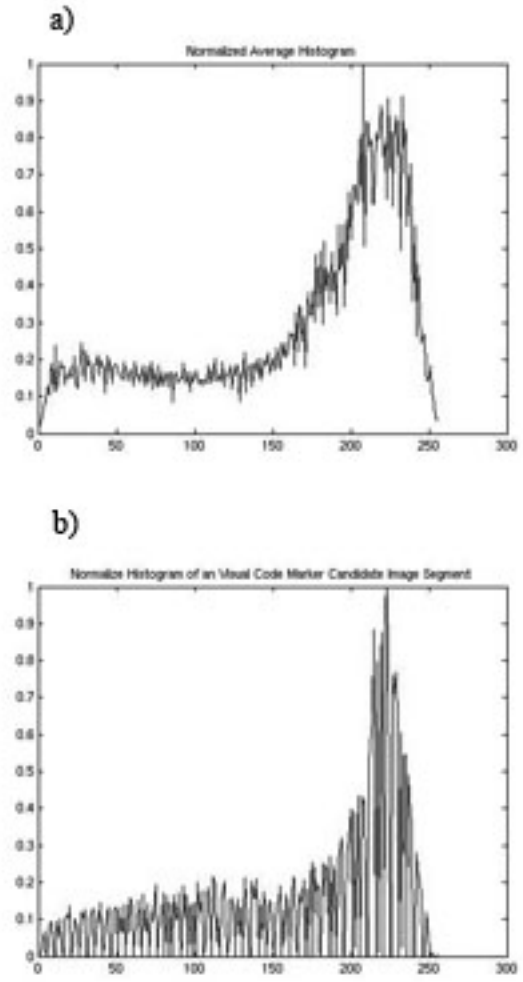

Fig. 10 Comparison of a) normalized average histogram and b) an actual visual code marker histogram

Two measures were used to quantify histogram distance: The first of these is the sum of the L^2 differences obtained by bin-by-bin subtraction. The second measure which is due to Smith et al, is histogram intersection, $d_I(h,g)$ as given in (1).

$$d_I(h,g) = \frac{\sum_{m=0}^{M-1} \min(h[m], g[m])}{\min(\sum_{m_0=0}^{M-1} h[m_0], \sum_{m_1=0}^{M-1} g[m_1])} \qquad (1)$$

In the above expression h[m] and g[m] are the two histograms with M bins that are being compared. The two measures combined provide an insightful measure of being a code marker. The threshold values for each of these distance measures have been set through empirical studies. Then, regions having histogram distances larger than the thresholds are eliminated.

Histogram based elimination may still leave some spurious regions within the image, which can be discarded using a contrast elimination algorithm. This algorithm

leaves regions with high contrast values intact and acts on regions of lower contrast. In applying this technique, the first step is to modify the image histogram to increase the brightness, which results in regions with low contrast to become mostly white. Such regions are then eliminated by thresholding.

From the mask image, for each of the regions, an approximate region that includes the code marker is extracted. These image segments corresponding to individual code markers are fed as an input to the marker characterization routine.

C. *Marker Characterization*

A high-level view of marker characterization operations is depicted in Figure 11.

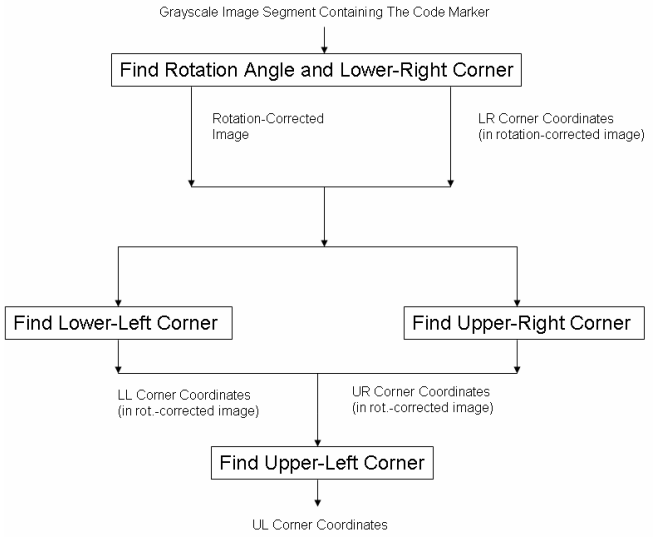

Fig.11. Overview of the marker characterization routine.

Marker characterization progresses by first finding the rotation angle of this image segment, compensating for the rotation, and generating a rotation compensated image. The routine that identifies the correct orientation of the image also returns the coordinates of the lower-right (LR) corner of the code marker that has the proper alignment. By proper alignment we imply the orientation in which origin of the code is at the upper-left of the segment.

Having generated the rotation-corrected image and found the coordinate of the lower-right corner of the code marker within, these coordinates can be used to find the coordinates of the upper-right (UR) and lower-left (LL) corners of the code marker in separate routines for each. The location information of the UR and LL corners of the visual code marker can then be used to localize the UL corner.

It must be noted that all operations within the marker characterization routine are performed after the grayscale input image segment has been tresholded. The image has to be converted into this logical form since the operations that follow utilize such data. Figure 12 demonstrate the gray scaled input image and its thresholded logical version for the example image.

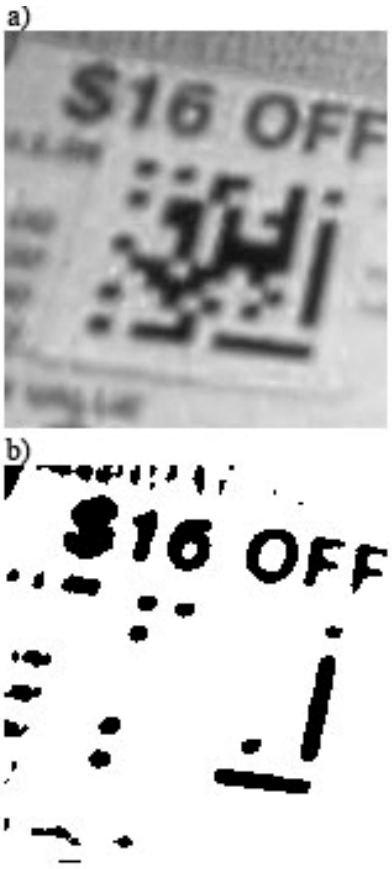

Fig. 12 a) Gray Scaled Input Image b) Its thresholded version.

We next elaborate on the steps taken in finding the parameters.

Rotation angle and LR-corner coordinates: The steps involved in finding the rotation angle and LR-coordinates are epitomized in Fig.13.

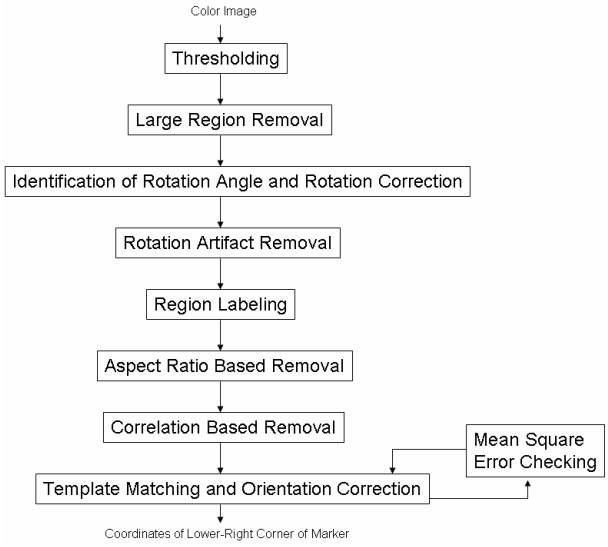

Fig.13. Finding the rotation angle and LR-corner.

Following some processing on the tresholded image, a template matching algorithm is executed to identify the correct orientation of the code marker. The generic gray template image is shown in figure 14.a. The orientation correction routine is necessary since Hough transformation cannot differentiate between orientations that differ by 90 degrees. (See figure 14 b-e for different orientations of template). From the four possible orientations corresponding to the same rotation angle, it is desired to identify the one in which the horizontal guard line is aligned with the horizontal axis and the vertical guard line is aligned with the vertical axis. Finding the proper orientation of the code marker is crucial in localizing the corners of the code marker correctly.

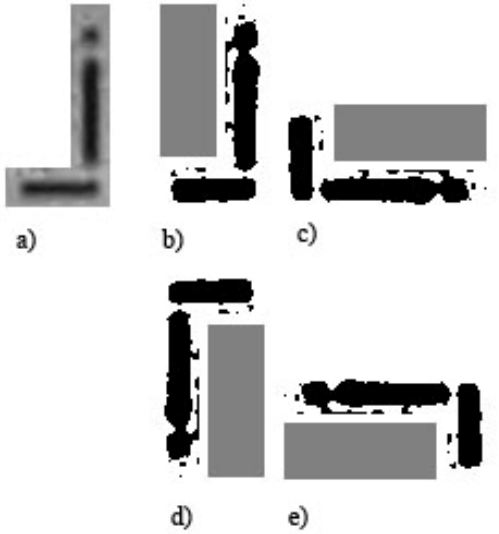

Fig. 14 a) Generic Template Image b-e) Thresholded versions of generic template image having different orientations.

Before executing the orientation correction routine, the clutter within the image segment and parts of the visual code marker not involved in the orientation correction have to be eliminated. The objective of this elimination is to remove everything but the guard lines of the visual code markers.. Although the removal of all artifacts can seldom be achieved, the elimination takes out most of the artifacts to thwart any errors in template matching. Figure 15 illustrates the image after artifact removal

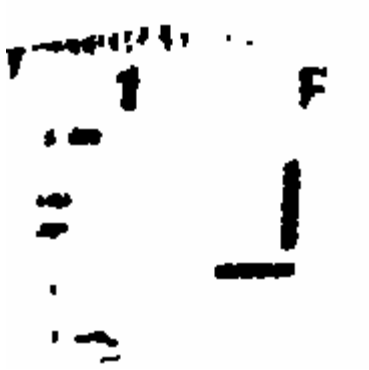

Fig. 15 Image after artifact removal

The elimination techniques applied are illustrated in sequence in Figure 13. Once the rotation of the visual code marker portion has been identified via Hough transformation, the image is rotated to align the visual code marker with the horizontal and vertical axes. This rotation results in a large black frame surrounding the now oblique image. This black frame is removed by a frame removal algorithm.

Following this removal, large connected regions that arise inside the data block are eliminated. Through empirical studies on the training set, a good threshold for large region elimination has been determined to be 5% of the size of the logical and rotated image. This threshold has thus far proved to be an excellent separation between the guard lines and other regions.

Another extra measure with which black regions except for the guard lines can be distinguished and eliminated is the aspect ratio. The aspect ratio of the guard lines have been measured on the processed training set images and it has been observed that for the vertical guard line the aspect ratio is between 2 and 9 and for the horizontal line it is between 0.125 and 0.66. Using these thresholds, region labeling can be applied and any region that has an aspect ratio outside of these two intervals can be removed from the image.

The specified intervals for aspect ratio allow for some safety margin to prevent the elimination of guard lines. In most cases these intervals do not suffice for thorough elimination of artifacts and an extra measure for elimination is needed. This extra measure is the correlation coefficient of the row and column indices.

It is known that the correlation coefficient between two vectors is zero if there exists a fully flat or a fully vertical line. Similarly the correlation coefficient is 1 if the two vectors are linearly dependent. For any other value of the correlation coefficient, the two vectors are nonlinearly related. Ideally, if the rotation of the code marker could be fully compensated and the morphological operations on the image yielded straight lines for the guard lines, it could be possible to discard all regions with correlation coefficients different than 0.

This is not the case in reality, since the rotation of the code marker cannot be fully compensated and the guard

lines do not retain a linear shape after logical operations. On the other hand, the guard lines still appear close to being straight lines and exhibit a correlation coefficient significantly close to 0 or 1 depending on the extent of rotation-correction. Then, it is concluded that any region that demonstrates a coefficient close to 0.5 cannot be a guide line and may be discarded. Studies on the training images have demonstrated that removing regions with correlation coefficients between 0.4 and 0.6 is an efficient yet reliable way of exploiting the correlation measure.

Template matching runs the template image through the original image and returns the coordinate of the pixel at which the sum of the mean square of the difference between the template and image pixel values, i.e. the mean square error (MSE) is minimum. The minimum MSE serves as a figure-of-merit to determine whether the orientation of the code marker is correct.

We have identified that a minimum MSE value smaller than 30 always signifies that the orientation of the code marker is correct. For any value of the minimum MSE larger than 30, template matching is repeated in the three other 90 degree rotations of the template. Following this, the orientation that results in the lowest minimum MSE is chosen. Once the correct orientation of the code marker has been identified, extra rotation due to the orientation is added to the rotation angle obtained through Hough transformation.

To leverage the computational complexity involved at this stage, template matching has been performed once in every three pixels of the image. This sampling has had no effect on the quality of the outcome, yet has resulted in a crucial reduction in execution time, namely by 1/9.

Finding the proper orientation of the code marker also allows one to find the coordinates of the LR-corner in the rotation corrected image. An approximate value of these coordinates can be found by adding the dimensions of the templates to the coordinates of the point of minimum MSE, and then subtracting a small, fixed offset to bring the point to the close vicinity of the horizontal guide bar. Once this has been accomplished the exact coordinates of the LR-corner can be found by region labeling and application of Algorithm 2 (on the image processed by the elimination routines):

Algorithm 2:

If approximate LR-corner is a black pixel **then**
  get the label of the region containing approximate LR-corner.
  exact LR-corner is the LR-corner of the region found above.
Else
  create a 5x5 search matrix.
  populate the rows and columns of the search matrix with the rows of the image at (row(approx. LR-corner) – 2) to
    (row(approx. LR-corner) + 2) and
    (col(approx. LR-corner) – 2) to
    (col(approx. LR-corner) + 2) respectively.
  find the indices of the search matrix that are black.

  If there is at least one black element in the search matrix **then**
    find the index of the closest black element in terms of Euclidian dist.
    get the label of the region containing the index found above.
    exact LR-corner is the LR-corner of the region found above.
  Else
    LR-corner coordinates are equal to approx. LR-corner coordinates.

The exact coordinates of the LR-corner are defined to be the coordinates of the LR-corner of the horizontal guard line. The algorithm presented retrieves the label to the horizontal guard line and then identifies its LR-corner as the exact LR-corner. The above algorithm assumes that the approximate LR-corner falls at most to two-pixel proximity of the horizontal guard line. If the approximate LR-corner falls out of the horizontal guard line, a 5x5 search matrix is used to identify the black entry closest to the center as belonging to the horizontal guard line. If by any chance the approximate LR-corner falls to more than two pixels away from the guard line, the approximate LR-corner is taken as the exact LR-corner since no better estimate can be provided.

UR-corner coordinates: Knowing the LR-corner coordinates one can isolate a strip (see figure 16) made up of the columns to the left and right of the LR-corner, in which the UR-corner element is sure to exist. Through experimentation on the training set, the optimum configuration of this strip was discovered to include the columns of the -rotation and orientation compensated – image between indices (col(LR-corner) - 10) and (col(LR-corner) + 10).

Fig 16. Strip of columns image used for UR-corner detection.

Once this strip has been isolated, a profile of it over the rows is obtained by summing over the columns. The profile is then traced vertically starting from the horizontal guide line and working towards the UR-corner element. To ensure that the algorithm starts from the horizontal guide line each time, the starting index is set to a black pixel within the horizontal guideline that has previously been obtained through Algorithm 2. Assuming we have the profile for black dots, starting from the horizontal guideline we are expecting to traverse 1-bit length of black, 1-bit length of white, 7-bit length of black, 1-bit length of white and finally a 1-bit length of black in order. Knowing the starting index and the bit pattern that the profile corresponds to, one can obtain the location of the UR-element. A typical profile of a strip is shown in Figure 17.

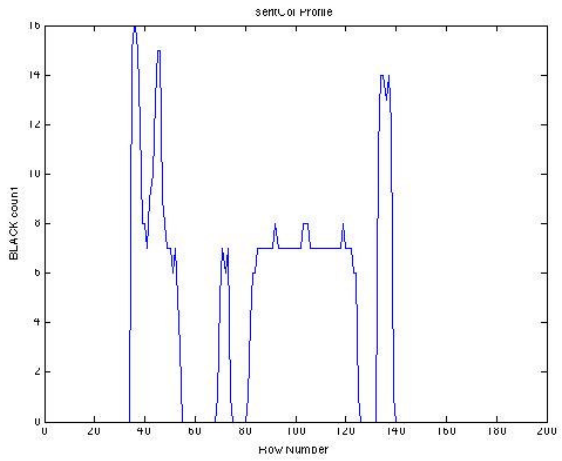

Fig.17. Typical profile of the strip used to find the UR-corner. The profile indicates the sum of the black pixels in each row of the strip.

The profile only provides information on the row of the UR-element. Having obtained the row information, a search matrix is be defined in the vicinity of this row. This matrix is summed over the rows to obtain the total number of black pixels in each column. The column that has the largest count of black pixels is expected to contain a line within the black UR-element. This column and the row previously found from the profile are used to identify the region that corresponds to the UR-element after region labeling. The UR-corner of the code marker is defined as the UR-corner of this element.

Lower-Left Corner: This part of the algorithm progresses by estimating the slope of the horizontal guide line. It is known that the horizontal guard line is 5 bits long, and this allows the calculation of the length of a bit along the line with the estimated slope. Since the LL-corner element is the 11^{th} bit to the left, the lower left corner may then be found by extrapolating the line passing through the guard line with the slope estimated previously until the position where the 11^{th} bit must lie. See figure 17 for an overview of the strip image.

Fig 17. Strip of rows image used for LL-corner detection.

At this location, one can once again search a local square of pixels to find the location of the LL-corner element. This search can be done by performing erosion. Once the location of the LL-element has been found, the corresponding region can be grabbed through region labeling. The LL-corner of the code marker is then defined as the LL-corner of the element.

Upper-Left Corner: Having found the UR and LL corners of the image, the UL corner element can be sought in a much smaller area within the image which can be delimited using the coordinates of the UR and LL corners. The smaller rectangular region (see figure 18) is given as the rows and columns of the image (row(UR-corner)-30+k) to (row(UR-corner)+20) and (col(UR-corner)-30+l) to (col(UR-corner+l) respectively, where k and l are offsets arising to uncompensated rotation and are given in (2) and (3).

$$k = \text{row (LL-corner)} - \text{row (LR-corner)} \qquad (2)$$

$$l = \text{col (UR-corner)} - \text{col (LR-corner)} \qquad (3)$$

Fig 18. Neighborhood of UL corner used for detection. Almost circular black region closer to center shown in figure is the actual UL corner for this specific image.

Region labeling is performed within this rectangular region. It is expected that the UR-corner element is closer to the intersection of the horizontal line extending from the UR-corner and the vertical line running through the LL-corner. Thus within the rectangular region, it is inferred that the black region that is closest to the intersection point of these two lines is the UL-corner element of the code marker. Median of the region is selected as reference origin in distance calculation. The UL-corner of the region that is closest to the intersection of the above-mentioned lines is taken as the UL-corner of the code marker.

D. Marker Analysis

This routine takes as input the image segments provided by marker extraction, and also obtains the required rotation angle as well as the four outmost coordinates of the code marker from the marker characterization stage.

Using this information, the algorithm generates a rotation-corrected and histogram adjusted image, and then applies a projective two-dimensional perspective transformation with bicubic interpolation. This perspective mapping is necessary due to the sheer and perspective effects in both the horizontal and vertical directions. Figure 19 provides a comparison of the input image and the transformed image:

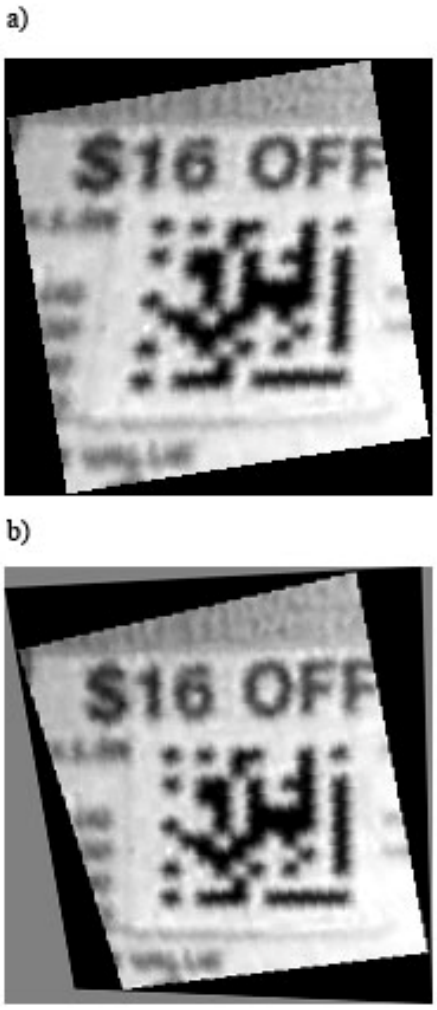

Fig 19. a) Input Image b) Transformed Image

The transformation method implemented in this work uses the given parameters and generates an image with non-integer indices in [0,1], with the corners of the code marker at (0,0),(0,1),(1,0),(1,1). These corners are the individual mappings of the four corners identified in the marker characterization routine. Following this, the image is linearly divided into 121 equal squares (cells). Figure 20 shows the identified code marker region. UL corner of each cell is marked by a blue dot.

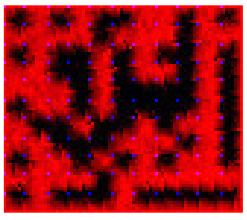

Fig 20. Identified code marker used for bit detection, red channel ranging from 0-255 (black – red) shows the gray value for that pixel. Blue dots shows the UL corner for 121 bit cells.

To read the bits off the image, the grayscale image is used in a weighted-averaging scheme. For each cell, the center pixel is found, and letting d be the Euclidean distance for any pixel in the cell to the center pixel, each pixel is summed with weight w=1/1+d. The weighted sums are then normalized to fall between 0 and 255.

A thresholding algorithm is needed to convert these values into logical values. The threshold algorithm requires a reference value for pure black, which can be obtained by reading off the intensity value corresponding to the cells that fall to the four corners of the code marker. In practice, however, each of these four cells will have a different intensity value, and it is not possible to come up with a single correct reference value.

This fact has led to the development of a position dependent algorithm that applies a different threshold to each cell via a bilinear interpolation formula. Letting a be the distance to the bottom edge in the orthogonal direction and b denote the distance to the right most edge, the threshold for the cell represented by the coordinate pair (a,b) is given by (4).

$$th = ab.(ul) + b\,(1-a).(ll) + a(1-b).(ur) + (1-a)(1-b).(lr) \quad (4)$$

where *ul*,*ll*,*ur*,*lr* represent the gray values of the upper-left, lower-left, upper-right, and lower-right pixels respectively. This local thresholding algorithm has proven to be much more efficient than setting a single threshold for all cells.

Once the thresholding has been accomplished, the binary data lying in each cell is clustered, and organized into a real 83-bit-long data segment.

The marker analysis routine also yields the position of the row and column of the LL-corner of the original image. These coordinates are referred to as the code coordinates and mark the location of the code marker on the image.

The data needed for this calculation are the rotation angle and the coordinates of the LL-corner in the rotation-compensated image. Using these and a record of the operations that have been performed on the image to obtain the coordinates, the operations can be reversed to obtain the coordinate of the LL-corner on the true image.

IV. Performance Evaluation

The performance of the proposed algorithm was assessed on the training set. It was observed that the implementation was quite satisfactory, owing in part to the fact that the algorithm had been trained specifically for this particular set of images. Over the training set all code markers were detected, and the cumulative bit error rate was less than 1 percent. The performance metrics obtained on the training set are presented in Table 1.

Table 1. Peformance Metrics for the Training Set

Number of Images	12		
	Actual	Correctly Detected	False Alarm & Repeats
Number of Visual Code Markers	23	0	0
Number of Data Bits	1909	1892	
	Total	Avg. Per Image	Avg. Per Code Marker
Execution Time	30 s	2.5 s	1.3 s
	Code Markers		Code Marker Data
Detection Rate	100%		> 99.1%
Error Rate	0%		< 0.9 % (Less than 1 bit per code marker)
	Max	Min	Average
Bit Error Rate for Visual Code Markers	7	0	< 0.74

In terms of execution time, the algorithm was seen to process an average code marker within 1.3 seconds, and spends 2.5sec on average for a single image. This execution time is 24 times faster than the project requirement.

The algorithm was tested on other images obtained via a CCD camera and similar to the training set. Although the performance of the algorithm diminished slightly on these images as compared to the training set, the algorithm was still capable of locating the code markers and acquiring the binary data on most images. The downgraded performance on the images outside of the training set are also believed to be due to the severe chromatic noise inflicted by the particular camera with which the images were acquired, as well as to camera calibration discrepancies.

V. Conclusion And Future Work

Investigations on the training set have indicated that the thresholds and parameters that govern the algorithm are heavily dependent on the characteristics of the training images provided. This results in a detriment of quality when the algorithm trained on one set of images is attempted to be used on another set. This detriment can be alleviated by using a larger training set which includes images taken under different illumination conditions, and by different cameras.

Although it has been demonstrated through performance studies that the algorithm in its current state has significantly small execution time on a personal computer, and is thus supposedly viable for implementation in mobile devices, the operational complexity involved may be overwhelming in the mobile realm.

This work has also granted the authors with the opportunity to put concepts such as template matching, morphological operations, linear system theoretical tools such as correlation, rotation, and various statistical methods into practical use in a fruitful learning experience.

References

[1] M.R. Rohs, P. Zweifel, "A Conceptual Framework for Camera Phone-based Interaction Techniques," presented at Pervasive Computing: Third International Conference, Munich, Germany, May 8-13, 2005

[2] M.R. Rohs, B.Gfeller, "Using Camera-Equipped Mobile Phones For Interacting With Real-World Objects," *in Advances in Pervasive Computing*, pp. 265-271, Vienna, Austria, Apr. 2004.

[3] M.Rohs, "Real World Interaction With Camera Phones," presented at 2nd International Symposium on Ubiquitous Computing Systems, Tokyo, Japan, Nov. 2004.

[4] J. R. Smith and S.-F. Chang. "Tools and techniques for color image retrieval.," in *Symposium on Electronic Imaging: Science and Technology - Storage & Retrieval for Image and Video Databases IV,* volume 2670, San Jose, CA, February 1996.

Appendix : Log of the Project

WHO	Work
	0. Infrastructure
Ahmet	a. Gray Scaling, Thresholding Functions
Both	b. Rotation Function using Hough transform and maximum selection
Emre	c. Scale by max function
	1. Visual Code Markers Identification
Emre	a. Generation of an Average Histogram
Ahmet	b. Mean Square Histogram Difference Metric Function.
Emre	c. Minimum of minimums Histogram Difference Metric Function
Ahmet	d. Region Identification
Ahmet	i. Noise reduction, grey scaling and thresholding.
Ahmet	ii. Corner Detection
Ahmet	iii. Finding Colored Region and Re-evaluating detected Corners
Ahmet	iv. Applying Morphological Operations
Ahmet	v. Applying Statistical Reduction
Ahmet	vi. Applying logical operations to different masks coming from different corner detection algorithms
Ahmet	e. Region Reduction
Ahmet	i. Small Region Removal
Ahmet	ii. Using Histogram
Ahmet	iii. Using Contrast
Ahmet	f. Region Labeling and Region Counting
	2. Visual Code Marker Characterization
Ahmet	a. Small Segment Extraction from the large image using the mask. Extraction size decision
Emre	b. Noise and Non Guiding Element Artifact Removal
Emre	i. Using Correlation Techniques
Emre	ii. Median Filtering
Emre	iii. Large Region Removal
Emre	c. Template Matching for alignment analysis (90 degree rotations)
Emre	d. Template Matching for lower right corner identification.
Emre	e. Upper right corner identification
Ahmet	i. Using Profile for finding row
Emre	ii. Using small region search for column
Emre	f. Lower left corner identification
Ahmet	i. Estimation using linear models
Emre	ii. Small region search for exact position.
Emre	g. Finding upper left corner
Emre	i. Using intersection of two neighboring corner position and linear model estimation
Emre	ii. Small region search for exact position.
Emre	h. Finding outer most corners by
Emre	i. Small region search
Emre	ii. Image labeling
	3. Visual Code Marker Analysis
Ahmet	a. Applying Transformation
Ahmet	b. Dividing transformed image into 121 cell regions
Ahmet	c. Applying value evaluation for each cell
Ahmet	i. Find weighted sum of grey with more emphasis on center for each cell
Ahmet	ii. Linearly map results of all cells to 0-255 region
Ahmet	d. Find a thresholding value for each cell
Ahmet	i. Apply position dependent function
Ahmet	e. Threshold the image and extract 83 bits as required
	4. Visual Code Marker Origin Analysis
Ahmet	a. Using known linear offset, rotation angle and the position of the origin in the rotated small image, find the origin in the original image.
	5. Test and Analysis
Both	a. Tests for Code Marker Region Finding, related Parameter Characterization
Emre	b. Tests for Corner Finding in the Code Marker, related Parameter Characterization
Ahmet	c. Tests for Bit Reading and Parameter related Characterization
Ahmet	d. Generation of a 2nd training set and test
	6. Report
Emre	a. Actual report writing and formatting
Ahmet	b. Figure and plot generation for report.