Detection of Visual Code Markers

Murat Aksoy, Tolga Çukur, Yusuf Özuysal
Department of Electrical Engineering, Stanford University

Abstract

In this paper, we present a recognition algorithm to identify visual code markers embedded in a camera-phone image. The algorithm is capable of detecting code markers with arbitrary rotation, tilting and perspective distortion. It can also read the marker bits by mapping the image coordinates to the code marker coordinates. The algorithm can effectively recognize visual code markers in a time-efficient manner, without making any assumptions about the characteristics of the camera used for capturing the image.

1. Introduction

The task given for the project was to recover positions and information contained in an arbitrary number of code markers (between 1-3) in an RGB image taken by a cell phone camera. Through the work done, a novel algorithm was devised to accomplish this task. The algorithm systematically locates the code markers in the image and extracts the necessary data strictly avoiding false positives. A brief explanation of algorithm steps is given below.

The first step of the algorithm is to recover candidate region locations in the RGB image where the guide bars and the corner elements may be located. This is done using a procedure which involves edge-detection for finding the region boundaries and the use of an algorithm which finds and fills in the innermost contours in the edge detection output. Then these regions found, are systematically filtered until only the guide bars and corner elements for the code markers are left.

The first elimination step looks at the eccentricity of all the images and classifies candidates for long and short guide bars. Then the next step looks at the area ratios and the distances of all possible long and short bar candidate pairs and eliminates the pairs which don't match the specific criteria given. Then possible corner element locations for the pairs which have passed these verification steps are calculated and these corner elements are checked for eccentricity and area constraints. In this last step also the angle between the long and short bars is checked to eliminate the pairs composed of two in line bars.

After the above steps, it is assumed that the only candidate pairs left are the long and short guide bar pairs for the code markers in the image. Their locations and also the locations of corresponding corner elements are used to subject the code marker regions in the image to an inverse projective transformation. Lastly, the output of the transformation is processed to obtain the data bits in each code marker.

The algorithm thus finds an arbitrary number of code markers in an RGB image and returns the data bits contained with the marker positions in the image.

2. Algorithm Steps

The input to the algorithm is an RGB image of arbitrary size. This RGB image is processed to extract the positions and the bit information contained in the code markers, by the algorithm implemented whose steps are discussed below.

2.1. Generating the Initial Gray Scale Images

As the first step of the algorithm, we generate a gray scale image from the RGB input image. As will be discussed below, an edge detection algorithm is used to process this grayscale image and the edge-detected images will then be used to define the candidate regions in the image. So it is desired that the output of this grayscaling and edge-detection process has clear and connected edges. To accomplish this, two different grayscale images are generated from the RGB image. The first one is the average of the green and red channels and the second one is the average of all three RGB channels.

The average of only the green and red channels is needed, since the blue part of the image is a low-pass image, it contains less information and using it for edge detection may lead to errors blurring the average. However if only these two channels are used, there may be cases where the edges are not connected, since all the image information is

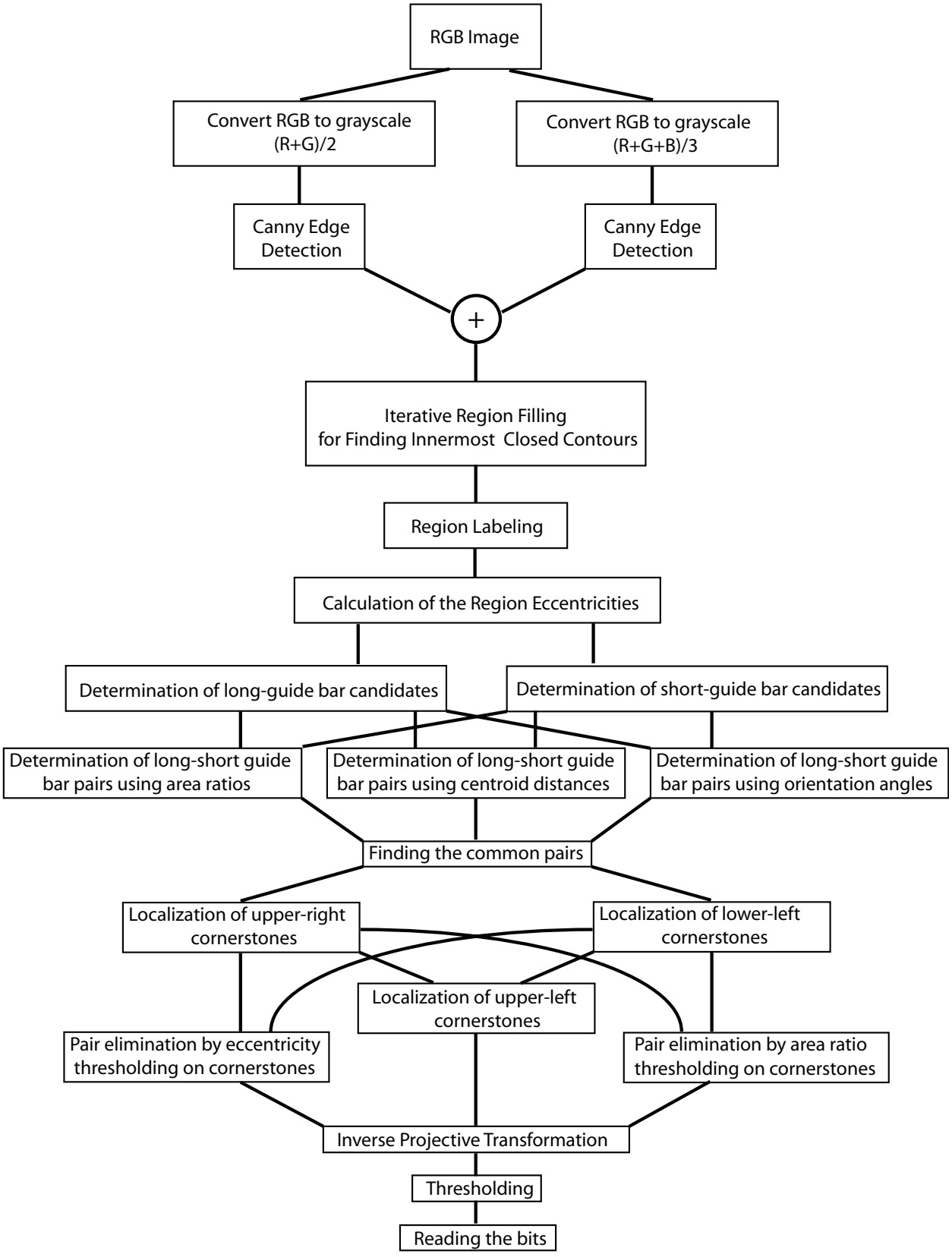

Figure 1. Flow diagram of the algorithm.

not used. Thus it is desired to use the information in the blue channel, while making sure that its low-pass characteristic does not cause any blurring and errors in the final output.

As a consequence of the above observations, it was decided to generate two separate grayscale images and feed them separately to an edge-detection algorithm.

2.2. Edge Detection

The two grayscale images are edge detected by a canny edge-detection algorithm. It is expected that each edge-detected image will contain information required for the construction of a binary image having closed contours around the marker guide bars and corner elements. For images having dim lighting conditions in the region where a marker is placed, the information in the first edge detection output will be the most useful. However when any contour in the first edge detection output is not closed, the low-pass characteristic of the second grayscale image will cause the edge-detection output to have a closed contour for that region. Thus to combine all the information in the two edge-detection output images, another binary image is generated as the output of an entry-wise OR operation between the two edge detection outputs.

This binary image is expected to contain closed contours around the guide bars and the corner elements and will be used as an input to the algorithm for finding the innermost closed contours and define the binary regions to be used for candidate selection and elimination.

2.3. Finding the Innermost Closed Contours

After the binary image with edges around the guide bars and corner elements is obtained, it is required that the inner pixels of the contours corresponding to these regions are filled. If it is assumed that no closed contour has another closed contour containing it, this is a very trivial task. The image can be filled with ones starting from the corner of the image and then can be negated causing every closed contour to be filled. Here note that we can assume there is not any region boundary corresponding to exactly the corner where we start the filling operation. However, in general the region to fill may be bounded by closed contours that are contained in an arbitrary number of other closed contours. Thus the requires task is to fill in only the innermost closed contours in the binary image. This is accomplished by an algorithm which uses an iterative procedure to number every region according to the number of closed contours containing its boundaries.

The algorithm starts with filling the binary image starting from the upper-left corner. Then at every step, a new image is constructed by filling the negated output of the previous step. This procedure makes sure that a region is not

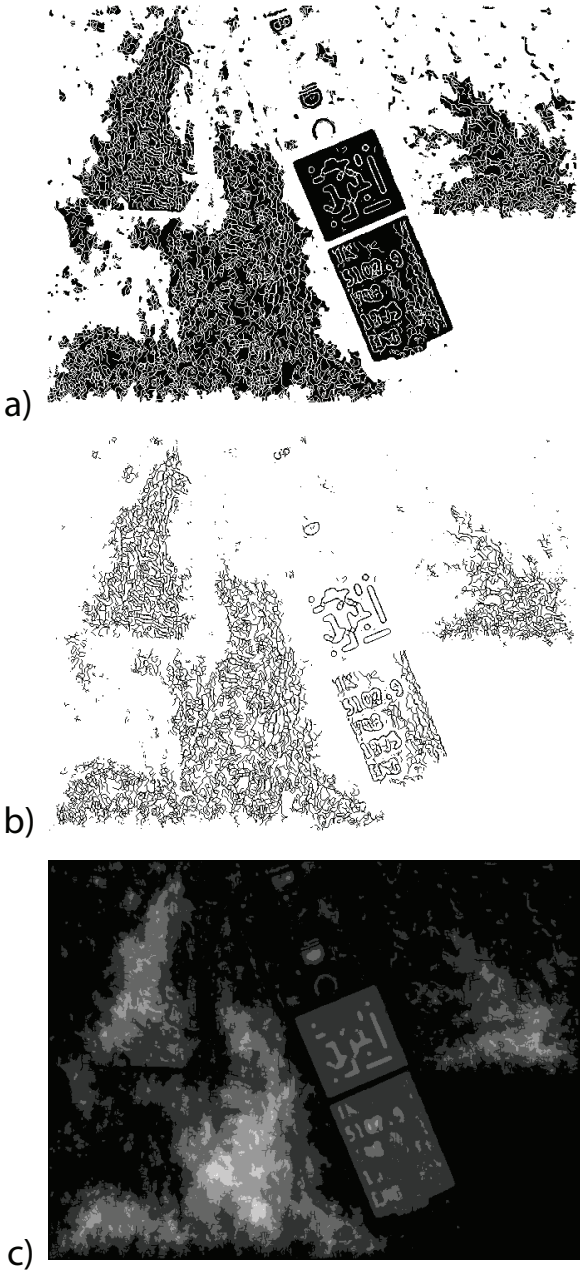

Figure 2. Output images for some selected steps of the algorithm. a) The output of the first fill operation b) The output of the second fill operation. Note that the filled area has reached the inner regions c)The summed image

filled until the algorithm reaches its innermost boundary. The algorithm stops this iterative generation of filled images once the whole image is filled with ones, which means that every innermost closed contour in the binary image has been reached. Now if the resultant filled images from all steps are summed, the value for any region in the summed image will be the number of closed contours containing it. Thus the regional maxima of this image is found, only the innermost contours will be retained. As a result, this algorithm finds the innermost closed contours independent of the number of contours containing it in the binary input image. Examples for the output images for each can be seen in Figure 2.

The output of this procedure is expected to contain the guide bars and corner elements for the code markers. However there will also be other closed contours in this binary image. These are eliminated in processes described in the following sections.

2.4 Determination of Code Marker Candidates

After the binary image is formed, we need to do region labeling in order to find the separate set of points. We expect to have the long and short guide bars and corner elements among the labeled regions. Once the labeling is done, we need to determine the long and short guide bar candidates that can be used to localize a code marker as these bars have a high eccentricity, i.e. the ratio of the major axis length to the minor axis length is large. Namely, for long guide bars this ratio should theoretically be about 7 and about 5 for short guide bars. The ratio is found by exploiting the second-order moments [1]. The second order moments for a region of pixels Z can be expressed as,

$$\mu_{xx} = \frac{1}{|Z|}\sum_{(x,y)\in Z}(x-\bar{x})^2, \quad (1)$$

$$\mu_{yy} = \frac{1}{|Z|}\sum_{(x,y)\in Z}(y-\bar{y})^2, \quad (2)$$

$$\mu_{xy} = \frac{1}{|Z|}\sum_{(x,y)\in Z}(x-\bar{x})(y-\bar{y}), \quad (3)$$

where $\bar{x}$ and $\bar{y}$ are coordinates of the center of gravity, computed as the mean of the point coordinates in the region. The major to minor axis length ratio, ρ, is given by,

$$\rho = \sqrt{\frac{\max(d,f)}{\min(d,f)}}, \quad (4)$$

$$\text{where } d = \frac{\mu_{yy}}{4\mu_{xx}\mu_{yy}-\mu_{xy}^2}, \quad (5)$$

$$f = \frac{\mu_{xx}}{4\mu_{xx}\mu_{yy}-\mu_{xy}^2}. \quad (6)$$

After the eccentricities are found, we can threshold the regions to yield long and short guide bar candidates separately. However, the bars might not be lying along the horizontal or vertical axis. There could be a perspective distortion. All these effects will cause the ratios to deviate from the theoretical values. Considering that this is the first step of candidate elimination, the thresholding is performed within a lenient window, ± 4, around the theoretical values.

Once candidates are found, the number of possible pairings between these two sets (long and short) need to be reduced further using several other geometric properties. For a given long-short guide bar pair, we first look at the ratio of the areas, which has to be around $7/5$. The areas of the regions can be computed by region counting. During the training of the algorithm, it was observed that there can be deviations from the theoretical value when perspective distortion is present. Therefore, a window of values , ± 0.3, around the theoretical value is assumed to be acceptable. This eliminates the long-short guide pairings between regions of essential size difference.

The ratio of the areas does not provide information on how close the possible guide bar pairs are to each other, which is determined when the distance between the centroids is computed. The centroid of each region is the center of gravity. Assuming that the pair we are investigating is the correct long-short guide bar pair, the ratio of the distance between the centroids of long and short guide bar regions to the length of the major axis length of the long guide bar region should be a constant, approximately $6/7$, as shown in Figure 3. Of course, this analysis assumes that the orientation vectors of the long and short guide bars are perpendicular to each other. Again, possible distortions in the image will cause the actual angle between the two to deviate from a perfect 90^o. Therefore, a wider range of values, ± 0.35, for the ratio should be accepted.

Although the angle between the orientations of the long and short guide bar pairs may vary, we do not expect them to be parallel to each other. This observation can be used to eliminate false-positive pairs that have the right eccentricity, area ratios and are close enough in space to pass the centroid test, but actually are nearly parallel to each other. To figure out the angle, we need to determine the individual orientation vectors first. Without a reference point, we cannot uniquely determine the vectors pointing toward the corner elements as there are two possible solutions facing opposite directions for a specific orientation in space which are given by,

$$\mathbf{v} = \mp\begin{pmatrix} -\sin(\alpha) \\ \cos(\alpha) \end{pmatrix}, \quad (7)$$

$$\text{where } \alpha = \frac{1}{2}\arctan\left(\frac{2\mu_{xy}}{\mu_{xx}-\mu_{yy}}\right). \quad (8)$$

To remove the ambiguity in the direction, we find the

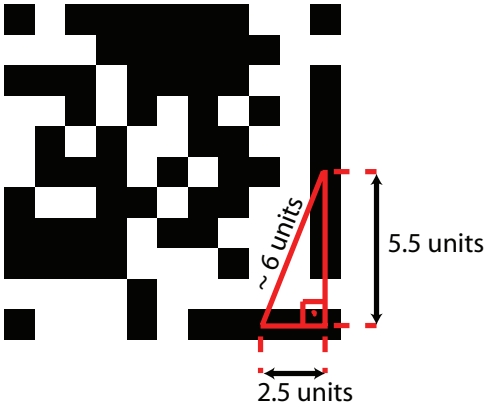

Figure 3. The distance between the centroids of the long and short guide bars in a given code marker is roughly 6 units. Therefore the ratio of this distance to the length of the major axis of the long guide bar is a constant, 6/7.

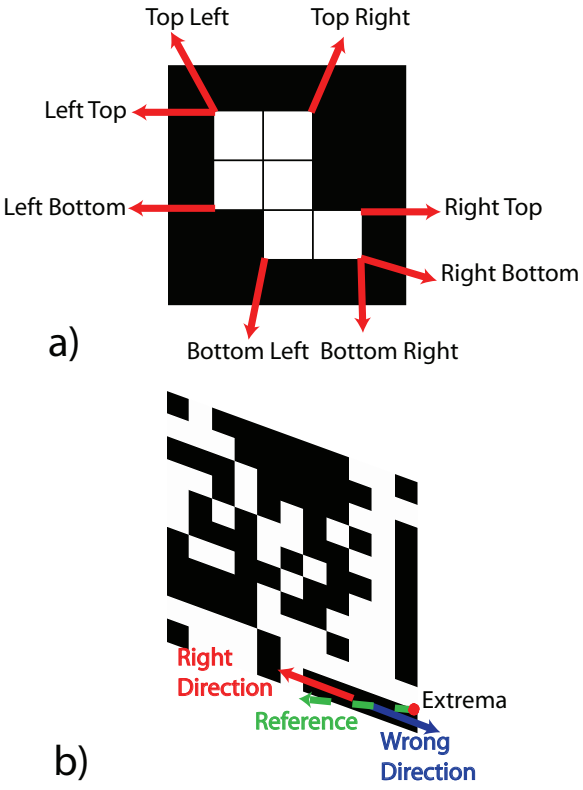

Figure 4. Illustration of the retrieval of the orientation vector. a) There are eight extrema points for any given connected region. b) The right orientation vector (red) makes an acute angle with the reference vector (green), whereas the other possible solution (blue) makes an obtuse angle.

extrema points for both regions and calculate the pair of points, one taken from each region, that yield the minimum Euclidean distance. Then we take the extrema point of the short guide bar that belongs to the pair with minimum Euclidean distance and we form a reference vector by taking the difference between the centroid and the extrema point as displayed in Figure 4. The actual orientation vector for the short guide bar is the one lying at an acute angle with this reference vector. The orientation vector for the long guide bar is found in the exact same manner. Once the unit orientation vectors are found, the cosine of the angle between them should be equal to the dot product of the vectors. By placing a threshold on the values of the dot product, i.e. its absolute value must be within $[0, 0.9]$, we can eliminate the nearly-parallel pairs.

2.5. Localization of Corner Elements

After the determination of possible long-short guide bar pairs using thresholding on area ratios, centroid distances and orientation angles, the next step is the localization of the 3 corner elements corresponding to each -possible- long-short guide bar pair. This is done using the corrected orientation vectors which were derived in the previous section and illustrated in Figure 4. If we denote the corrected orientation vectors for the long and short guide bars as $\mathbf{v_i^l}$ and $\mathbf{v_i^s}$ for the i^{th} pair respectively, then the estimated centroids for the upper right and lower left corner elements are given by :

$$\hat{\mathbf{c}}_i^{ur} = \mathbf{c}_i^l + \frac{5}{7}\mathrm{maj}_i^l \mathbf{v}_i^l \quad (9)$$

$$\hat{\mathbf{c}}_i^{ll} = \mathbf{c}_i^l + \frac{8}{5}\mathrm{maj}_i^s \mathbf{v}_i^s \quad (10)$$

where $\hat{\mathbf{c}}_i^{ur}$ is the estimated centroid of the upper right corner element, $\hat{\mathbf{c}}_i^{ll}$ is the estimated centroid of the lower left corner element and maj_i^l and maj_i^s are the major axis lengths. The real centroids are found by going back to the initial black&white labeled image and looking at a neighborhood of $\pm p_i^{ur}$ and $\pm p_i^{ll}$ pixels around each estimated centroid, where

$$p_i^{ur} = \sqrt{A_i^l/7} \quad (11)$$

$$p_i^{ll} = \sqrt{A_i^s/5} \quad (12)$$

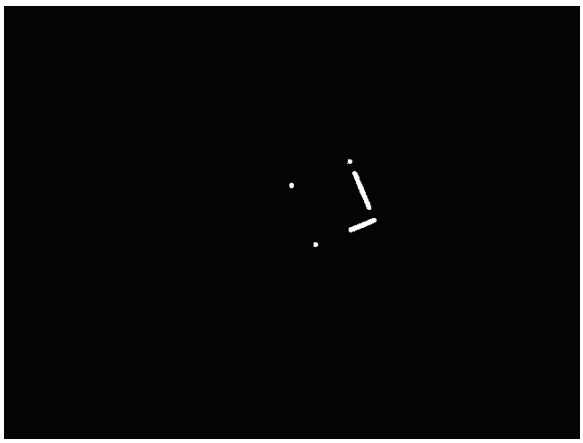

Figure 5. The long guide bar, short guide bar and corner elements corresponding to the markers in the image

and A_i^l and A_i^s denote the areas of the long and short guide bars. The region that has the most number of pixels in this area is taken to be the upper right and lower left corner elements, and their real centroids are obtained from the known black & white labeled image. Using those centroids, the centroid of the upper left corner element is found by :

$$\hat{\mathbf{c}}_i^{ul,1} = \mathbf{c}_i^{ll} + \frac{10}{7}\mathrm{maj}_i^l \mathbf{v}_i^l \quad (13)$$

$$\hat{\mathbf{c}}_i^{ul,2} = \mathbf{c}_i^{ur} + \frac{10}{5}\mathrm{maj}_i^s \mathbf{v}_i^s \quad (14)$$

$$\hat{\mathbf{c}}_i^{ul} = \frac{\hat{\mathbf{c}}_i^{ul,1} + \hat{\mathbf{c}}_i^{ul,2}}{2} \quad (15)$$

Again, we look at a neighborhood of $\pm p_i^{ur}$ pixels to find the real centroid of the upper left corner element.

The resulting three corner elements for each pair are checked for eccentricity and area. Since we expect the corner elements to be close to a circle, the eccentricity threshold is set to 2. The ratios of the areas of each corner element to the area of the long guide bar A_i^l is forced to be lower than $2/7$. Any long - short guide bar pair that has a corner element which has a greater value of eccentricity or area than the specified thresholds is rejected and not considered as a part of a marker. The guide bars and corner elements belonging to the marker in our training image is shown in Figure 5.

2.6. Inverse Projective Transformation

Knowing the label of each of the elements in the image, it is now possible to extract the markers individually from the scene. For this purpose, the extrema points corresponding to the guide bars and the corner elements are taken and the maximum and minimum values of the x and y coordinates of these extrema are used as the corners of the bounding box of that marker. Since the marker is viewed from an arbitrary angle, it is necessary to perform coordinate transformation so that the image has the appropriate viewing angle and scaling for decoding purposes. Regarding the nature of the way the image is taken, this transformation of the marker from the image space to the code space is modeled as a projective transformation. Projective transformation describes the perceived positions of the objects when the point of view changes. In this transformation, straight lines remain straight but parallel lines converge toward vanishing points. The projective transformation is defined as :

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \mathbf{T^{-1}} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (16)$$

Assuming

$$\mathbf{T^{-1}} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \quad (17)$$

$$u = \frac{u'}{w'} \quad (18)$$

$$v = \frac{v'}{w'}, \quad (19)$$

we get

$$u = \frac{Ax + By + C}{Gx + Hy + I} \quad (20)$$

$$v = \frac{Dx + Ey + F}{Gx + Hy + I}, \quad (21)$$

where (u, v) are the coordinates in the code space and (x, y) are the coordinates in the image space. The nine elements of this projective transformation matrix are found by the `cp2tform` function in MATLAB. This function takes 4 input points that contain the x and y coordinates in the image space and 4 base points that correspond to these input points in the code space. The points used for finding the projective transformation and their corresponding code space coordinates are summarized in Table 1.

After the transformation is found, it is applied to the corresponding box extracted from the grayscale image containing the marker. In order to find a better estimate of the individual bits, the resolution of the resulting code-space image is increased by scaling up the code space coordinates fed into the `cp2tform` function. The original and transformed marker images are shown in Figure 6.

Definition	Image Space Coordinate	Code Space Coordinate
Centroid of Upper Left corner element	$\mathbf{c}_i^{ll}$	$(0.5, 0.5)$
Centroid of Upper Right corner element	$\mathbf{c}_i^{ur}$	$(0.5, 10.5)$
Centroid of Short Guidebar	$\mathbf{c}_i^{s}$	$(10.5, 8.5)$
Centroid of Lower Left corner element	$\mathbf{c}_i^{ll}$	$(10.5, 0.5)$
Centroid of Long Guidebar	$\mathbf{c}_i^{l}$	$(5.5, 10.5)$

Table 1. Selected image space and corresponding code space points.

a) b) c)

Figure 6. The results for projective transformation and thresholding. a) The initial marker cropped from the image using extrema points b) Marker image after projective transformation c)Result of thresholding

2.7. Thresholding and Reading the Bits

After getting the high resolution code-space marker, a thresholding is applied to convert this image to binary. For this purpose, the mean value of the gray-scale marker image is used as the threshold (Figure 6). Possessing the knowledge of the coordinates of the center of each of the bits, a $\pm p/3$ pixel neighborhood of each bit is considered, where p is the pixel size of one bit, i.e. it is equal to the scaling factor used to scale up the code space coordinates. In this neighborhood, the majority value (0 or 1) is assigned as the value of the corresponding bit of the marker. The resulting binary image is then converted to a vector and returned as the code to the user, along with the centroids of the upper left corner elements.

3. Conclusion

The algorithm can successfully detect the locations of the upper-left corner elements of the markers in the training set images. Furthermore, the bits in the code marker are read without any error. The processing time for individual images are each under 1 min, with the total processing time for all 12 images being about 95 seconds.

Although the algorithm seems to be working fairly well, there are possible improvements. The algorithm relies on the edge detection routine to find the candidates for long and short guide bars and corner elements. Because we try to find the innermost closed contours using the edge information, the edge boundaries around the bars and the corner elements need to be closed. In cases where there is a discontinuity in the edge, a dilation scheme can be utilized. However, it is hard to determine the morphological operator to be used for dilation as we do not want to arbitrarily increase the thickness of the edges, which in turn will shrink the size of the resulting labeled regions.

Another consideration is the detection of the bits in the code marker. We use a simple thresholding routine to determine black and white pixels. In cases where a vast majority of the 83 pixels are of one type, thresholding by the mean value might yield false readings for some pixels. Nonetheless, tests up-to-date show that the thresholding scheme is quite robust.

The visual code marker detection algorithm can detect all code markers in a camera phone image without any false-positives or repeats, within under 1 minute. Moreover, the code bits in the marker are detected without any error. With the addition of the mentioned improvements, the algorithm shows promise to be a fast and robust detection scheme for visual code markers.

Appendix

Each group member equally participated in the implementation and testing of the algorithm.

References

[1] R. C. Veltkamp and M. Hagedoorn: State of the Art in Shape Matching. Principles of Visual Information Retrieval, Michael S. Lew (Ed.), Series in Advances in Pattern Recognition, Springer, 2001.

[2] M. Rohs:Real-World Interaction with Camera-Phones,Proc. UCS., 2004

[3] Goshtasby, Ardeshir:Piecewise linear mapping functions for image registration, Pattern Recognition, Vol. 19, 1986, pp. 459-466

[4] Goshtasby, Ardeshir:Image registration by local approximation methods, Image and Vision Computing, Vol. 6, 1988, pp. 255-261