Edward Kim

Gabriel Molina

EE368 Final Project

Due: 2 June 2006

Group # 19

Marker Code Detection

Introduction

The detection of marker codes allows users to receive even more content than before from their mobile devices. Using a simple point and shoot interface, a user can receive detailed information about an interesting object in a convenient and concise form.

In this paper, we propose and implement an algorithm based on MAP decoders and region-pairing to detect cell phone marker codes. We have been provided a training set of 12 training images, and will attempt to detect 100% of the bits encoded in each image without error.

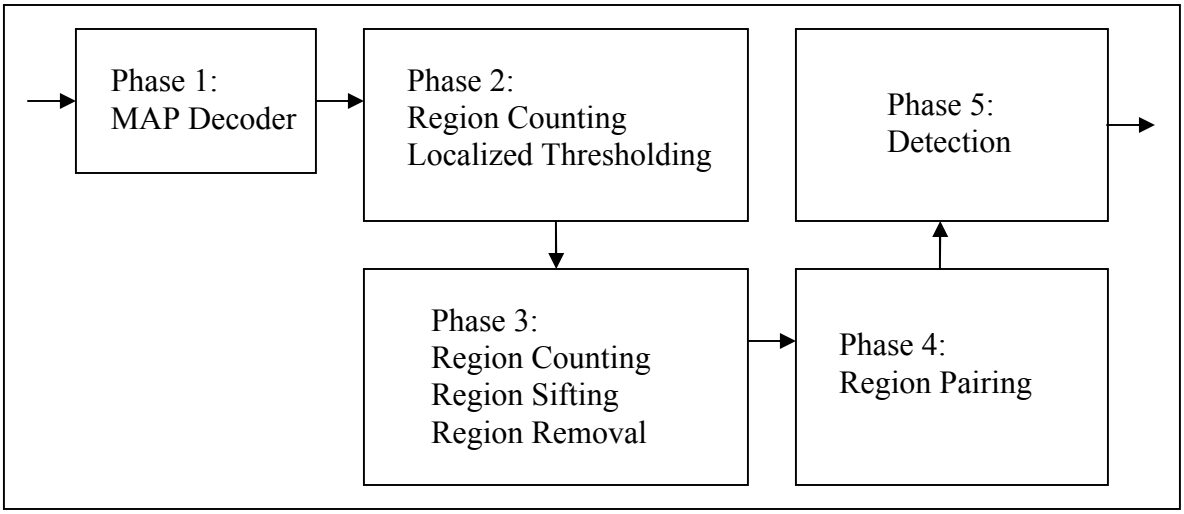


Figure 1.1: Block diagram of detector.

Algorithm

The proposed algorithm operates in several phases. The first 4 phases are preprocessing steps to eliminate unlikely pixels before the algorithm attempts detection. Here is a summary of each phase:

1.1 Map Decoder

The system uses two simple MAP decoders which have been trained using the data from 11 of the 12 training images. One MAP decoder is computed in RGB color space, and the other MAP decoder is computed in grayscale (i.e. HSI intensity space). By using two MAP decoders, we hope to achieve a simple rough mask of the marker pixels which will be refined in later steps.

The process for creating the MAP decoder is the same for both RGB and grayscale. First, masks must be manually created which identify the desired pixels in each training image. Then, conditional probabilities are estimated using the following formulas:

$$p(C_i \mid \text{marker pixel}) = \frac{n_{i\_mar\,\text{ker}}}{N_{mar\,\text{ker}\_pixels}} \tag{1.1}$$

Where $n_{i\_mar\,\text{ker}}$ is the number of pixels $C_i = \{r_i \quad g_i \quad b_i\}$ (or $C_i = \{I_i\}$) found within the marker. Similarly:

$$p(C_i \mid \text{background pixel}) = \frac{n_{i\_background}}{N_{background}} \tag{1.2}$$

Where $n_{i\_background}$ is the number of pixels of color C_i found in the set of background pixels. To implement the MAP decoder, marker pixels are defined as all pixels such that:

$$p(C_i \mid \text{marker pixel}) \geq p(C_i \mid \text{background pixel}) \tag{1.3}$$

In other words, a pixel is categorized as a marker pixel if its frequency within the marker tends to be greater than its frequency in the background region.

1.2 Combination of MAP Decoders

The two MAP decoders both worked to a certain extent, but each had the same problem: oftentimes either decoder would be prone to throwing out too much information. The resolution to this problem was to combine the results of the two decoders. The resulting image consists of pixels which both decoders agree are not marker pixels. Inverting this image creates a conservative but fairly accurate mask of the marker pixels.

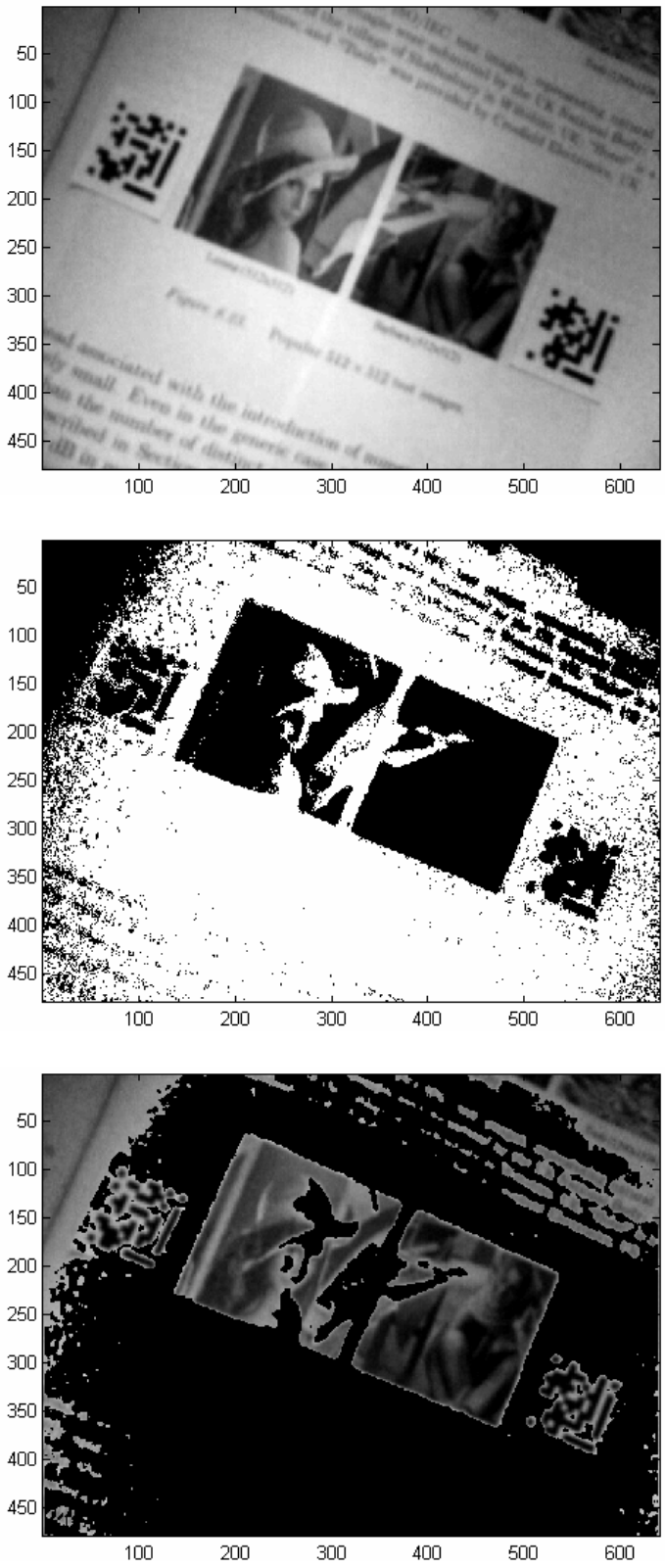


Figure 1.2

(T) Original image

(M) MAP image

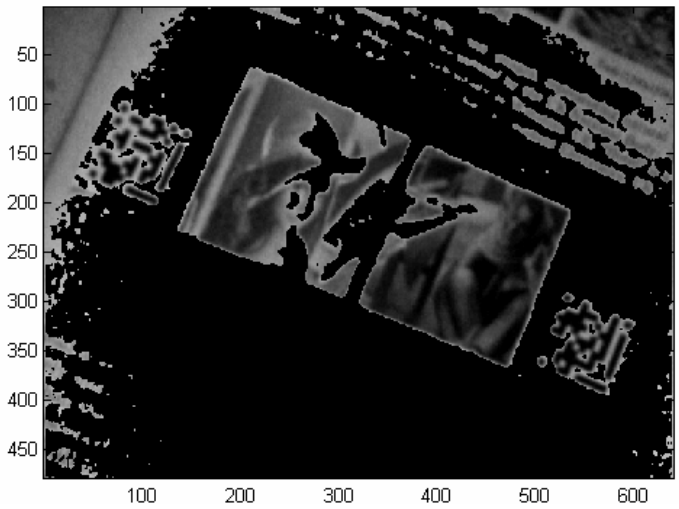
(B) Inv. Mask Result

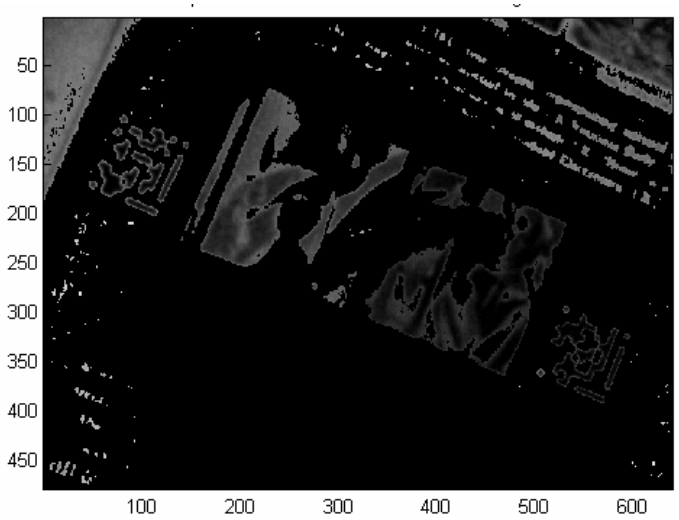
2.1 Region Counting and Localized Thresholding

The next step in the algorithm is to refine the remaining pixels into shaped regions that can subsequently be sifted to identify the locations of the markers. The first step in this process is to remove the 'halo' surrounding each black region in the marker.

First, each contiguous region in the mask is counted and labeled. Then, for each region the mean gray value is calculated. All pixels above this mean gray-value are removed, which has the effect of removing the lightest pixels, which tend to clump around the edges (that is, the 'halo'). Thus, we are left with an image where the marker regions are separate.

Finally, a mask is formed of the resulting 'halo' cleaned image.





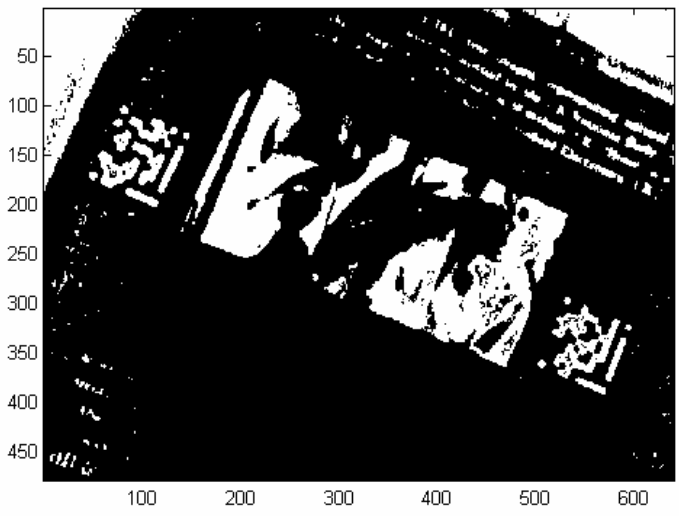


Figure 2.1

(T) ‘Halo’ image

(M) ‘Halo’-clean

(B) ‘Clean’ mask

3.1 Region Counting and Region Sifting

Now that the image is nicely segmented into distinct regions, it is time to detect and remove regions that do not match known characteristics of the marker alignment bars. First, the image labels all regions with a label i and stores the pixel count for each region R_i. With this information the algorithm can begin to discern which regions are most likely the marker alignment bars.

3.2 Sifting Criteria

3.2.1. Size

First, all regions outside a certain size range are rejected. Specifically, very large regions and very small regions are not likely to be the alignment bars. The sifting thresholds are set to eliminate small regions less than 20 pixels in area, and large regions greater than 1000 pixels in area.

3.2.2. Aspect Ratio

The algorithm makes use of the fact that the aspect ratio of the alignment bars is fixed, and is a very robust and selective method of removing unlikely regions.

For each region R_i, the algorithm calculates the centroid:

$$\bar{c} = \frac{1}{N_i}\sum_{1}^{N_i}\begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad \text{where} \quad \left\{\begin{bmatrix} x_i \\ y_i \end{bmatrix} \varepsilon\, R_i\right\} \tag{3.1}$$

Then, the algorithm forms a matrix A of the points $\left\{ z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \bar{c} \right\}$ for the region R_i:

$$A = \begin{bmatrix} z_1 & \dots & z_{N_i} \end{bmatrix}^T \tag{3.2}$$

which is then decomposed using SVD:

$$svd(A) \rightarrow [U, D, V] \tag{3.3}$$

The resulting matrix V gives the two normalized principal axes of the region, and the diagonal matrix D gives the resulting gain coefficients in those directions. The aspect ratio for region R_i is

$$(AR)_i = \frac{D_{11}}{D_{22}} \tag{3.4}$$

Using equation (3.4), every region can be associated with a good approximation of its aspect ratio. Since the aspect ratio of the alignment bars is fixed at 5 and 7, we can remove all regions that do not have an aspect ratio close to this range. The effect is to remove most undesired regions except those with an aspect ratio similar to the alignment bars.

3.3 Region Removal

The image is scanned once more, and all regions that do not fulfill the above criteria are removed from the image. The result is an image with only a few remaining regions, including the alignment bars. Occasionally, the marker's alignment bars will be the only regions remaining.

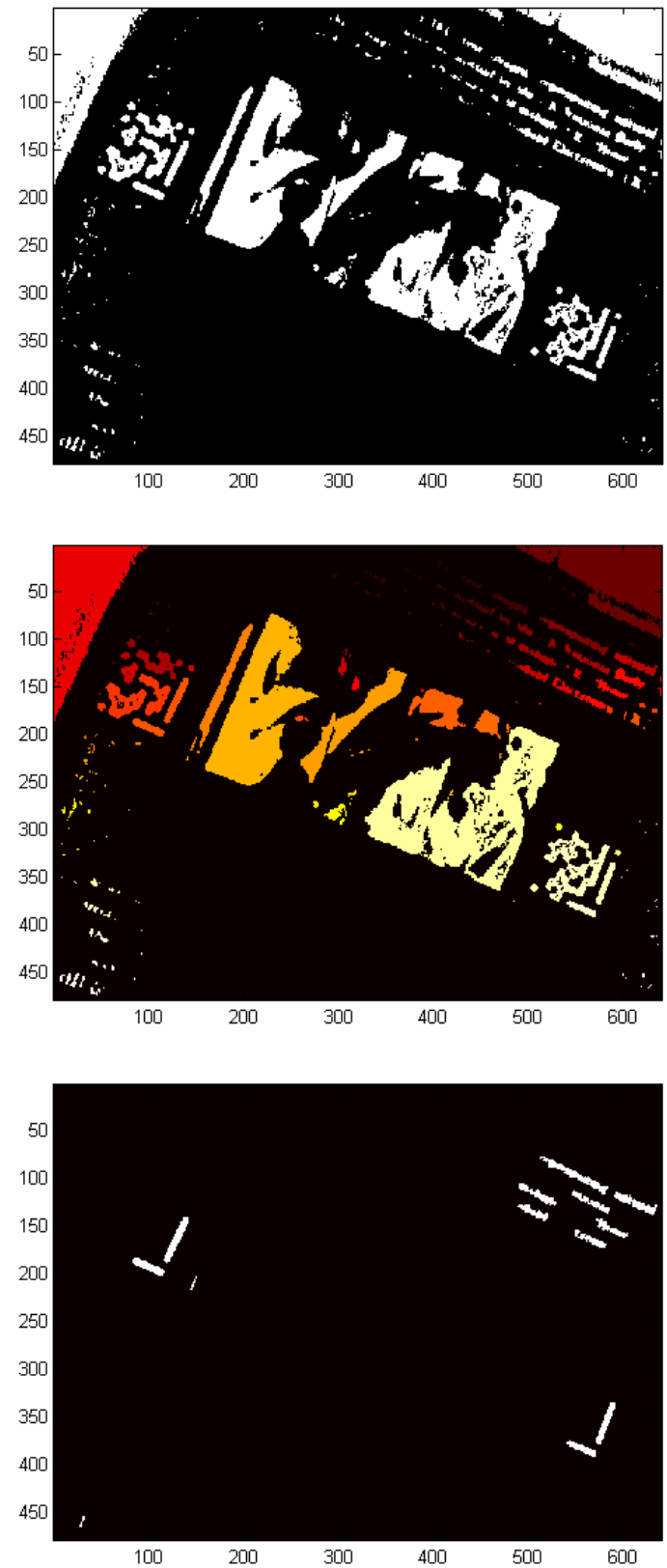


Figure 3.1

(T) Mask image

(M) Coded regions

(B) After removal

4.1 Region Pairing

The remaining regions in the output image have the highest probability of being the alignment bars. The region pairing algorithm uses the geometric ratios of the marker alignment bars in order to detect which other alignment bars might be paired with it.

4.1.1. Geometric Considerations

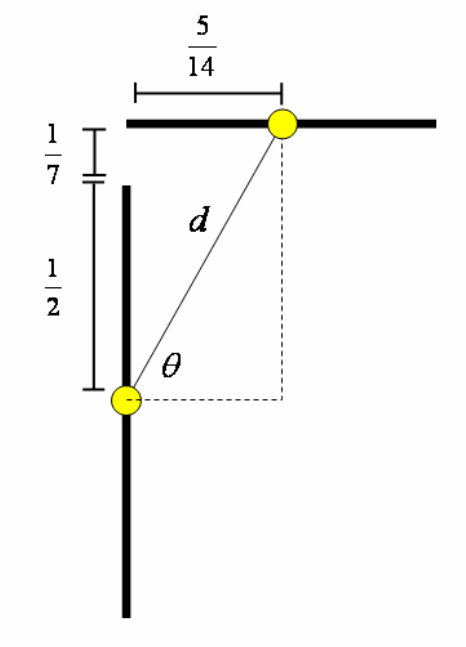


Figure 4.1. Normalized distance/angle diagram between centroid of a 7-region (left) and a 5-region (top).

Knowing the ratios of the distances, we can find the expected angle and distance from the centroid of a 7-region to the centroid of a 5-region. The distance and angle are:

$$d = \sqrt{\left(\frac{9}{14}\right)^2 + \left(\frac{5}{14}\right)^2} \cdot L_{region} = .7354 \cdot L_{region} \tag{4.1}$$

$$\theta = \arcsin\left(\frac{9/14}{.7354}\right) = 60.945 \text{ degrees} \tag{4.2}$$

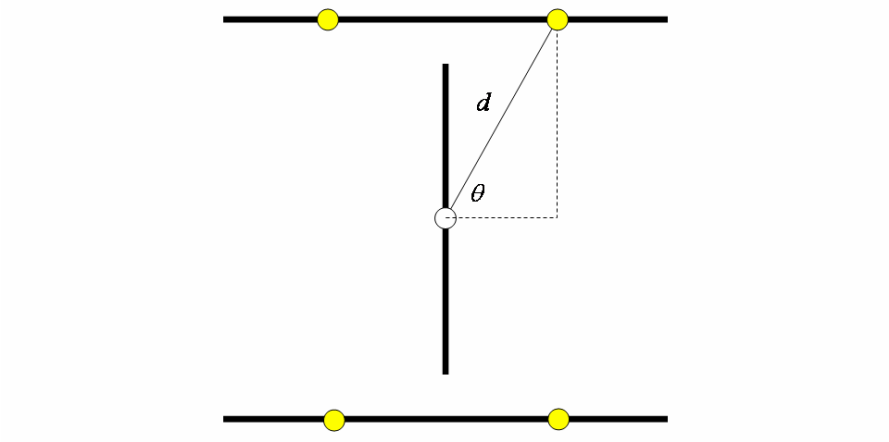


Figure 4.2: Four candidate areas (highlighted in yellow) to check for pairing regions.

4.1.2. Algorithm

The algorithm steps through each region and checks whether the centroid of another region exists in the above candidate areas. If a region is found and the angle between the regions is high enough, it means there is a high probability that both regions form a 7- and 5-region combination. All regions that do not have a paired region can be removed, further simplifying the detection process.

The algorithm has given good results thus far. Many of the remaining garbage regions are removed by this criterion. The algorithm passes pair data to the detection step in the following form:

$$P_i = \begin{bmatrix} lbl_{7-region} \\ lbl_{5-region} \\ \theta_{7-region} \end{bmatrix}, \qquad M_{pairs} = \begin{bmatrix} P_1 & \dots & P_N \end{bmatrix} \tag{4.3}$$

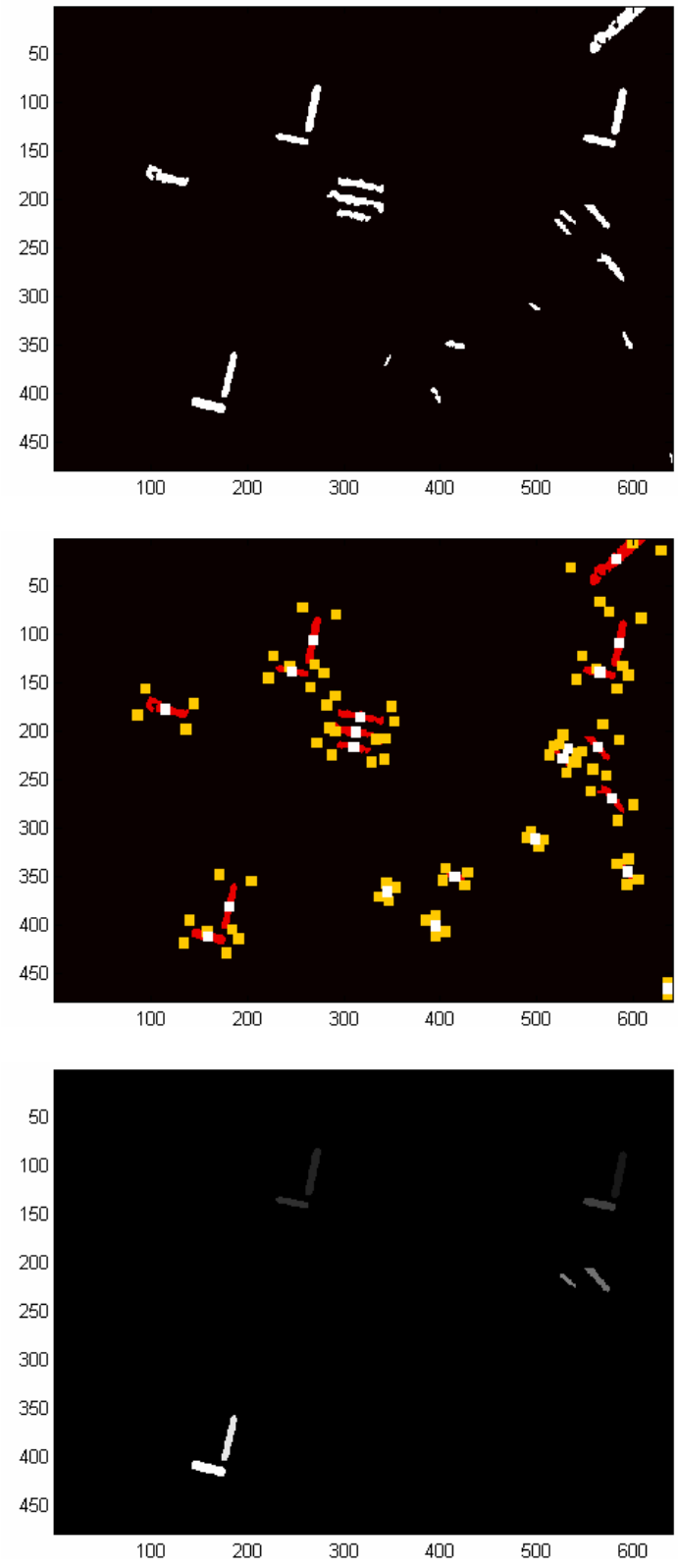


Figure 4.3

(T) Mask image

(M) Centroid detection. Centroids are white squares. Yellow squares are detection areas for a region. White squares overlapping yellow squares means the two regions have been paired.

(B) After removal of unpaired regions (some regions appear gray because image values are coded by region label).

5. 1 Detection

5.1.1. Orientation and Alignment

Using the eigenvectors from the SVD expansion as in Section 3.2.2, the marker's angle of rotation is calculated with respect to the x- or y-axis. The algorithm then rotates the image using the nearest neighbor approximation, which puts the two alignment bars in a vertical and horizontal orientation. However, there is some ambiguity in the orientation because there are four possible orientations of the code marker: 0^o, 90^o, 180^o, and 270^o clockwise rotation, as illustrated by Figure 5.1.

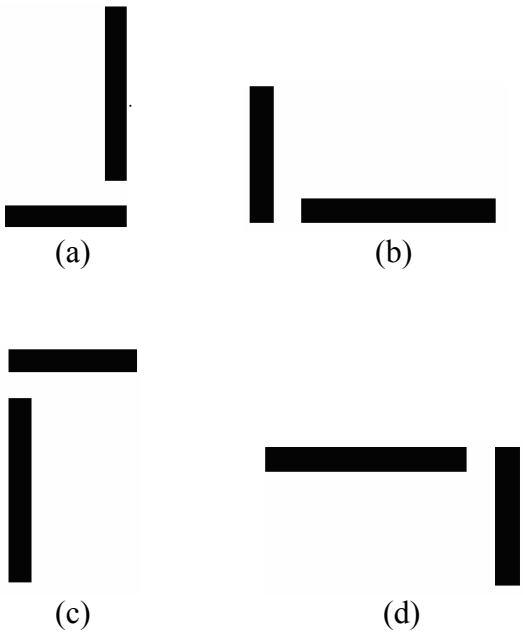


Figure 5.1: 4 possible rotations of marker alignment bars. (a) 0^o (b) 90^o (c) 180^o (d) 270^o clockwise orientation

The algorithm detects which of the four orientations the marker is in by determining the lengths of the horizontal and vertical bars. If the horizontal bar is longer than the vertical bar, then the rotation is either 5.1b or 5.1d. Then, if we detect that the horizontal bar is to the right of the vertical bar, we know the rotation is 5.1b. If we detect that the horizontal bar is to the left of the

vertical bar, the rotation is 5.1d. Similarly, if the vertical bar is longer than the horizontal bar, the rotation is either 5.1a or 5.1c. If the horizontal bar is above the vertical bar, the rotation is 5.1c, and if the horizontal bar is below the vertical bar, the rotation is 5.1a.

The algorithm's next task is to roughly locate the 3 alignment bits at the corners of the visual code marker using the calculated lengths of the alignment bars. Since the two bars are 5- and 7- bits long, the algorithm can determine the number of pixels per bit both vertically and horizontally. This is important because the pixels per bit in the horizontal direction can differ from the pixels per bit in the vertical direction if the camera imaged the marker at a non-perpendicular angle. Therefore, the algorithm must calculate these ratios separately. Knowing the relative bit-length of the 3 alignment bits in relation to the alignment bars, the algorithm can estimate the location of the alignment bits.

The location of the alignment bits is only an estimate used to place an upper and lower bound on the location of the alignment bits. Further measures must be taken to find the exact location of the alignment bit. Using the upper and lower bound on the location of the alignment bit, the algorithm performs a region count of all regions in this bounded area and calculates the centroid of each region. If there are multiple regions in these bounds, only the region with the most extreme centroid is kept. For example, if the algorithm is searching for the bottom-right corner alignment pixel, it keeps the region whose centroid is closest to the bottom-right. In this manner, the algorithm can determine the exact location of each of the 3 alignment bits.

Figure 5.2b shows a mask of the upper and lower bounds of the location of the 3 alignment bits. Figure 5.2d shows the calculated centroids of the 3 alignment bits.

(a)

(b)

(c)

(d)

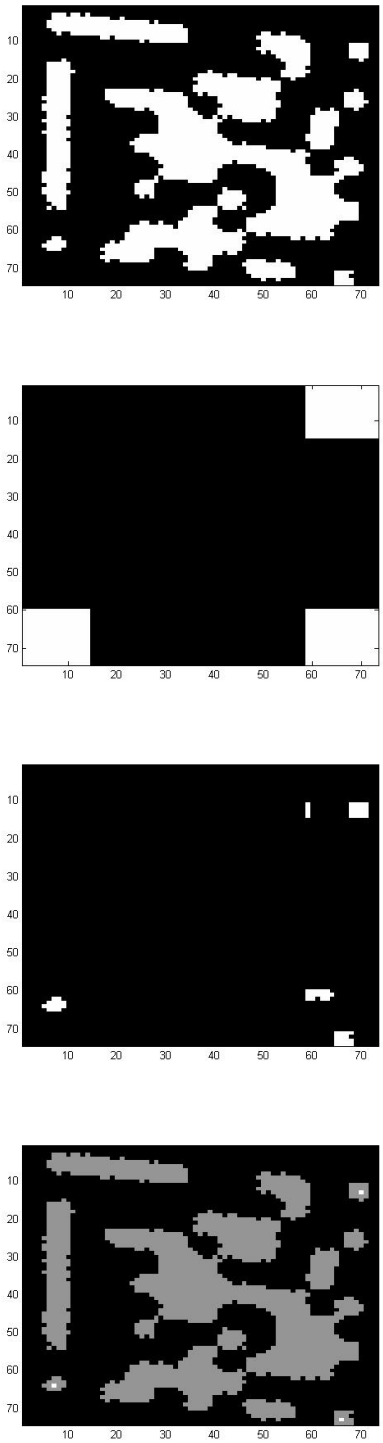


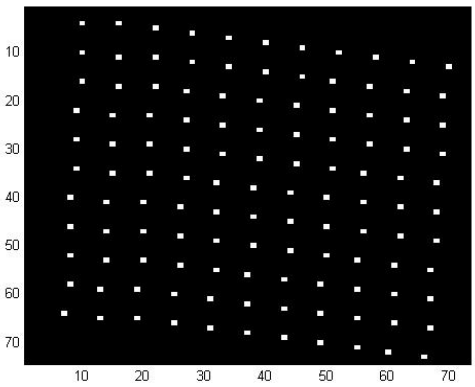
Figure 5.2: Detection of alignment bits. (a) A thresholded image of the code marker. (b) A mask of the upper and lower bounds of the location of the alignment bits. (c) Regions within these bounds. (d) Centoid calculation of the alignment bit. The centroids are denoted in this image by a single white pixel in the middle of the alignment bit

5.2 Code Marker Skew and Size Detection

With the locations of the 3 alignment bits determined, the algorithm calculates a more precise estimate of the marker size and the horizontal and vertical pixel spacing between bits. The horizontal bit spacing is determined by calculating the distance between the left and right alignment bit and dividing by 10, since there are 10 bits between the alignment bits. Similarly the vertical bit spacing is determined by calculating the distance between the left and right alignment bit and dividing by 10.

Depending on the angle that the camera images the marker, the horizontal and perpendicular alignment bars may not necessarily be perpendicular to each other. They can have a slight skew, which can be accounted for using the relative locations of the 3 alignment bits. For example, the lower right alignment bit in Figure 5.2d is lower than the lower left alignment. This distance denotes the vertical skew of the code marker. The lower right alignment bit is also to the left of the upper right alignment bit. This horizontal distance denotes the horizontal skew of the code marker. The algorithm accounts for the size and skew of the code marker and creates a constellation of the center points of each of the bits as shown in Figure 5.3.

(a)



(b)

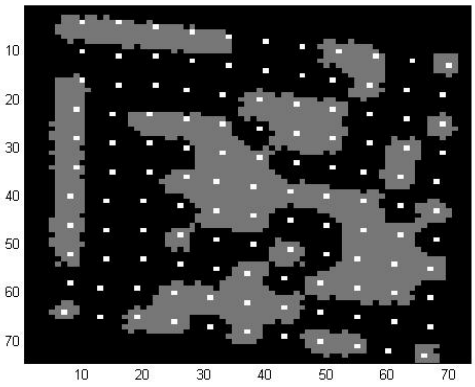


Figure 5.3: Detection of alignment bits. (a) A thresholded image of the code marker. (b) A mask of the upper and lower bounds of the location of the alignment bits. (c) Regions within these bounds. (d) Centroid calculation of the alignment bit. The centroids are denoted in this image by a single white pixel in the middle of the alignment bit

5.3 Bit Extraction

With the location of each of the bits determined, the algorithm performs a check to verify that the image is indeed a code marker. Of the 121 bit locations, 38 have known values. 12 bits contain 1 values and are used for the alignment bars, 3 bits contain 1 values and are used for the alignment bits, and 23 bit contain 0 values and are used for borders around the alignment bars and bits. To ensure that the image is a code marker, the algorithm checks the calculated locations of the alignment bars and bits (15 bits total), and ensures that the bits for these regions are set. Then, the algorithm extracts the bit values of the 83 information bits.

6. Results

In the end, our algorithm was hampered by poor performance in the detection stage. The detection process was not very robust when applied to the varied shapes and sizes of the image markers, and results were inconsistent at best.

One issue that we found particularly difficult was handling the perspective warping that occurs when a photo is taken at an off-camera angle. We were able to address this issue using a quadrilateral-to-quadrilateral mapping function, which allowed unwarping of the marker image. However, this occurred at a late phase in development and we did not have much time to consolidate our detection code and make it robust. In any case, we will present some results for our image processing algorithm.

6.1.1. Pre-Processing Results

Training Image	Markers Present in Image	Correct Pairs Passed to Detection	Total Pairs Passed to Detection
1	1	1	1
2	2	2	3
3	3	3	3
4	1	1	6
5	3	3	5
6	1	1	1
7	2	2	2
8	1	1	1
9	3	3	4
10	3	3	4
11	1	1	1
12	2	2	2

The pre-processing Phases 1-4 passed a total of 33 region pairs to the detector. Of those 33 region pairs, 23 pairs were alignment bars. This means that the pre-processing achieves about 70% accuracy in passing region pairs to the detector. Also, an important statistic is that all marker alignment pairs in the image were passed to the detection phase—that is, no markers were dropped during pre-processing for all of the training images. Thus, we were very satisfied with our pre-processing results.

6.1.2. Detection Results

Training Image	Origins Detected	Bits Correct	
1	1/1	83/83	
2	2/2	156/166	
3	3/3	246/249	
4	err	0	
5	3/3	247/249	
6	0/1	41/83	
7	1/2	141/166	
8	1/1	74/83	
9	3/3	188/249	
10	1/3	78/249	
11	1/1	83/83	
12	0/1	36/83	

Thus, as is apparent from the above table, while our ability to pre-process and sort the regions is very good, our ability to decode the region markers is unreliable and could use more refinement. Most of our detection problems were due to warping distortion in the image, which made it difficult to correctly identify the corner points. In any case, our detector still managed to detected many of the bits properly, and did a decent job of detecting the marker origin.

Appendix—Activity Log

We both spent around 60-70 hours on this project, particularly at the end, as we were having great difficulty with the detection phase of our code. The break down of work is as follows:

Edward Kim:

- Created first skeleton of image processing system (12 hours)
- Worked thoroughly to author, test, and debug all aspects of detection phase (50-60 hours, >1000 lines of code)
- Writing Paper (1-2 hours)

Gabriel Molina:

- Spent majority of time developing pre-processing (Phase I-IV) and making it robust (55-65 hours)
- Broke and re-conceptualized Phase I-IV numerous times
- Helped develop methods for overcoming detection stage difficulties (10-12 hours)
- Reading Papers/Ideas Research (2-3 hours)
- Writing Paper (2-3 hours)