Visual Code Marker Detection

EE368 Spring 2006

Kevin Gabayan, Itai Katz, Kiran Madhav

June 2, 2006

Contents

1 Overview

We propose a method of detecting two dimensional code markers in a digital image. First, an adaptive threshold is applied to the original image to create a binary image separating light and dark regions. Next, we take advantage of the code marker characteristics by searching for dark regions which are potentially the long guide bar in a code marker. Using statistics about the long guide bar and geometry of the short guide bar and corners, we search for corner points to define the marker region. By weighting many factors an error metric is calculated. Using this error metric and template matching, we either accept or reject the configuration as a potential code marker.

Once we have identified the four corners that define the code marker region, we must determine the data bits in the code marker. We apply a projection to create a flat image of that region which is perspective invariant. By examining the second derivative across each pixel row and each column of the code marker, we can statistically determine the demarcation between the units, apply a Gaussian filter to each unit and threshold to determine white or black units. Lastly, the digital bits are extracted and the marker is rotated to obtain the correct orientation.

2 Algorithm Design Implemented

2.1 Adaptive Thresholding

The first algorithm through which we process an image is a thresholding function to separate dark and light regions producing a binary image. The purpose of this binary image is to identify features in the black code markers. Because images vary in illumination, we do not use a global threshold here. Instead, a local threshold is selected for each pixel based on the range of intensities in its local neighborhood. Searching the local neighborhood allows for foreground and background intensities to be weighted into the local threshold. We implement this by convolving the image with a Gaussian low pass filter, then subtracting this image from the original and thresholding based on this difference image[**?**]. Since we use this binary image to identify long guide bars only, we further impose the requirement that the centroid of the each identified region must fall within that region. This masks out L-shaped or curved regions which may pass the eccentricity and perimeter to area ratio tests, but which are clearly not in the shape of a guidebar.

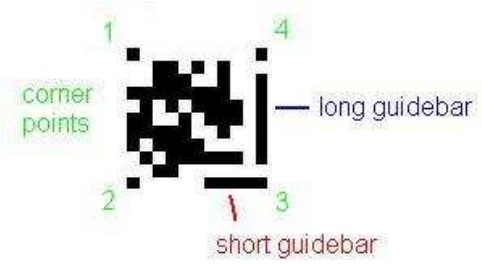

Figure 1: Code marker features used for identification. Corner points labeled in green.

2.2 Identification of Long Guide Bar Candidates

White regions in the binary image are then chosen as potential long bars, or as we call them, candidates. These regions are chosen based on their eccentricity. Ideally, the long guide bar has a ratio of 7 to 1 (seven units long by one unit wide), i.e., the ratio of the major axis to the minor axis of the region is seven. Due to skewed ratios because of perspective, we choose all candidates which have an axis ratio between 3 and 12. In addition, we eliminate very small regions as well as regions whose perimeter to area ratio is very high. Small regions are considered noise and a high perimeter to area ratio is usually an indication of text in the digital image, or some other non guide bar shape.

2.3 Corner and Short Guide Bar Location Prediction

After long guide bar candidates are identified, we use knowledge of the given code marker template, namely the three corner points and short guidebar. These code marker features are illustrated in Figure **??**. Based on the dimensions of the regions identified as the long guidebar, we calculate predicted locations of the centroids of the three corner points, as well as the short guide bar.

If we continue along the direction of the orientation (the angle of the major axis of the region) of the long guide bar, we can identify Corner 4 by tracing the line through the major axis in one direction and the short guide bar by tracing the line in the opposite direction. Based on the length of the long guide bar, the locations of the centroid of Corner 3 (which lies in the short guide bar) and Corner 4 can be predicted. Next, we predict the location of Corners 1 and 2. We know the axis along which the centroid should lie based on the major axis of the long and short guide bars, and we can compute the distance based on the length of the long guide bar. We store the centroids of the four predicted corner points

(a) (b)

Figure 2: Key Features Marked on (a) digital marker and (b) extracted from an image

and short guide bar. See Figure **??** for marked examples of the ideal code marker and the adaptively thresholded image.

2.4 Identification of Nearest Regions Based on Prediction

Once the predicted centroids are generated for key features in the code marker, a search is performed within a windowed area of that predicted centroid to find the nearest white region which would be the corner or short guide bar. We impose various constraints on identifying these white regions based on known characteristics of the code marker.

For the short guidebar selection, we determine the potential regions orientation based on the major axis of the region. This is compared to the orientation of the major axis of the long guide bar region. For a correct code marker, the angle between them should be close to 90 degrees. Between the two regions identified, the corner and short guide bar are distinguished based on eccentricity. The more circular shaped region is identified as the corner, and the more elongated region as the short guidebar. For choosing the nearest region to a predicted corner, we impose the eccentricity constraint that the chosen region corresponding to the corners must be circular and their area must be less than the area of the long guide bar.

2.5 Error Metric for Potential Code Markers

For each region in the image which is a potential code marker for which predicted corners and corresponding regions have been identified, an error metric is computed. Thus, when the algorithm is run across the entire image and all potential code markers have been identified, we can chose the true code markers using this error metric. We choose the three potential markers with the smallest error. The false positives are then rejected in the orientation function.

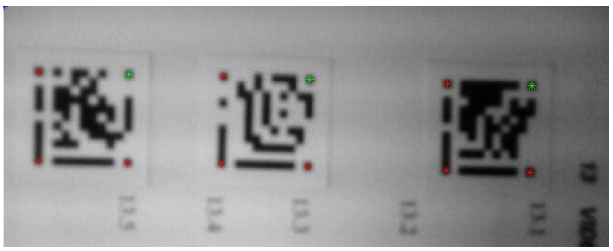

Figure 3: Identified corners in training image

The error metric is computed based on the error of the three corner points and the centroids of the long and short guide bars. After finding the nearest regions that satisfy this criteria for potential corners, we measure the distance between the centroid of each potential and predicted corner. This distance is then weighted by the eccentricity of the each corner region. This puts more error weight on the more elongated regions. The orientation of the long guide bar to the short guide bar should be 90 degrees, and the proximity of the candidate region to this orientation is also factored into the error metric. See Equation **??** for this computation for the ith potential code marker region, where Θ_{LG} is the angle of the major axis of the long guide bar and Θ_{SG} is the angle of the major axis of the short guidebar. Equation **??** describes the calculation of the error weight for each corner, where k is the corner number, ϵ_k is the eccentricity of the kth corner.

$$e_{angle}(i) = \left| \frac{\pi}{2} - |\Theta_{LG} - \Theta_{SG}| \right| \tag{1}$$

$$e_{tot}(i) = e_{angle}(i) \sum_{k=1}^{3} \epsilon_k \left((x_{p_k} - x_{c_k})^2 + (y_{p_k} - y_{c_k})^2 \right) \tag{2}$$

This error metric is computed for all the regions originally identified as long guide bars. We then look at all the error values and threshold for up to three configurations which are code markers. The error value for the true code markers is at least an order of magnitude less than the misidentified code marker regions. We use an adaptive threshold based on the different between the errors to determine the number of true code markers. Figure **??** is an example of correctly identified corners in an image.

2.6 Perspective Projection and Corner Stretching

At this point, the four corners of the code markers have been identified. In order to extract the bits in the code

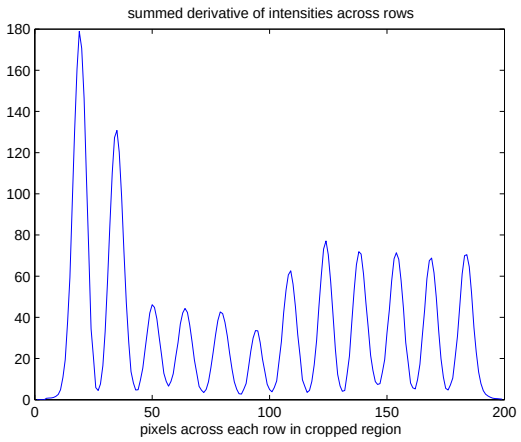

Figure 4: The peaks in this signal identify where code marker transitions lie, and where to place the grid line demarcations.

marker, a perspective invariant image is required. This is performed using the MATLAB image transformation function.

Successful data extraction after the image transformation requires that the transformation corner points bound the entire marker image. Stretching these corners outward from the marker center yields our desired transformation corners. For each corner point and its opposite corner, the line equation and distance is calculated. Solving the line equation for coordinates that are the desired distance from the opposite corner point yields two solutions. The solution nearest to the original corner point is the desired stretched coordinate.

2.7 Grid Placement

Once a square image of black and white regions is obtained, the image must be segmented into an 11 x 11 grid to identify the digital marker. Due to perspective projection, lighting conditions, noise and various other factors, the width of each unit across a code marker is not necessarily uniform. As such, a statistical method is implemented to determine the demarcations between elements in the code marker.

We examine the intensities across each row of pixels to identify where the transitions between black and white fall. Looking across all the rows in the code marker region, these transitions will statistically identify where the grid lines should be placed. Rather than finding the transitions based on only the intensities, we perform some computations on the signal to improve accuracy. Intensities across each row are summed, differentiated, and smoothed with a low pass filter. This produces a signal such as shown in Figure **??**. The identical transformation is applied to the elements in each column, and the grid lines for each row is determined. The positions of the peaks in this image correspond to the grid line demarcations. In a correct code marker, there will be 11 peaks in both row and column dimensions. See Figure **??** for an illustration of the resulting grid lines placed on top of the cropped code marker region.

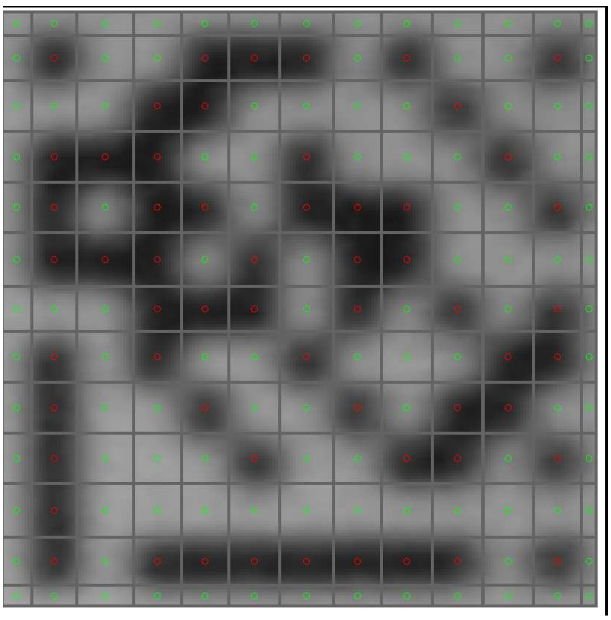

Figure 5: Cropped region with grid lines overlaid. Red Mark → Black Unit and Green Mark → White Unit

2.8 Extracting Data Elements

After the grid lines are placed and the units demarcated, the area within each unit must be classified as black or white. The first step involves taking the difference between the pixel values and the average intensity of the entire image. A Gaussian window is applied to the resulting pixels in each unit, which weights the center pixel in the unit more than the rest of the pixels in the unit. This computation is represented mathematically in Equation **??**. Note that ρ is the average intensity of the image, $p_k(x, y)$ is the cropped region of the image which corresponds to the kth code marker, $w(x, y)$ is the Gaussian window which is centered at the center pixel within $p_k(x, y)$, and $bw_k(i)$ is the ith bit weight being computed for the kth code marker.

$$bw_k(i) = w(x, y)(p_k(x, y) - \rho) \quad (3)$$

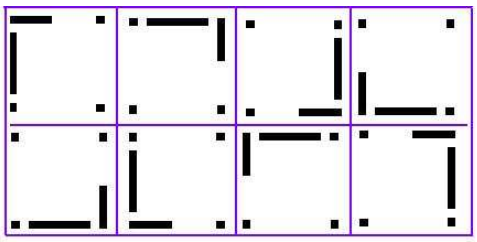

Figure 6: Eight different orientations of the code marker.

After the weights of each pixel in $p_k(x, y)$ are calculated, a heuristically determined threshold is applied to distinguish dark from light regions to classify each bit i in each region k. In some images, due to non ideal stretched corners given to the image transform, a white area surrounding the code marker is in the cropped region. This border of white units is in the image in Figure **??**. In such cases, we remove those borders which are classified as all white.

2.9 Orientation Correction

After the 11x11 bit data has been extracted, the fiducial guide bars uniquely determine the necessary flips and rotations required to obtain the desired data orientation. The rectified marker image can appear in one of eight orientations: four rotations of the original marker, and four rotations of the mirrored marker as shown in Figure **??**.

If the black pixel count in a corner's 2x2 matrix is 2, the corner is known to contain the short guide bar. The orientation of the short guide bar in this corner is determined from this corner's 2x2 matrix by template matching. The short guide bar orientation then uniquely determines the number of rotations and flips necessary for the desired orientation. Further template matching to the fixed bits of the 11x11 marker matrix verifies if the binary data could be a marker matrix. Failure at any one of these steps reports that the marker contains invalid data.

2.10 Repeated Code Marker Removal

After the final valid code markers are returned, and just before the `data_detected` array is populated, a check for repeated code marker identification is performed. If the centroid of a valid code marker is within 20 pixels of the centroid of another detected marker, then we reject that as a repeat. This problem arose in testing if one of the four corners was misidentified, but close enough to the true corner that projection is correctly performed, and valid data is extracted.

3 Other Attempted Algorithms

3.1 Feature Density Based Sub-Region Selection

It can be computationally intensive to search across the entire image for code markers. One improvement is to limit the search region by identifying subregions that are sure to contain all potential code markers. Some feature characteristic of the code marker regions can be identified to extract these subregions.

The marker's Harris corners were found to be a distinguishing feature that could be used to select regions to check for marker candidates. The Harris corner detection algorithm we implemented is described online. [**?**] Corner detection was highly sensitive to changes in illumination, but succeeded when the corner score threshold was selected as a linear function of the image's intensity variance. A density map of the detected corners was generated by convolving a binary image containing the corner points with a 2D Gaussian kernel. On the assumption that our markers generate the highest density of corners in the image, the brightest spots in the density map are selected by an adaptive threshold. The markers are expected to generate a roughly uniform but high density of corners that appears as a flat region in the density map intensity histogram. The highest intensity for which the second derivative falls under a threshold indicates this flat region and is used to threshold the density map. The remaining blobs indicate the presence of a marker.

Guidebar search is then restricted to a window centered at each blob centroid. While this method succeeded on our test images, guidebar searching more accurately rejected false candidates and was fast enough to be performed on the entire image. Candidate region estimation was no longer necessary.

3.2 Corner Based Region Selection

Harris corner detection was also useful for identifying the four corners of the code marker. A convex hull was applied over a region of corners, and polygon produced was roughly square shaped with corners based on the outer corners of the code marker. This could identify the four corners upon which to apply the perspective projection and grid placement. Depending on the illumination of the image and the parameters in the corner detection algorithm, the outermost corners from the image may not be detected. As such, this method is not reliable on its own as it does not take into account many features in the image. See Figure **??** for an example of code marker selection based on corner detection. One corner unit of the image is not correctly identified because Har-

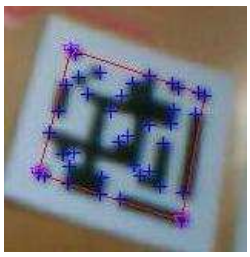

Figure 7: Purple triangles indicate code marker corners. Note that 3 out of 4 are identified. All corners marked in blue x's.

ris corner detection did not pick up the outermost corner corresponding unit.

3.3 High Chrominance Region Removal

Another metric to eliminate entire regions is by considering the intensity of color at each pixel. This was implemented by thresholding away pixels whose red, green or blue component deviated from the mean of red, green or blue by a significant amount. This removed some regions, such as the red squares in training image 5, but was not significantly helpful elsewhere. Because of the multiple comparisons performed on a pixel by pixel basis, this algorithm was extremely slow. The execution time was about thirty times longer, i.e., thirty seconds, than other functions in the code at the time of implementation. The benefit of high chrominance region removal was not worth the cost in computation time.

4 Conclusion

For this image detection problem, we found that the best solution was to generate many metrics to determine candidates for the marker and combine these metric in a weighted error function. We identified regions of potential code markers by examining a binary image and searching for the regions corresponding to fixed features such as guide bars and corner points.

5 Appendix

5.1 Work Distribution

See Table **??** for detailed task breakdown.

Table 1: Work Distribution

Tasks	**Kiran**	**Itai**	**Kevin**
Color Based Segmentation			
Chrominance region removal	x	x	
HSV region removal			x
Corner Detection			
Harris algorithm	x		x
Color based corner rejection	x		
Density distribution			x
Subregion selection			x
Convex hull corner selection	x		
Binarization			
2-D Adaptive Thresholding	x	x	x
1-D Scan Based Binarization		x	x
Guide Bar Candidate Selection			
candidate rejection	x	x	x
corner prediction		x	
prediction robustness	x	x	x
error metrics	x	x	x
long guide bar candidate selection		x	
nearest region selection	x	x	
corner expansion			x
Perspective Projection			
implementation		x	
sub region cropping		x	
Grid			
line determination concept	x	x	
implementation		x	
Data Extraction			
thresholding units		x	
orientation selection			x
rotation/orientation robustness	x	x	x
final candidate selection	x	x	x
Miscellaneous			
MATLAB error handling	x	x	x
improving robustness	x	x	x
repeated region removal	x		x
Project Report			
text and content	x		x
graphics and images	x		
editor	x		x
work distribution table	x		x

References

[1] Wellner, Pierre D., “Adaptive Thresholding for DigitalDesk.” Technical Report EPC-1993-110, Rank Xerox Research Centre, 1993. 17 pages.

[2] R. Fisher, S. Perkins, A. Walker and E. Wolfart.

Adaptive Thresholding [cited May 2006],

Available `http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm`

[3] Rohs, Michael. ”Visual CodeWidgets for Marker-Based Interaction” WSAWC’05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, Columbus, Ohio, 2005.

[cited May 2006],

Available `http://www.vs.inf.ethz.ch/res/proj/visualcodes/`

[4] Peter Kovesi, MATLAB Harris Corner Detection

[cited May 2006],

Available `http://www.csse.uwa.edu.au/~pk/research/matlabfns/`