Detection of Visual Code Markers in Camera-Phone Images

Prabhu Balasubramanian
Stanford University
Department of Electrical Engineering
prabhub@stanford.edu

Allen Huang
Stanford University
Department of Electrical Engineering
ahuang06@stanford.edu

Abstract

We describe the algorithm design and implementation of our submission for the EE368 final project, which involves detection of visual code markers in camera-phone images. We describe our approach and results, as well as the challenges faced during the development of our algorithm. For the reader's interest, we also include a brief account of the algorithmic possibilities which we explored but did not ultimately employ.

1. Introduction

In this project, we were charged with the task of locating and reading 'visual code markers' in cameraphone images. These markers are 11x11 arrays in which each element is either black or white. Some of these elements are part of distinguishing features which identify the pattern as a valid marker; these features include fixed guide bars and fixed corner elements. 83 of the 121 elements in the array represent binary-coded data (where a black element is a 1 and a white element is a 0).

Additional information regarding the specifications and parameters of the project can be obtained at the EE368 course website,
http://www.stanford.edu/class/ee368/project.html.

This report describes the algorithm design and implementation which we used to solve the problem of detection and reading of visual code markers. It also outlines the results we achieved with the training image set and mentions key challenges and thoughts for future exploration if we were to refine this algorithm to make it even more robust.

2. Algorithm

To solve the given problem, we employed a number of standard image processing techniques, all implemented in MATLAB, making heavy use of the capabilities of MATLAB's Image Processing Toolbox, in addition to the software's standard matrix processing functionality. On the whole, our algorithm is strongly biased towards the use of morphological techniques, such as dilation, erosion, and region labeling, although the final reading of bits happens by processing the original (non-binarized) image.

Furthermore, the algorithm we used is best described as a hybrid method which employs both *top-down* and *bottom-up* approaches. In particular, our processing of the input image begins by taking a macroscopic view of the image in which we eliminate large segments of the image which we know cannot be related to visual code markers. After a sufficient number of image regions have been eliminated from candidacy by this step, we approach the image from the other direction, characterizing the small features and looking for regions which could be the guide bars or corner elements of markers. Confirming the nature of these small regions and their spatial relationships with each other allows us to conclusively identify markers and proceed with reading them.

2.1. Image preparation

Our first step is to convert the input image to grayscale and employ morphological techniques for edge detection. In the first place, this step reduces the image to its essential features. Moreover, because the visual code markers should contain particularly strong edges (since transitions are, roughly speaking, from black to white), edge detection tends to highlight the regions containing markers (among other candidate regions). Edge detection takes place using the gray-level morphological erosion of the grayscale image. By using the erosion, we prevent nearby regions from merging, which would lead to shapes that could not be processed. This allows us to then binarize the image by thresholding. We thereby eliminate regions which are "not edgy enough" – all pixels which do not meet or exceed a certain brightness level (with respect to the overall mean of

the image) are zeroed out. This thresholding level is taken to be a standard deviation about the mean in the image. By adapting the threshold level to each image, we can compensate for differences in the original illumination of the photograph. We subsequently perform morphological operations on this newly binarized image; opening the image eliminates small 1-regions, and closing it solidifies the remaining regions.

2.2. Region classification

Having eliminated all but relatively large contiguous regions from our binarized image, we then proceed to determine which of these regions could be parts of visual code markers. In particular, we use region labeling to distinguish among all the regions in the image and then iterate through them, noting the geometric properties of each region. After zeroing out regions which are either too big or not solid enough to be marker elements, we group the remaining regions according to their major-to-minor-axis ratios; some regions are candidate square elements at the corners of the markers, and some regions are candidate rectangular elements to serve as horizontal or vertical guide bars.

2.3. Region evaluation

We now test the regions which are candidates to be guide bars in potential markers. Our approach is to look at each region and see if it has a corresponding short bar region near it such that it could be a vertical guide bar; this is achieved by determining the orientation (along the major axis) of each bar and looking 'above' and 'below' it for another bar that is approximately perpendicular (allowing for variation in this angle due to perspective skew).

If we have found two bars in the correct configuration, we use the width of these bars and their perpendicular distance from each other to calculate an element size for the potential marker that would be framed by these guide bars. We then use this estimated element size to calculate an expected location for the upper right fixed corner element of the marker. Verifying the presence of an approximately square region at this location adds further robustness to the marker identification scheme.

2.4. Locating the top left element

Having identified the two guide bars and one of the three fixed corner elements, the algorithm now goes about locating the top left fixed corner element and determining its position (since this is one of the outputs expected from the algorithm). This is a more difficult task than it sounds because of the perspective skew that is inevitably introduced into source images of this type. Our approach is to use the "side length" of the square marker (calculated using the distance between the horizontal guide bar and the upper right corner element) and an approximate skew factor (calculated based on the angle between the two guide bars) to find a candidate for the position of the top left element. Then, we look for the nearest regions to this point (within a relatively small radius) and choose the one which is the leftmost and topmost of the set; it is not adequate to simply choose the closest region, because the skew could cause misidentification of elements. In order to improve performance, the guide bars are searched in order of size, and when three markers have been found, the algorithm will return the result for that image. Finding the centroid of the chosen region completes the first phase of the algorithm – locating markers by their top left elements.

2.5. Reading bits

Reading the bits in an individual marker was a much more difficult task than first imagined. We soon realized that due to perspective skews, the markers were not only non-rectangular, they were not parallelograms. In order to help compensate for the skew, we use an algorithm similar to that which we used to find the upper-left corner to find the upper-right and left-bottom corners. The upper-right corner is located by finding the nearest square dark spot in the major axis of the right-hand marker. The upper-left corner is found by moving a distance away based on the width calculated in the previous section.

Using the corner points of the marker, the slanting on each of the four sides can be found, and each bit-step can compensate for any slant by moving in both the x and y directions. The slanting is applied linearly so that in the middle, skew is calculated by taking the average of the slant on either side. By stepping across the matrix, each bit value is found from a 3-by-3 region surrounding the pixel in question.

3. Results with training images

Overall, we found that our algorithm performed excellently when applied to the training image set. It was able to accurately detect and locate every one of the 23 markers in the 12 images, and on most images, it read the bits out with very few errors.

For all twelve of the training images, no false positives or repeated origins were found, and at least 90% of the bits were correctly determined in 20 of the 23 marker images given. In 13 of the images, all 83 bits were correctly read. The most difficult images for our algorithm were image #5 and image #12, which both had strong perspective skews in the image. Logically, the images that were most poorly interpreted were the markers that suffered the most from skew.

Execution time for the images was variable depending on the number of features to examine in each image. Images for which the algorithm quickly found three markers were able to finish as quickly as 7.7 seconds, while an image with only one marker took as long as 28 seconds. All of the images were read within 30 seconds. Our composite score was 1722, over 90% of the 1909 points possible.

Figures 1-3 (see right) illustrate a typical training image (image #4) and the results of our algorithm when given this image as an input.

As mentioned, the greatest challenge faced by the algorithm was presented by training image #5, which contained three markers pasted on an extremely feature-heavy coupon sheet. Although we were able to locate all three markers, we had difficulty reading the bits correctly, largely due to the extremely perspective skew of the image, especially in the center and right code markers, which suffered from the steepest skew angles found anywhere in the training set. Throughout our time working on developing this algorithm, we found ourselves making tradeoffs between performing well on image #5 and sacrificing good performance on other, less challenging images. We attempted to rectify this problem by applying the concept of perspective projection, which was discussed in Professor Girod's EE368 lecture, but we were unable to correctly implement the necessary transformation before the project deadline. Thus, a particularly interesting area of exploration which we would pursue, given additional time, is correction of the perspective skew of the various images, which we believe would dramatically improve the bit-level accuracy of our algorithm (which is already remarkably accurate, given the quality of these images).

4. Other avenues we explored

Before settling on our final algorithmic approach for this project, we spent a great deal of time exploring other possible methodologies. Though we ultimately rejected these, the time spent in understanding them was extremely valuable, so it is worthwhile to describe these methods in outline here.

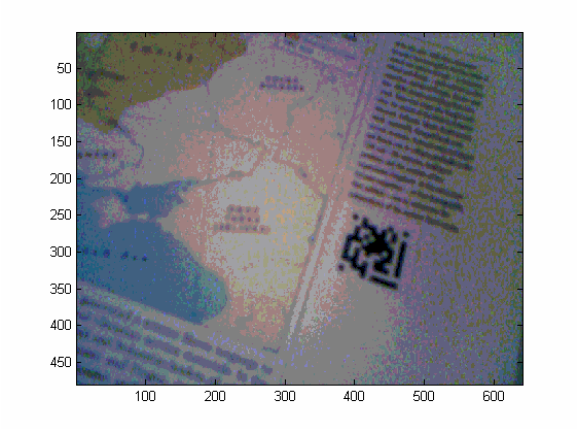

Figure 1. Training image #4, which contains one visual code marker.

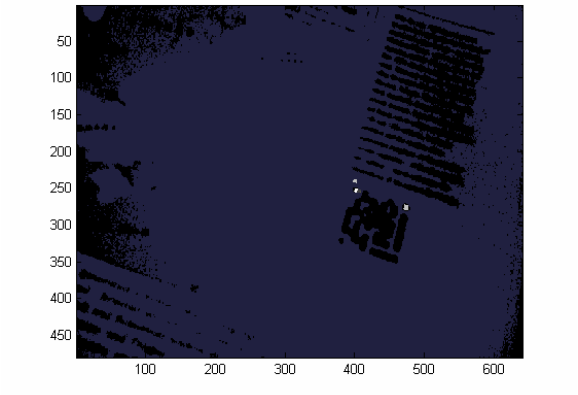

Figure 2. A dimmed version of training image #4 with our candidates for the upper-right and upper-left fixed corner elements highlighted.

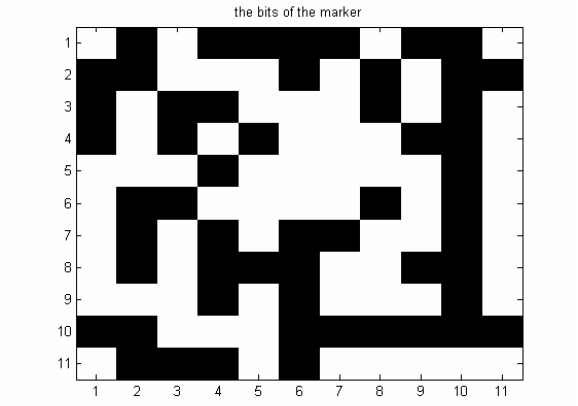

Figure 3. Plot of matrix of bits from marker in training image #4 as generated by our algorithm.

4.1. Color and luminance-based segmentation

We considered using color and luminance as criteria by which to exclude sections of the image from being potential markers. For example, any region deemed "too colorful" (i.e., having color components which were not close enough to being equal on average) could be eliminated, since the markers are supposed to contain (within a certain level of tolerance) only black and white. Similarly, any region which was "too gray" (i.e., intensity not extreme enough) could be eliminated, since the markers should only contain extremes of intensity. However, we found that these approaches were ultimately not very productive; because of the extreme noisiness and blurriness of the images, as well as the wide variation of light levels in the photos, we found that we had to be extremely conservative in applying these algorithms, to the point that it was more efficient to simply omit them.

4.2. Searching for entire markers

Initially, our mentality was that the best approach was to find the region containing each marker in its entirety. We thought that it would be easiest to find features of markers (and distinguish them from other, similar features throughout the image) if we "narrowed down" the image to a specific region of interest first. Consequently, most of our initial efforts were focused on looking for entire markers in an image, rather than smaller features such as guide bars or fixed corner elements.

4.3. Haralick corner detection

One of our earliest ideas for finding features that would distinguish markers from the rest of the image was density of corners; it seemed (and indeed, was) quite reasonable that a section of the image with so many small square elements would naturally have a higher incidence of corners than other, ordinary sections of the image. We further believed that this approach would be more productive than simply searching for edges, because corners were more unique to markers than were edges. Our idea was to pick out regions with high corner densities and perform various morphological operations (erosion, dilation, opening, and closing) with different structuring elements, along with some template matching techniques (discussed below), to find regions which enclosed each marker. What we ultimately discovered, however, was that this approach was extremely sensitive to the parameters we gave to the various algorithms involved, and it was extremely difficult to 'tweak' it in a manner that robustly avoided false positives as well as false negatives. We also found that we had great difficulty consistently arriving at the correct sizes for our marker regions. These problems led us to abandon this approach as well.

4.4. Template matching

We considered using template matching as a way of measuring "squareness," which we thought to be one of the most fundamental properties of a marker. The idea was that most markers should be approximately half white and half black, and they should have a distinctly square shape. So, we tried filtering images by a 50x50 square matrix of all 1's. Peak-finding then yielded the regions most likely to contain markers; we originally planned to use these peaks as a way of selecting among a set of candidate regions, found by morphological methods, which might represent entire markers. We ultimately rejected this approach because it depended too much on "niceness" of the marker images and was too susceptible to false positives. Indeed, once we began to focus on looking for guide bars instead of looking for entire markers, this technique became obsolete.

5. Conclusion

This project proved to be a compelling exploration of the applications of image processing techniques discussed in EE368. It resulted in a great deal of experience in fine-tuning these algorithms and understanding the kinds of tradeoffs associated with their use. We believe that our efforts resulted in a robust visual code marker detector which should perform quite well with the final test images.

6. Appendix

Allen Huang worked on designing and implementing the algorithm, focusing specifically on the detection of markers. He also worked on reading bits from markers and overcoming difficulties involved in doing this after the markers had been located. Prabhu Balasubramanian worked primarily to explore methods of correcting perspective skew in the marker images, in addition to being the primary author of this report. Both authors were heavily involved in suggesting various algorithmic possibilities, many of which did not ultimately find their way into the final product. The total estimated number of man-hours spent on this project is approximately 100.