VISUAL CODE MARKER DETECTION

Eric Chu, Erin Hsu, Sandy Yu

Introduction

With the abundance of cellular phones all over the world, along with the advent of cameras on these phones, there is a huge arena available for image processing. In Japan, this idea has already begun to be implemented, as witnessed firsthand by one of our group members. Companies or advertisers can place a marker, with some consistent markings (such as the three points and two bars in our case) and some distinct pattern and have cell phone users take a picture of it with their camera. Then, with some image processing, the bit pattern is used to determine what information should be relayed to the cell phone user.

This new way of spreading information is powerful and will continue to grow as more cell phones come equipped with cameras and as more people use cell phones. However, this also presents an image processing problem, as there will be many variations in how the user decides to take the photograph, with regard to size, lighting, orientation, and quality. Cell phone cameras are still not as advanced as other cameras, so noise and blurriness also have to be taken into account. Because the cell phone camera will be operated by people, which brings in the largest variation, there the image processing algorithm used to decode the marker must be very robust.

This report presents one way that the marker can be identified and the bit pattern returned by using filtering, morphological functions, edge detection, and perspective projections.

Code Detection Algorithm

The algorithm used to attack the code marker image processing problem begins with only an input image, of any size. The general flowchart of the algorithm is shown in Figure 1. The first main job is to preprocess the image to get rid of any clutter and noise that will get in the way. This is done by using morphological operations, filtering, and region finding to determine extremes in sizes.

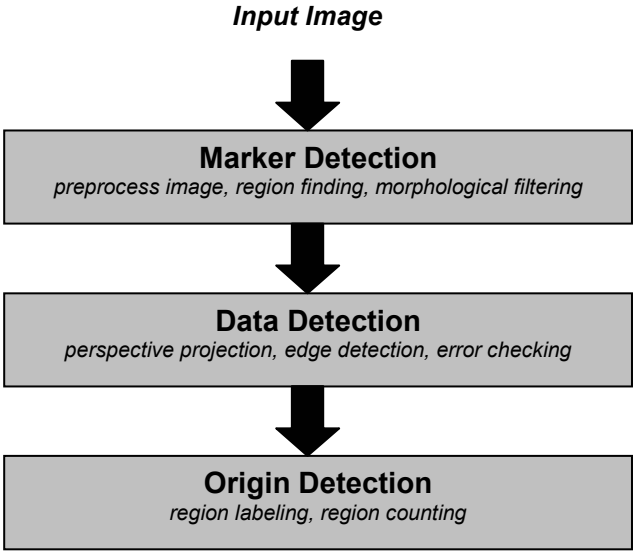

Figure 1: Flowchart of Code Detection Algorithm

Next, these images are passed on to the data detection algorithm, which uses the Hough transform to find the four corners bounding the image marker. Implicit error checking occurs, since when certain characteristics of the marker are not present, the method will throw an error and discontinue operations on a particular element. A perspective projection is used to map the image to a perfect square, and it is sampled at even spacings to get the data pattern.

This square is then passed on to the final stage of analysis, in which the coordinate of the upper left corner is extracted. This occurs by using region finding to locate the corners and various calculations to detect where the origin would be, even if the dot does not exist. The following pages describe the implementation of each portion of the algorithm in more detail.

Marker Detection

The goal of this portion of the algorithm was to identify regions that could potentially contain markers and not lose any markers in the process. First, the image was prepared for region searching by scaling the image according to color, then attempting to remove all possible color without removing the black and white data in the markers. This was done by scaling the red, green, and blue bins from 0 to 1 and removing the portions of the image that lie in the middle of each channel, and not on the extremes, which would represent white and black.

The noise was then filtered using a Weiner filter, and the image was "opened" using erosion and dilation in order to smooth objects in the image. Very large regions were removed using morphological operations, and the image was ready to be dilated.

The overall goal of the region searching was to first dilate the objects in the image, so that each square marker would become one square region. However, the weakness in this algorithm is that the structuring element used for dilation depends on the size of the squares. Also, because the markers could be oriented at

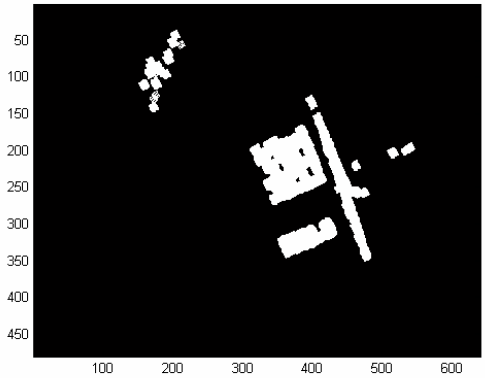

Figure 2: Training Image 1 after dilation

an angle, dilation occurs at an angle as well, so the image is less likely to become attached to another object in the image. The Hough transform was used to find the strongest angle, and a line structuring element oriented at that angle, with a width as a function of the mean of the areas in the image was used for dilation. A line oriented perpendicular to the angle was also used for dilation, so the marker was increased in both directions. Figure 2 shows the image from testing with Training Image 1 after dilation.

In order to extract regions at angles, the strongest angles were used to rotate the

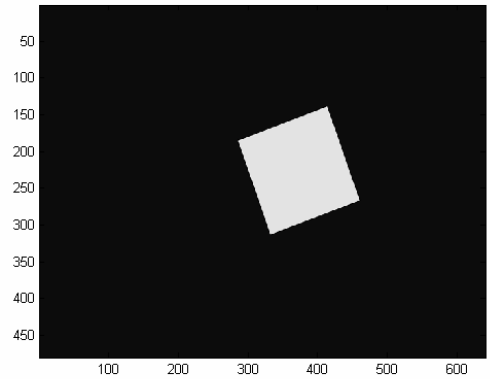

Figure 3: Rotated Mask of Training Image 1

image, with the idea being that the square markers would now be rotated upright. A function called `regionprops` was used with 'BoundaryBox' as a parameter. This function returns the coordinates of the smallest rectangle that fits around each region. In this way, a mask around each region could be found and multiplied to the original image and rotated back. An image of the marker found in Training Image 1 returns a mask as shown in Figure 3, which is slightly larger than the actual marker to ensure that none of the edges of the marker are lost, since they are very important for the rest of the marker analysis.

Knowing the boundary box coordinates also allows the portion of the region of the rotated image to be extracted and passed on to the rest of the analysis, since the marker should generally be upright and fully contained in a small image, with little clutter on the outside. Note that the image is not explicitly checked to determine whether the region actually holds a marker. This part of the algorithm simply identifies and passes on the regions that could possibly contain a marker, by

checking ratios and some sizes. For Training Image 1, the marker image passed on is shown in Figure 4, and the rest of the clutter in the image is not passed on because it does not approximately resemble a square.

This algorithm was successful in finding at least part of every marker in the training set, along with some extra regions

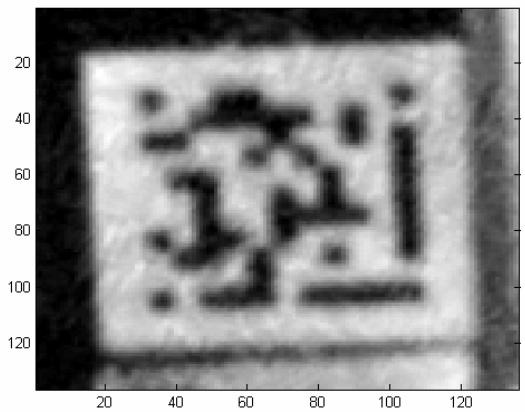

Figure 4: Region found for Training Image 1

with no markers inside, but it had difficulties when the size of the markers was very large or very small. This was the main weakness of this algorithm. Moreover, although rotation aided in acquiring masks at less of an angle, it caused problems later in calculating the upper-left point by inverting rotation because the function `imrotate` acted differently than trigonometry would expect.

Data Detection

Taking the image masks, the relevant information for finding the data is the orientation of the image marker and the spacing between the information on the image. To prepare the image for processing, the image was normalized by subtracting its mean and setting the standard deviation to one. This procedure "standardizes" the contrast in the image and allows for better threshholding later.

All incoming images also underwent morphological transforms (closing and dilation) to accentuate the edges of the markers.

Applying the Hough transform to the processed marker, the dominant four lines are chosen. The algorithm is written such that if the masked image is not an image marker, then the Hough Transform tends to have a tough time returning four lines that define a square. Hence, the algorithm throws and error and discontinues operation on the false image marker.

Having found the four sides of the square, the four corners can also be found (i.e., the intersection of the four lines). The upper left corner is also found at this point to facilitate proper orientation of the image marker. Using these four corners, a perspective projection re-maps the image marker to a regular square.

The data is then easily accessible (after threshholding) from the square. Figure 5 shows the resulting image with the dots showing where the data was collected.

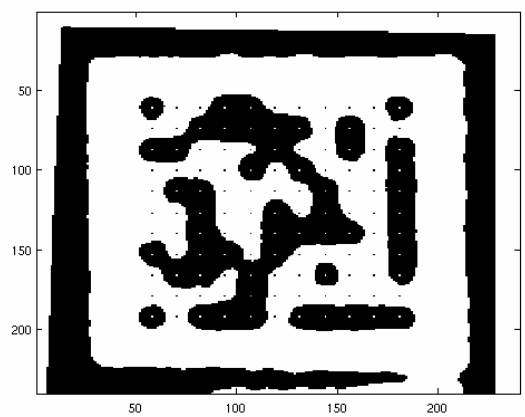

Figure 5: Image marker with proper orientation and data points marked. Note blurring around edges of data "squares".

Note that this algorithm is very sensitive to dilation, since the individual data points tend to occupy a small number of pixels (compared to the 640x480 image). A significant amount of erosion can be seen in figure 5. It is thus possible to corrupt data in the process of transforming and dilating the marker to get the appropriate data. Also, because of the reliance on strong lines for the Hough transform, artifacts surrounding

the image may interfere with the determination of the actual four corners and four sides of the square.

An alternative method to data detection would have been to use an averaged template (by taking the pixels around the ground truth upper left corners) and find the single dots on the image markers—then finding the spacing between the data points. However, the template matching algorithm also tended to pick up some noise and peaked along the guide bars as well, so the approach was discarded.

Origin Detection

Due to the complex nature of the marker content, a simple template matching will not suffice to find the upper left-hand corner. The basis of the implementation comes from using existing data after large region removal, namely the three corner dots and sometimes the guide bars, as seen in figure 6 The coordinates of the upper-right and lower-left corners are important because they represent the diagonal of the square. Initially, region labeling and region counting are used to find the coordinates of the upper-right and lower-left corners. From there, several implementations are possible:

I. **After eliminating the upper-right and lower-left corners, find the small region that is closest to two orthogonal sides of the square image**

Initially, this is the most straightforward implementation. However, since the orientation of the marker is unknown, it is difficult to find which two sides of the square are relevant for the upper-left dot. Furthermore, since the location of the upper-left dot will not be exact (due to erosion processes), other dots in similar locations could yield misleading results. This implementation was not robust and depended highly on the original marker image being square and flat, which is not true for real world applications.

II. **Find the guide bars and the upper-left dot will be across the diagonal of the square**

This implementation avoids many lengthy calculations, but relies on the guide bars being present in the input image, which is not always the case. Since large regions are removed in advance, the guide bars are sometimes removed as well. Thus, this method is not adaptive to different images and is not useful.

III. **Using the coordinates of the upper-right and lower-right corners, find the general region that the upper-left dot is supposed to be in and search for it.**

This implementation turned out to be most promising because it used all the information that could be calculated or inferred from marker qualities and the upper-right and lower-left coordinates. After the coordinates for the upper-right and lower-left are calculated, one of the remaining corners has either a guide bar or nothing, and the other remaining corner contains the upper-left dot. Next the area of a dot is calculated, using the upper-right or lower-left (whichever is larger) as a reference. The area provides a radius of search to find small labeled regions near the two other corners. After the two regions near the upper-left and lower-right corners are labeled, the size of the regions can be used to determined which is the guide bar (or nothing) and desired dot. The reasoning is that if a region area is large >3600, it means there is no labeled region in the area. If there are labeled regions in both corners, then the smaller region will represent the dot and the larger will represent the guide bars. By this point, the upper-left pixel coordinate of the upper-left dot is determined, and using geometry the coordinates of the middle of the dot can be calculated.

This implementation is accurate when the

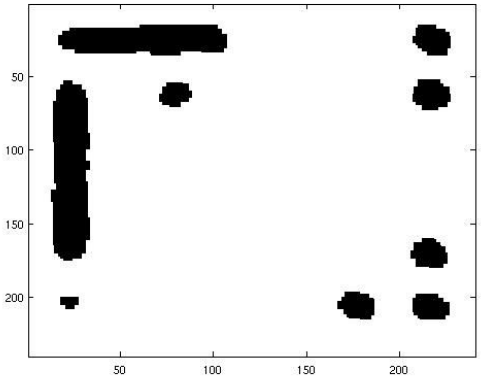

Figure 6: Input image for origin detection, in which larger regions have been removed.

noise is low in the marker image, or else small dots could be mistaken as corner guiding dots and used in the calculations. Otherwise, this implementation takes into account shifts, rotations, and some error margins for accurate detection of the upper-left dot coordinates.

Conclusion

The implementation and algorithm work for most cases, but there are sources of error. Sources of error in the marker detection stage have a significant effect because data is potentially lost at every further stage of the implementation. Initially, the different sizes of markers in one images can cause confusion and result in false positives. Furthermore, the angle does not always align with the markers, which creates errors for subsequent calculations.

Other image processing techniques including cropping, rotating, and dilating also result in loss of information, and are especially difficult to recover from. In the future, not using `imrotate` will allow easy inversion of scaled coordinates of the marker to the original coordinates in the image. Other ways of detecting the upper-right hand corner can also be developed to increase the accuracy. There were many unanticipated issues that required significant testing and redesign, but most were resolved. This project was a thorough application of image processing techniques learned throughout the quarter, and was a good exercise in weighing tradeoffs when making implementation decisions.

Distribution of Roles

Eric Chu

- Wrote Data Detection Algorithm and Code

Erin Hsu

- Wrote Marker Finding Algorithm and Code

Sandy Yu

- Wrote Origin Detection Algorithm and Code

** Other duties were equally distributed, such as discussing the algorithm, helping each other debug, and writing the report.