Visual Code Marker Detection using Digital Image Processing Techniques and Matlab

M.M. Limlamai
Stanford University | EE368 Final Project

***Abstract*-In a climate of advancing technology, information can be shared more extensively and more broadly. One way is to provide more information about everyday items is with a visual code marker. These two-dimensional barcodes can contain information that might normally not be included in legible form. Thus, these barcodes would require the intervention of some detection and discerning algorithm. Based on camera-phone pictures as inputs to a code that can determine the contents of the visual code marker, these markers can share information to a vast number of consumers without a large impact to current resources available. The one algorithm described here uses a collection of digital image processing techniques such as template matching, morphological operations, and perspective transformations. Even with the complexity of these digital image processing methods, the algorithm described is fast and accurate.**

I. Introduction

The growing desire for quick access to information sometimes poses a problem due to a lack of resources, be it customer service representatives or easy access to the internet. This problem has led to one solution in the form of visual code markers paired with digital image processing. The idea is to proliferate these two-dimensional barcodes onto everyday objects and have them readable by the average cell phone camera [1]. This gives rise to two immediate difficulties in the image processing realm: accuracy and speed. Because the detection and deciphering of these visual code markers is to be done routinely, there needs to be a way to find these markers and their information speedily. More importantly, the algorithm that is operating on the cell phone needs to be accurate so as to retrieve the correct information that is linked to a specific visual code marker. Because of the portability of these visual code markers, the algorithm also needs to be fairly robust – able to handle a variety of angles, perspectives, sizes, and rotations. This paper outlines, in moderate detail, one method of detecting visual code markers. The algorithm uses a variety of image processing techniques including thresholding, morphological operations, template matching, segmentation, filtering, and projective transformations.

II. Background

A visual code marker is an 11x11 square comprised of black and white bits. For digital detection of these codes, two guide bars and three guide points are included as fixed points in every visual code marker. Of the total 121 bits, 83 contain specific information. This information is read out top to bottom, left to right; slightly different from the way we read.

These visual code markers can be of any size, placed anywhere, and imaged from all angles and perspectives. These codes will then carry information pertaining to what it is attached to so that anyone with a camera phone can pass by, take a picture, and retrieve the code's information. Potential uses include merchandise price, product information, naming labels, and many more.

III. The Algorithm

One major assumption this code makes is that the input image is 480x640. A simple function call and replacing variables will correct this and will be added in the next version of the code. The algorithm also assumes that the input image is in color because the color information is removed and the image readjusted to have a maximum value of one and a minimum of zero. The rationale behind this first step is because the visual code marker is a square of black and white squares. A white value in RGB space contains no additional information than a white value in intensity space. Therefore to save on coding complexity, only one dimension is preserved for color.

A template matching routine is then used to find the three guide points and two guide bars of each visual code marker in the image [2]. Because the size of the visual code marker is sometimes small, a morphological operation is used to grow the areas that positively matched the template to better define them from the background. This new intensity image is then region labeled to remove small regions, and segment potential guide points from potential guide bars. This segmentation is accomplished by finding the eccentricity of each region and setting certain thresholds for what could be a guide point and what could be a guide bar. Processing stops for now on the potential guide points, while further refinement on the potential guide bars is done.

A Euclidean distance function is applied to the segregated image and then thresholded to find guide bar pairs [3]. Note that these potential guide bars are not yet named to each other, i.e. they are still bars floating in space even though the thresholded distance image contains joined guide bars. This new intensity image is again region labeled and each region's eccentricity and extent are determined. The extent is the area to bounding box ratio. This information is used to weed out highly eccentric bars (which are mostly edges in the image)

and areas that appear as squares [2]. What remains is a small subset of potential guide bars.

The algorithm then returns to considering the potential guide points. Specifically, a guide point is paired with a guide bar. To make this work, detailed information about each guide bar is required, and knowing things like the length, and the angle of orientation help determine guide point pairs. The algorithm also uses knowledge of the visual code marker itself, namely the fact that there is a long and short guide bar [4]. Currently, the algorithm begins with the first potential guide bar and searches all potential guide points that lie within a realm of possible pairs for that guide bar. This exhaustive search does have a negative impact on speed, but more will be discussed later. If a guide point-guide bar pair has been found, it is recorded in a variable to be later used with all pairing information.

After this comprehensive search, the algorithm has filtered the image into potential short pairs and potential long pairs. A potential short pair is a pairing of the short guide bar with its immediate corner neighbor in the direction away from the long guide bar. The potential long pair is the reverse. These pairs are then matched to each other based on an intersection of the imaginary line that connects each point to its respective pair (see Fig. 1). This intersection should go straight through the guide bar corner. The algorithm now has information on three of the four corners for any visual code marker in the image. Fig. 1 shows the intersection of two paired guide bars with a red dot in addition to the yellow and green dots which are the related guide points. The blue dot is the inferred position of the fourth corner and confirmed by searching available guide points.

Based on the distance and orientation of the known corners, a guess is made on where the remaining corner will appear. Another search is performed on the remaining potential points to determine the nearest region to the guessed location. Unlike the earlier exhaustive search, the nature of this variable as it is handled in Matlab allows for expediency of the code. The conclusion of pairing and searching ends with four points specifying the corners of all visual code markers detected in the image.

Because these visual code markers are meant for everyday use, each object plane does not necessarily match the image plane. Also, there is no requirement for the input image to be rotated so that the guide bars (in the lower right-hand corner) form a ninety-degree angle in the image. After the algorithm determines the four corners, a transformation is completed to account for the two complications described above. This projective transformation returns the guide bars to the lower right-hand corner and makes any skewed visual code marker appear square.

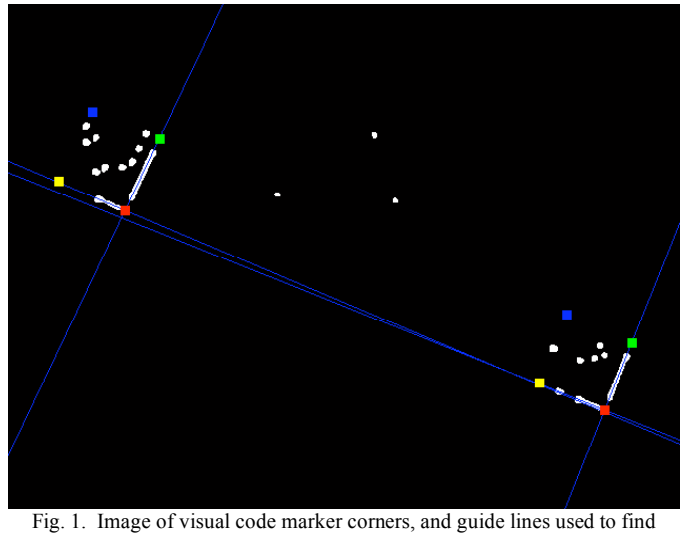

Fig. 1. Image of visual code marker corners, and guide lines used to find pairings between guide bars and guide points.

This new image, which is sampled from the original intensity image of the scene, is fed into another algorithm that determines the contents of the marker. Because the size of each visual code marker varies locally in the image (sometimes more drastically than others) this new algorithm has to determine a proper spacing within the visual code marker. Each square consists of many pixels from the original image that are sampled and a new image is created that is exactly 11x11. Another image is created that contains the mean of the nearest four neighbors for each pixel in the guessed visual code marker. The mean image is then used to threshold the guessed image. This process is more robust than a global threshold because small white regions in the midst of a large black region will be called black using a global threshold. This local thresholding allows a small white square to be readily distinguished from its surrounding black neighbors more easily.

Finally, this algorithm returns to the main algorithm a vector of size Nx83 bits where N is the number of visual code markers, and 83 is the number of non-guide elements in the visual code marker. The result of the main algorithm is a location for the marker, based on the upper left-hand corner of the visual code marker, which has been determined by the main algorithm, and the visual code marker data as determined by a discerning algorithm.

IV. Accuracy and Speed

Tests were done on the accuracy and speed of this algorithm using the training set of images provided through the course website [2] [3] [4]. Due to time constraints, other visual code markers placed on everyday objects were not captured with a camera-phone and thus were not passed into the algorithm. Future validation of the code would require a larger training set based on more variations of the input image, like brightness, orientations, and perspectives. The code currently can handle a variety of these input variables, but more images would allow for more refinement of the accuracy of the data this algorithm produces. Rotating and inverting the

given training set would double the current training set count, but an unfortunate time crunch will not allow for that to happen in this first version of the code.

Currently the algorithm has a performance characteristic as shown in Fig. 2. The errors are due to one of two causes: (1) the local thresholding using the nearest 4 neighbors is not specific enough. That is, if the algorithm used the nearest 8 neighbors to compare averages, the discerning might be improved. Additionally, using the original image to determine the mean of the neighbors might give rise to more balance weighting of the original image. The other cause (2) could be misalignment of the visual code marker after it has been handed off from the main algorithm. Because the perspective transformation returned from Matlab is not perfect, determining the size of the right-angled-square is difficult and adds an error in discerning.

To expound on Matlab's technique of perspective transformation, the author will attempt a brief explanation. A perspective transformation requires at least four points from the original image to properly determine the resulting four points in the output image. But the user also has to specify where in the new image these four points will map. Ideally, the corner points will make a 100x100 square in the transformed image. In practice, though, this is not always the case. Thus, in the transformed image, most of the corner information is lost or distorted. Further understanding of Matlab's transformation techniques would alleviate this issue, but updates and improvements will come in the next version of the code.

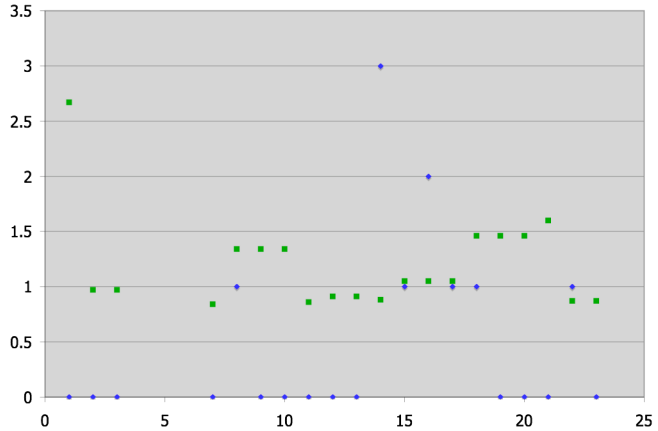

Fig. 2. The x-axis represents one point for every marker. The blue dots are false bits, i.e. a 0 appeared as a 1 or vice versa. The green dots are the time in seconds. The data points are both plotted on the same scale.

The speed of the code, in the author's humble opinion, is impressive. For most images, the code completes in under a second on the scien machines (see second axis in Fig. 2). The reason for delay is because of the exhaustive search to find pairings between each guide bar and one guide point. Not all the input images from the training set caused this slow algorithm response, because other features in the image did not match the criteria for potential guide points. But, specifically, in the case of training_1.jpg, the background appears to be visual code marker guide points even though it is out of focus and in the wrong object plane. Unfortunately, the code does not know how to account for the three dimensional distance from the camera to the object, so in the input image it simply appears that there are many guide points. Favorably, these guide points are discarded once the algorithm determines there is no matching guide bar.

Unfortunately, there is a problem with my code. Because it fits a line to each guide point, and then uses that line to determine a related guide point, the markers that closely match the image frame (i.e. markers that look like squares and have a ninety degree guide bar corner) cannot find matching guide points. I just realized this a little too late, and will not be able to fix it in this version of the code.

V. Conclusion

By using known digital image processing techniques, this algorithm is able to quickly and accurately determine the contents of most visual code markers. The code is able to take an input from an average camera-phone, determine where a visual code marker is located, and discern the marker's data in usually less than 3 seconds. Methods that determine visual code marker location include template matching, morphological operations, and segmentation. Discerning the marker's data involves perspective transformations and thresholding.

Accuracy and speed can be improved by utilizing the vector-based nature of Matlab rather than performing computationally expensive for loops to search for guide bar and guide point pairs. The accuracy might also be improved by using more information from the image than the current 4-neighbor averaging threshold.

References

[1] M. Rohs, "Real-World Interaction with Camera-Phones"
[2] B. Girod, *Personal conversations*.
[3] A. Mavlankar. *Personal conversations*.
[4] C.L. Chang. *Personal conversations*.