EE368 Final Project Report: Visual Code Marker Detection

Stefan Schuet
Dept. Electrical Engineering
Stanford, CA, USA
sschuet@stanford.edu

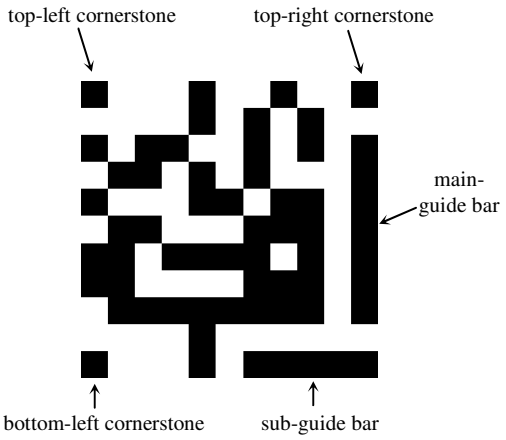
Abstract

This report presents a method for detecting visual code markers in cell phone images.

INTRODUCTION

Visual code markers are binary information bearing tags similar to bar codes except that they can be read with the use of a cell phone camera and detection algorithm (such as the one presented in this report). The concept of a visual code marker and its various uses are presented clearly in Michael Rohs' paper, "Real-World Interaction with Camera-Phones." [1] In addition, this paper presents a method for detecting the markers which was used as a starting point for the development of the algorithm presented here. While I take the same general approach as the one presented in Roh's paper (because I thought it was a good one), my implementation of it differs in many respects and fills in a few details that were left out. This report focuses on the specification of the algorithm I created to detect and read visual code markers in cell phone images.

I begin by presenting the visual code marker shown below along with a few definitions of key features in the marker. Please note the names assigned to these features as they are referred to throughout this report.



My method (which is based on Roh's method) for identifying and reading the visual code marker consists of the following steps:

1. Obtain a binary image that contains the marker identification features shown above.
2. Generate a labeled map containing each object in the binary image.
3. Identify the main-guide bar, sub-guide bar and cornerstone objects in the labeled map.
4. Extract the code from the identified visual marker.

PREPROCESSING

The input image is a 640×480 color jpeg image. This is converted into a gray scale image representing luminance via the formula in our class notes: $Y = 0.177 \times red + 0.813 \times green + 0.011 \times blue$ [2]. This gray scale image is then expanded to 670×510 by mirroring intensity values about its edges. This is done to mitigate edge effects from the application of a high pass filter described later. Next a 3×3 median filter is applied to the expanded gray scale image to remove sharp intensity changes and reduce noise in the image.

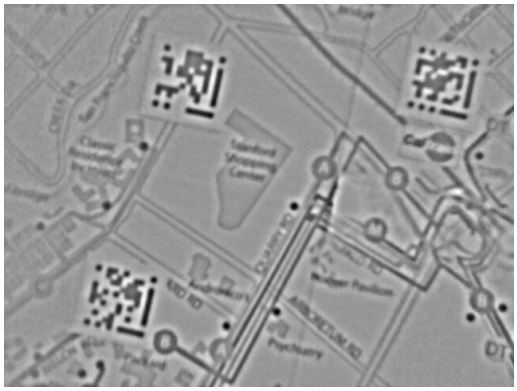
This gray scale image is then preprocessed in an effort to prepare it for the application of a global threshold. This is necessary because uneven lighting conditions, often caused by the presence of glare, result in images where the marker features have varying intensities of gray. This leads to detection problems when a global threshold is applied to the image directly. To solve this problem, a high pass Gaussian filter is applied to the expanded gray scale image. This is accomplished by first filtering the input image with a 22×22 low pass Gaussian filter kernel with standard deviation 11. The output high-pass image is then found by removing the low-pass image from the original.

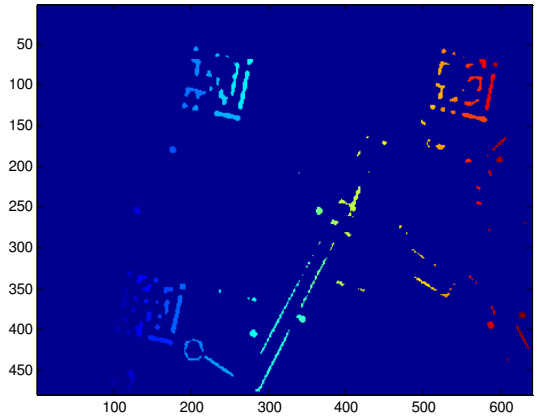
Next, the central 640×480 region of the high-pass image is retained and rescaled to stretch it over a range from 0 to 255. A global threshold is applied to detect dark regions in the image, which should include the visual marker's detection features along with clutter caused by perhaps similar looking non-marker elements in the image. Note also that if the visual marker is large the high-pass filter will attenuate internal regions of the marker features, which in turn may lead to the detection of only the marker feature edges. However, the feature identification method presented next is insensitive to this phenomenon.

Finally, the binary image obtained from the previous step is labeled with the typical region labeling algorithm presented in class (or more specifically *bwlabel* in Matlab) [2]. Any object touching the border of the image is removed from this map before it is passed on for further processing. The 640×480 central region of the median filtered gray scale

image is also retained and used later to decode the visual markers.

The following three images illustrate the glare suppression and detection achieved by this process (taken from training image 9).





The first image is the median filtered luminance image. Note how the glare drowns out the more pertinent marker features. The second image is the high pass filtered image with almost no glare. The third image shows the labeled map obtained after applying a global threshold of 90 to the second image (pixels with intensity less than 90 are taken as 1).

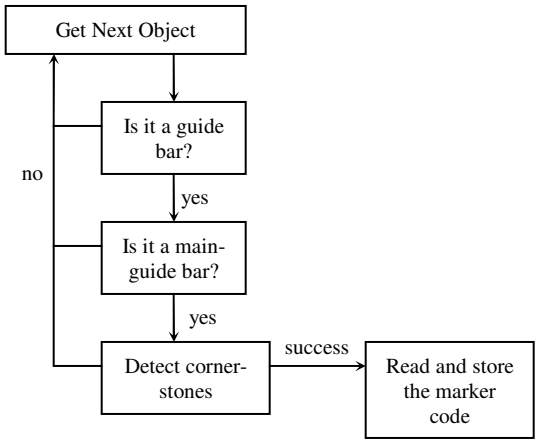
All the preprocessing described in this section is implemented in two functions called *createcomposite* for the filtering, and *getlabels* to label the binary map and remove objects touching the edges.

FEATURE IDENTIFICATION

The goal of the feature identification process is to find the main-guide bar, sub-guide bar and cornerstone marker features in the labeled object map.

From the start I wanted to design an algorithm that was not sensitive to marker position, size, orientation, or perspective distortion. These goals make the use of matched filtering to identify the markers impractical since the template would have to be translated, scaled, rotated, and distorted in order to find the best match. Instead, the decision was made to identify marker features by detecting objects in the labeled map that meet marker feature requirements regarding shape, orientation, and position with respect to other objects in the image. This is also the approach taken in Roh's paper [1], but as mentioned previously there are a few differences in the way this algorithm is implemented.

An overview of how the feature identification algorithm works is diagramed below:

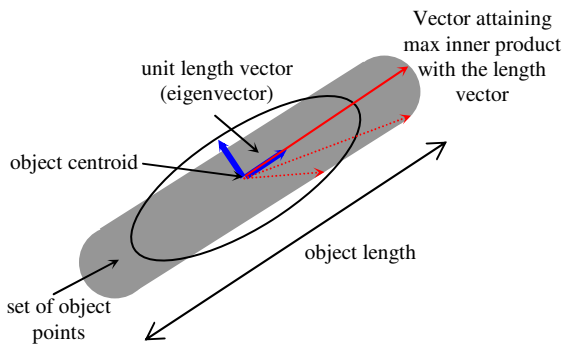


The key to this algorithm is of course how the questions presented in the diagram are answered. A description of each one is presented below.

Is it a guide bar?

To answer this question the length and width of the current object are measured. This is accomplished by first finding the covariance of the coordinate positions of all the pixels making up a particular object in the image. This matrix defines an ellipse that will generally be aligned with the axes of the object. Furthermore, the normalized eigenvectors of the covariance matrix are unit orthogonal vectors that point along the objects length and width. The length and width are then defined as twice the maximum of the inner product between the corresponding eigenvector and

each point in the object. The following figure illustrates the process of finding the length of a given image object.



Note that this method for finding length and width is insensitive to whether the object is defined by its entire body or just its edges, and is representative of the actual object length and width in pixels – making it easy to think about the physical properties that would make the object a guide bar. In addition, the length to width ratio and area of the object (defined as length × width) are calculated.

These values are compared against a set of constraints used to define a guide bar object in the image. If a particular input object does not meet these constraints is thrown out. The constraints for guide bars are presented in the table below.

Guide Bar Constraints	Value
Length (min,max)	(5,301)
Width (min,max)	(1,45)
Length/Width (min,max)	(3,15)

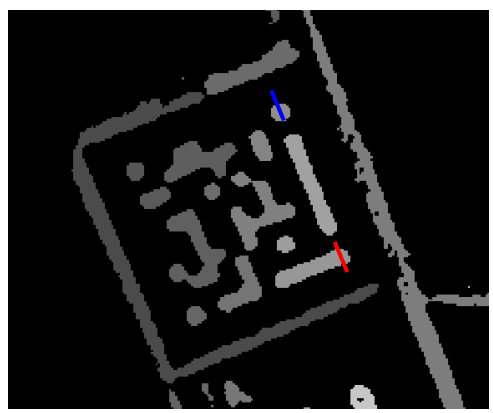
These were obtained by selecting reasonable values based on the maximum and minimum allowable visual marker geometry. For example, the minimum length and width of the sub-guide bar must be at least five and one respectively. It should also have a length to width ratio of five. Also, since the input images are 640×480 the maximum size a marker can be is 480×480 – a scale factor of about 43. However, image distortion forces one to loosen these values a bit. This is reflected in the table where values were obtained by starting from reasonable ones and then manually tuning them until the algorithm performed correctly on a set of training images.

To summarize, if the current image object satisfies the size constraints discussed above, it is considered a main-guide bar candidate and stored for further processing. Note that size information alone does not determine whether the guide bar candidate is a main-guide bar or a sub-guide bar. Depending on perspective distortion in the image the main-guide bar can actually appear shorter than the sub-guide bar and that problem is perhaps best solved by looking at the objects that neighbor the guide-bar candidate detected here.

When the function implemented to execute this step, called *isguidebar*, is successful it returns with a structure containing the coordinates of each pixel in the object, the object's length and width measurements, length and width unit direction vectors (i.e. the eigenvectors of the object covariance matrix), along with the mean x and y position of the object in the image – which I often refer to as the object's centroid. There is also a similar function, called *iscornerstone* that accomplishes the same for cornerstone objects. The inputs to both of these functions are two vectors containing the x and y image coordinates of all points in the object under consideration.

Is it a main-guide bar?

A main-guide bar is distinctly defined as a guide bar that has another guide bar neighbor on one end (i.e. the sub-guide bar) and a cornerstone on the other. This will be true regardless of the markers size, orientation and tilt. So the task is simply to select the objects (if there are any) that lie above and below the current guide bar candidate. This is accomplished within the image by using the length measurement of the current guide bar candidate to estimate the distance to the possible objects above and below it, and then using two projected line segments to select those objects. To see how this works consider the image below (taken from training image 1):



The image shows a candidate guide bar along with the selection lines used to get its neighbors. Any object under the red or blue selection line is checked with the *isguidebar* and *iscornerstone* functions. If there is one of each it becomes quite possible that we have found a main-guide bar. Note: if multiple objects fall under a selection line, the object closest to its center is taken as the neighbor.

When the function that implements this step, called *ismainguidebar*, is successful it returns with a structure that identifies the associated top-right corner stone, main-guide bar, and sub-guide bar objects in the image. As an input, this function simply takes the output from the *isguidebar* function described earlier. As a last effort to identify specious sub-guide bars the function also checks that the angle between the sub-guide bar and main-guide bar length direc-

tion vectors is greater than 45 degrees. This turned out to be a very important constraint since it has the power to reject false positives that can still arise when all other constraints are met (often caused by broken line segments in the labeled object map). Also, before exiting with success the function ensures the main-guide bar length vector points towards the top right corner stone and that the sub-guide bar length vector points away from the main-guide bar (i.e. to the left in a rectified marker image). This becomes important later when correspondence points are found between the image and visual marker coordinate systems.

Remaining Corner Stone Identification

In order to complete the marker detection process all that remains is to detect the top and bottom-left cornerstones.

The first step in this process is to estimate the location of the bottom-left cornerstone with an affine transformation. To find the appropriate affine transformation the following correspondence points are found in the image:

Image Points	Corresponding Visual Marker Coordinates
1. centroid of top right cornerstone	(10,0)
2. top of the main-guide bar	(10,1.5)
3. bottom of the main-guide bar	(10,8.5)
4. center of sub-guide bar's right most pixel	(10,10)
5. far left of the sub-guide bar	(5.5,10)

With the information output from the *ismainguidebar* function we can derive the locations of these points in the image as follows. The centroid of the top-right corner stone is known. The formulas for the top of the main-guide bar (x_{mgt}), bottom of the main-guide bar (x_{mgb}), and far left of the sub-guide (x_{sgl}) bar are shown below:

$$x_{mgt} = m + \max_i[(x_i - m)^{\mathrm{T}} l_m] l_m$$

$$x_{mgb} = m + \min_i[(x_i - m)^{\mathrm{T}} l_m] l_m$$

$$x_{sgl} = s + \max_i[(x_i - s)^{\mathrm{T}} l_s] l_s$$

Where m, l_m, s and l_s are the main-guide bar centroid, normalized main-guide bar length vector, sub-guide bar centroid, and normalized sub-guide bar length vector respectively – note these are all vectors in R^2. The max is taken over all coordinates x_i that make up the main-guide bar or sub-guide bar object – whichever is appropriate.

Finally, the center of the sub-guide bar's right most pixel is obtained by calculating the intersection between lines projected along the lengths of both the main and sub-guide bars.

Armed with the knowledge of these five points, and of course the corresponding points in the visual markers coordinate system shown in the table above, the affine transformation between the two coordinate systems is estimated via least-squares, a method presented in class [2]. Using this transformation an estimate for the position of the bottom-left cornerstone is calculated. Note: an affine warping model is used here because there is not yet enough information to determine perspective warping reliably. Finally, the cornerstone object closest to the estimated location is selected (if one exists within a certain search radius equal to twice the measured width of the sub-guide bar).

Detecting the top-left cornerstone is the most interesting task. This is because its distance from the other features is traversed over an as of yet undetermined perspective warping. To get the best chance of finding it I add the location of the just found lower-left cornerstone to the list of know correspondence points and this time estimate a perspective transformation between the marker and image coordinate systems. Again, the cornerstone object closest to the estimated location is selected if it exists.

To estimate the perspective warping transformation between the marker and image coordinate systems I again resort to least-squares. Although the perspective warping is nonlinear in its parameters (as discussed in class) we can still find them with a set of linear equations that depend on non-linear functions of the observed correspondence points. To see this recall the perspective displacement field from class [2],

$$x' = \frac{a + bx + cy}{1 + gx + hy} \qquad y' = \frac{d + ex + fy}{1 + gx + hy}$$

and note that we can simply rearrange the terms to get,

$$x' = a + bx + cy - gxx' - hyx'$$

$$y' = d + ex + fy - gxy' - hyy'$$

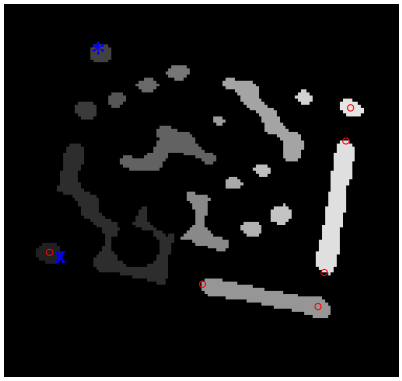
which becomes a pair of linear equations for each pair of correspondence points. So with n correspondence points we get $2n$ equations that can be solved via least-squares (provided there are no rank deficiencies, which I check for in my algorithm).

When the function that implements this part of the algorithm, called *completemarker*, is successful it returns with a structure that identifies all guide features in the visual marker.

As a final remark, this process works well when the markers are only rotated or scaled and begins loosing its potency as perspective distortion in the image increases. Nevertheless it remains tolerant of distortion for two reasons. The first is that corner objects closest to the estimates are selected so that we don't have to predict there locations perfectly. The second is that I intentionally do avoid using the centroids of the main and sub-guide bars, since under perspective warping they do not generally correspond to their respective centers in the marker coordinate system.

The image below (taken from training image 12) indicates the type of result achieved by this function. The red circles to the right show the locations of the five known marker

correspondence points presented in the table. The lower left blue x marks the estimate of the lower-left cornerstone location obtained from the affine transformation. The upper left blue * marks the estimate of the top-left corner stone's location obtained with the perspective warping transformation.



READING THE MARKERS

With the known locations of all the marker guide features we are finally ready to read the code it contains.

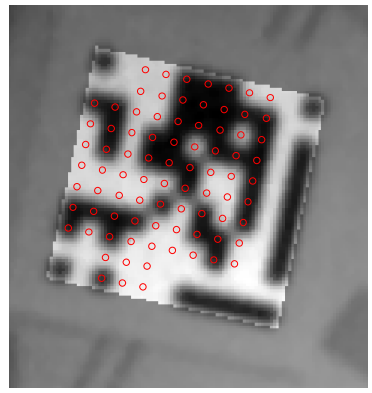
The algorithm that accomplishes this task begins by re-estimating the perspective warping using the centroid of the three cornerstones and sub-guide bar lower right corner. These points in the image correspond to marker coordinates (0,0), (10,0), (0,10), and (10,10). They are all points that are easy to get, trust worthy, and the least susceptible to correspondence inaccuracy caused by perspective distortion.

With the known perspective transformation an estimate for the corners of the distorted box that surrounds the marker is derived. This is accomplished by finding the points in the image that correspond to the marker coordinates (-0.5,-0.5), (-0.5,10.5), (10.5,10.5), and (10.5,-0.5). The image coordinates then define a polygon that surrounds the marker. From this polygon a binary mask is generated (using *roipoly* in Matlab) which is used to precisely select the image pixels that comprise the visual marker.

In the next step the mask is used to select a sub-image, which is then completely reprocessed to get the best possible detection result. The reprocessing begins by stretching the image intensity values under the polygon mask to fill an entire intensity range from 0 to 1. A threshold (of 0.5) is applied to get a binary image which is then labeled (via *bwlabel* – it's just so useful). Each marker guide feature is selected in the new labeled map and then checked and processed again with the appropriate *isguidebar* or *iscornerstone* function. The perspective transformation between the marker and image coordinate systems is then estimated for the final time the same way it was before, except this time using updated coordinates for the marker's guide feature locations.

We are finally ready to read the visual marker's code. This works by first converting the marker code coordinates into image coordinates via the most recent perspective transformation. A bilinear estimate is then found for the image intensity at each of these coordinates. Finally, a threshold is applied to obtain the marker code (intensities less than 0.5 are taken as 1).

The image below shows one example result achieved using the process described in this section (taken from training image 9).

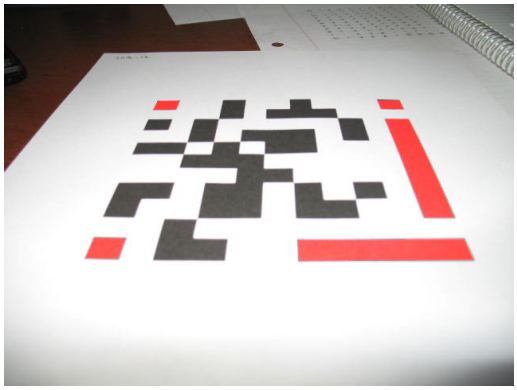


The red circles indicate the points where the visual markers intensity is evaluated to determine its code. Also note the brightening of the marker. This is the region under in the binary mask previously discussed.

SUMMARY

The algorithm for marker detection and decoding presented here works by progressively identifying the visual marker's guide features. If it can find a guide bar, it checks to see if it's a main guide bar. If so, it tries to find the cornerstones in locations estimated by an affine mapping for the bottom-left cornerstone and a perspective warping for the top-left one. During this entire process if an expected feature is not found, or does not meet the constraints the process is aborted and the loop continues with the next object from the labeled binary map.

This algorithm was tested on the set of training images provided for the project. It was able to successfully detect and decode all the visual code markers without any false positive or negative readings (using the *evaluate* function provided on the class website). The algorithm was also tested successfully on synthesized images with pure uniform random noise to see that it would not find any markers in the noise. Finally, to show that the algorithm works on visual code markers of different sizes – a goal I wanted to accomplish from the beginning – I present two images of both very large and very small visual markers. Note: the algorithm's ability to successfully detect these markers is evidenced by the program changing the color of the detected marker guide features red.



REFERENCES

[1] Rohs, M., "Real-World Interaction with Camera Phones", Swiss Federal Institute of Technology, Zurich, Switzerland.

[2] EE368 Class Notes, Spring Quarter 2006