EE368: Digital Image Processing Project Report

Ian Downes downes@stanford.edu Stanford University

Abstract—An algorithm to detect and decode visual code markers in medium resolution images is presented. The algorithm uses adaptive methods to segment the image to identify objects. The objects are then used to form candidate markers which are examined for several criteria. Potential markers are then sampled and guide information present in the marker is used to verify the data. The algorithm is invariant to scale and rotation and is robust to motion blur and varying illumination. The algorithm is implemented in C and takes approximately 100 ms to execute per image using a Pentium 4 3.40 GHz computer.

I. Introduction

The presence of cameras in cell phones is becoming extremely commonplace as the price of inclusion plummets. As more and more people are equipped with these cameras it becomes feasible to develope a range of applications that utilize the camera for purposes other than simply taking a snapshot. One such application is to use the camera to sample the data contained in a visual code marker and to use this as a code to reference information. A typical use case might be to include a visual code marker next to the advertisment for a movie. By taking an image of the marker the phone can decode the data and then query a backend database for the local screening times of the movie.

To achieve this type of application the camera must be able to reliably identify and decode visual markers. This report details the development and implementation of an algorithm that can be used for this purpose. The report first analyses the problem and establishes the requirements for the algorithm. It then examines the major steps of the algorithm and how it meets the requirements. The results of testing the algorithm are discussed before the report concludes with a summary.

II. ALGORITHM DESIGN

Before the design of the algorithm was started the problem was analysed to develop some ideas about how the visual code markers could be effectively detected and the issues that would need to be solved. Once a basic high level method was established the lower level stages of the proposed algorithm were designed and tested. The follow sections detail how the problem was analysed and the resulting algorithm.

A. Problem Analysis

Figure 1 shows the format of the visual code marker. The marker is composed of an 11×11 grid of binary elements. The binary elements are represented by a black square for a '1' and a white square for '0'. The origin of the array is defined to be

the top left corner (that is, the corner opposite the intersection of the guide bars) and the data is ordered in column major order. The central region of the marker contains 83 bits of data. The remaining 38 bits are spaced at the corners and are used to form guide elements. The guide elements consist of three corner square elements and two guide bars. The guide bars are perpendicular to each other and are of different lengths. In addition to the 11×11 array the marker also has a white border extending (at least) one element width around the array. This border serves the same purpose as the border around barcodes and ensures that the marker is separate from any background it is placed on.

The image source is a digital image from a camera equipped cell phone. The image is of VGA resolution (640×480) and provided as a compressed 8-bit RGB JPEG file. The compression ratio is approximately 18:1. This resolution is comparatively low but it is of course definately adequate to sample and extract the data from the markers under the right conditions. Of more importance are the conditions under which the image is obtained. Specifically, the camera is most likely to be handheld, leading to a degree of motion blur, and to be positioned somewhat haphazardly, leading to arbitrary transformations in angle, translation and perspective. Furthermore, the lighting conditions are not controlled and it is possible that the illumination will not be constant over the image and definately not between images. The high contrast between the binary elements (black vs. white) will help to differentiate the binary values under a wide range of lighting conditions. The marker is likely to be a small fraction of the total image data. It is expected that the marker will be approximately between 100×100 and 150×150 pixels. As part of the project brief the image may contain one to three markers in an image.

Although it may be possible to find the location of markers by perhaps looking for mixed dark/light regions it is essential that the guide elements be used to orient the marker. Correct orientation is required so that the data can be read in a meaningful way. The provision of the perpendicular guide bars allows the detection of any rotation of the marker. In addition, if necessary, the observed change in the 90° angle could be used to estimate the perspective of the camera view, opening up the possibility of rectifying the image. The corner guide elements can be used to accurately determine the extent of the marker. This will be particularly important when the viewing perspective distorts the shape of the marker.

This analysis has shown that the algorithm must be able to

detect and orient multiple code markers located anywhere in the image. It must be robust enough to cope with rotation, perspective, motion blur and variations in illumination and size. It has also been shown that the marker contains structure and other design aspects to help overcome these issues.

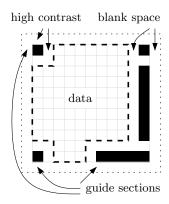


Fig. 1. Analysis of the visual code marker.

B. Dectection Algorithm

The development of the algorithm is based on the idea that the guide bar pair is the key element that can be used to identify marker candidates in an image. Once potential guide bar pairs are identified the incorrect candidates can be filtered out by looking at other characteristics of the marker. This primarily includes finding the three corner guide elements in sensible locations relative to the guide bar pair. In addition, for any marker that is sampled the correct guide 'data' can be compared against what is expected to further remove false positives.

The general idea of the algorithm is that aims to have a high detection rate and then to apply a set of loose filters to prune the number of candidates to a sensible number. Typically a number of false positives are sampled but the final verification of the guide bits is used to (hopefully) eliminate these.

The main steps of the detection algorithm are shown in figure 2 and are discussed in detail in the following sections.

1) Adpative threshold of grayscale image: The marker elements are distinguished as being either white or black squares. The image values corresponding to the elements are likely to be near the two extremes of the pixel value distributions. However, varying and inconsistent illumination across the image mean that a single global threshold is not appropriate. Instead, an adaptive threshold can used to accommodate the change in illumination by calculating a separate threshold for each pixel. Adaptive thresholding is commonly used to segment (dark) text from a (white) paper background and has much in common with this application.

The adaptive threshold algorithm was chosen to segment the guide markers in the image. A Gaussian weighted window of size 31×31 was used to calculate the mean in the region surrounding each pixel. The threshold for the center pixel was set at 5 above this mean. The window size was chosen at 31×31 after examining a number of training images. In these

images the range in width of the guide bars was between 10 and 20 pixels. A window size of 31×31 means that when the threshold is over a part of the guide bar the window extends beyond the width of the guide bar. This ensures that some of the surrounding white border is included in the mean such that the black pixels of the guide bar will be above the mean. The reverse of this applies for the white borders between the guide bars and the data/surrounding background. If the marker occupies a larger portion of the image (such that the guide bar width exceeds the window size) then the interior of the guide bar might not be complete. However, the shape of the guide bar will not be altered and it will still be recognisable.

The adaptive threshold was applied to the grayscale version of the image and results a binary image. The binary image was further processed by applying a single morphological open operation. This aids slightly in separating objects.

2) Find and classify objects: The objects in the binary image are found by first determining the contours in the image. The use of contours to define an object rather than simply connected pixel objects removes the dependence on complete objects (as mentioned above large objects may be hollow). To complete partially open contours the convex hull of each contour is taken. This new closed contour is then used to classify the object as either likely to be a guide bar, a corner guide element or something else (discarded). The classification is achieved by calculating the seven Hu moments [1] for each contour. The Hu moment set is invariant to scale, translation and, most importantly, rotation. The bounds applied to the Hu moments are fairly loose but they serve to classify the objects as being either long bars, approximately circular or something else. A minimum area requirement is also applied to eliminate objects that are too small. Before continuing to the next stage the objects are approximated by a rectangle. The rectangle encompasses the length, width and angle of rotation of the object and allows easier comparisions between objects.

The result of this stage in the algorithm is a list of possible guide bars and a list of possible corner guide squares. There is typically some overlap in these sets but this is intentionally allowed.

- 3) Pair together suitable guide bars: From the list of possible guide bars all pair combinations are examined and tested against several criteria to determine if they form a likely marker. In all of the criteria the bounds are quite relaxed in order to accommodate changes due to any perspective transformation.
 - Firstly, the relative locations are tested. The pair is deemed not part of a marker if the bar centers are closer that one half of the longest length or more than twice the longest length. This excludes most of the unnecessary pairings.
 - 2) Secondly, the ratio of the longer length to the shorter length is tested to be in the range [0.5, 2.5].
 - 3) Similarly, the width ratio is tested to be in the range [0.5 2.0].
 - 4) Finally, the difference in angles is restricted to be in the approximate range of [40, 140] degrees.

The result of this stage is a list of guide bar pairs which form the initial candidates for markers.

4) Find corner guides: The next stage of the algorithm attempts to find the three corner squares for each guide bar pair. The expected locations of the corner squares can be computed with simple trigonometry. To allow for perspective distortion an area around the calculated point is searched for the nearest square guide element. Provided the guide element is within 15 pixels of the expected location it is assigned as the corner element. This process is repeated for the three corner elements.

The result of this stage is a list of candidate markers which comprise of a pair of guide bars and zero to three corner guide squares.

- 5) Remove duplicate/incomplete markers: The final stage before the markers are sampled is to remove duplicate and incomplete markers. The following conditions are checked to determine if a marker should be discarded.
 - 1) If any of the corner elements were not found the marker is discarded.
 - 2) If any of the corner elements or guide bars are colocated.
 - 3) If the guide bar pairs are duplicates

The result of this stage is a greatly pruned list of marker candidates. The list is hoped to have a very high probability of containing the correct markers with the tradeoff of also possibly including a number of false positives.

6) Sample the grayscale image: The binary elements (both data and guide elements) are sampled from the grayscale image by averaging over a small 3×3 region for each of the 121 points on the 11×11 grid. The grid points are determined by linearly spacing (in the image) the rows along each side and the columns along the top and bottom. This method accomodates the rotation of the marker and the non-parallel edges due to perspective. It does not adjust for changes in spacing in the image due to perspective foreshortening, however this was found to not be a problem for a moderate range of perspective views. Throughout the algorithm subpixel dimensions and locations have been used. The 3×3 sample region for non-integer pixel locations is found using bi-linear interpolation.

The result of this stage is a set of arrays of sampled values (in the range [0,255]) and the coordinates of the top left corner element.

- 7) Extract data and guide parts threshold: The threshold for the array is calculated using the mean of the white guide elements and the mean of the black guide elements (assuming the array represents a marker). The threshold is set at the midpoint of these two values. This method effectively separates the two modes of the pixel value distribution and does not rely on any statistics of the distribution of the data values. The threshold is calculated separately for each marker and thus provides some degree of adaption to different light levels.
- 8) Verify the guide bits: The final check before a marker is accepted is the verification of the guide bits. The guide bits

can be used as a simple form of redundant information to verify that the marker is most likely to be real. To allow for misaligned sampling or errors in the threshold up to two errors in the guide bits are tolerated before a marker is discarded. It is not quite clear how effective this step is. From the limited testing performed it works very well but no attempt at mathematically quantifying its effect has been made.

A further step was added which was unique to the specifications of this project. The project brief stated that the number of markers in an image would be between one and three. The algorithm exploits this by ranking the markers based on the number of errors and taking up to three from those with errors less than the threshold.

C. Algorithm Implementation

The algorithm is implemented almost entirely in C and utilizes the OpenCV image processing/computer vision library. The algorithm can be compiled to run either standalone or as a MEX function that can be called from within the Matlab environment. The final two steps of the algorithm (thresholding the sampled data and verifying the guide bits) are performed in Matlab and are not included in the standalone application (but would be trivial to add). The source files include a 600 line C file and an 80 line MATLAB function file.

Figure 3 shows some of the major steps in the algorithm for one of the training images (training 9.jpg).

III. ALGORITHM PERFORMANCE

The algorithm was tested on the provided training set of 12 images plus additional images obtained independently. The training images helped to refine parameters in the algorithm and although testing was not comprehensive it provided indications of the algorithms robustness and performance.

A. Robustness

The algorithm has been designed to be invariant to rotation and scale (within limits). It has also been designed to be adaptive to changes in illumination by calculating local thresholds rather than using hard coded values. Initially it was found the algorithm had been overtrained to the first image set. By testing with other images virtually all of the parameters were relaxed somewhat. This introduced more false positives but these are easily removed in the final stages.

B. Computational Performance

Although the use of C and the OpenCV library was chosen mainly as it was a recently used and familiar environment (ee289 - Intro to Computer Vision) it potentially has advantages in speed over the equivalent implementation in Matlab. The processing time depends on the complexity of the source image but it is typically 100 ms per image. The implementation has not been optimized at all (apart from optimizations within the library) and improvements in both speed and particularly memory usage could be obtained.

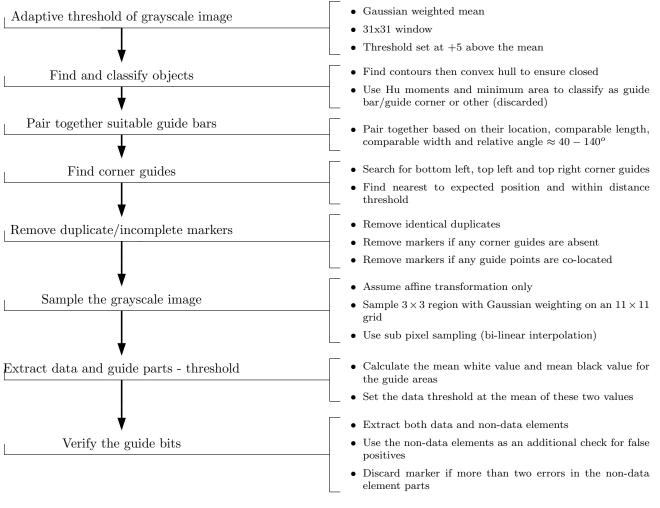


Fig. 2. Main steps of the detection algorithm.

IV. CONCLUSION

This report has detailed the development and implementation of an algorithm to detect visual code markers in images taken from cell phone cameras. The algorithm applies an adaptive threshold to segment the image into objects and background. It then classifies the objects based on their Hu moments and forms likely candidates for visual code markers. A succession of tests are then applied to filter out false positives before sampling the image to obtain the data. As a final test the 'data' in the guide sections is verified. The algorithm has been designed to be invariant to scale and rotation and to be robust against motion blur and varying illumination. Testing has been performed with two test sets, the first of 12 images and the second of 25 images. The second set was created to test the robustness of the algorithm and included markers of quite different sizes and of all rotations. The algorithm correctly identified all of the visual code markers in both sets without error. The algorithm has primarily been implemented in C and includes a wrapper to allow it to be called from Matlab. It typically processes a VGA resolution image in 100 ms.

REFERENCES

 R. C. Gonzales and R. E Woods, *Digital Image Processing*, 2nd ed. Upper Saddle River, New Jersey: Prentice Hall, 2002.

APPENDIX

To use the algorithm as a MEX function from within Matlab requires that the Matlab environment be configured in a certain way. The following two sections explain how this can be done. Alternatively, I have modified the Matlab startup scripts (.matlab7rc.sh and mexopts.sh) to automatically configure everything. This requires that Matlab be started in the same directory as the startup files (the root of the directory structure that I supply).

A. OpenCV

The OpenCV library is available at http://sourceforge.net/projects/opencvlibrary/. It can be compiled and installed by running the configure script and then make && make install. By default it will try to install to /usr/ which will require root priviledges. The default installation path can be changed by running the configure script as follows

```
./configure --PREFIX=/path/to/directory
```

To compile the source the following command will work

```
g++ -00 -I/path/to/opencv/include/opencv
-o process_img process_img.cpp
-L/path/to/opencv/lib
-lcxcore -lcv -lhighgui -lcvaux
```

The libraries are dynamic and thus the path must be known if they are not installed in a system path. The following environment variable should be set prior to executing the program both as standalone as under Matlab.

```
LD_LIBRARY_PATH=/path/to/opencv/lib
```

B. Matlab

The version of Matlab present on the SCIEN machines trys to link to a different version of the GCC library than that present on the machines (GCC 3.3 rather than 3.4.4) which will cause problems. This can be remedied by forcing the GCC 3.4.4 library to be loaded before Matlab loads its own. This can be achieved by setting and exporting the LD_PRELOAD environment variable as

```
LD_PRELOAD=/lib/libgcc_s.so.1
```

before running Matlab.

To compile the program as a MEX function under Matlab ensure that MATLAB is defined in process_img.cpp then execute the following from within Matlab.

```
mex -I/path/to/opencv/include/opencv
process_img.cpp
-L/path/to/opencv/lib
-lcv -lcvaux -lcxcore -lhighgui
```

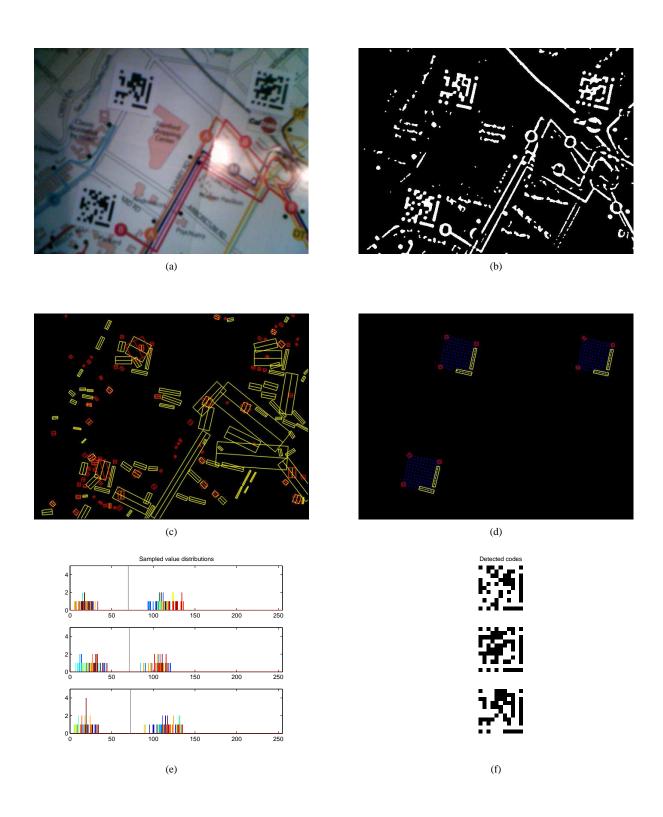


Fig. 3. The images show selected points in the sequence of operations in the algorithm