Automatic 2D Barcode Location and Recognition

By: Patrick Martin and Jason Scott

Contents

Introduction 4
Problem Statement 4
Solution 4
Method 4
Designing the Barcode 4
Locating the Barcode 5
Identifiable Features 5
Corner Detection 5
Orienting the Barcode 6
Reading the Barcode 6
Observations 6
Practical Uses 8
Augmented Reality 8
Automated Training of Neural Networks 8
Point of Sale 8
Item Tracking 8
Code Overview 9
Image Processing Library 9
Convolution Operator 9
Corner Identification 9
Midpoint Circle Algorithm 9
Barcode Identification 10
Vector Library 10
Locked Bitmap 10
Region Identification 11
Barcode Generating Program 11
Geometrical Grid Driver 11
cs370FinalProject 11
Materials 11
System Requirements 11
References 12
Appendix A: CD Layout 12

Appendix B: Program Listing 12

Introduction

Problem Statement

For many digital applications it is desirable to parse metadata from real-world images. In most cases this would be added after the fact by a human being or an application side algorithm would have to isolate and process features in the image. In many speed critical or resource starved systems, it is desirable if this metadata can be extracted using as little processing time as possible.

Solution

To address this issue, we designed a barcode system roughly based on relatively new developments in barcode design. Our goal was to generate a barcode that would provide uniquely identifiable features that could be quickly extracted and translated into a format that the application can easily read.

Method

Designing the Barcode

Our barcode design is roughly based on the DataMatrix[1] standard. To simplify the reading process we reduced the size of the barcode to a 10x10 grid with the information being stored in the middle 8x8 square. To aid in location we then constructed a rectangle around the barcode area which we use to isolate the barcode's location on screen and determine the location of its corners.

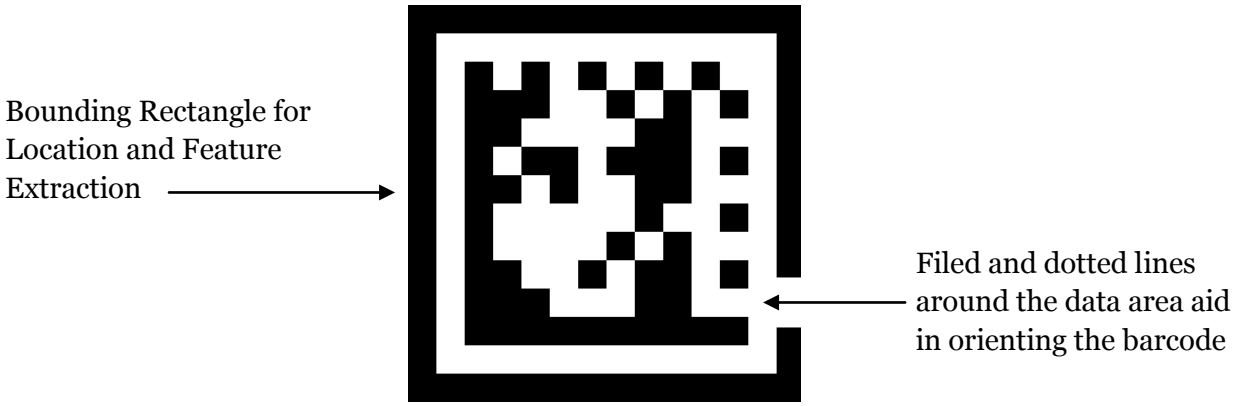


Illustration 1: Example Barcode with the message "Save Pic"

The left and bottom edges of the data area are entirely black to aid in barcode orientation. Additionally the top and right edges have an alternating black and white pattern to also aid in orientation and to help identify the columns and rows of the data area. The data itself has a

1 ISO standard, ISO/IEC16022—International Symbology Specification

single ascii byte encoded in each row, and a total of eight rows is used to represent an eight byte string. For each byte a black square represents a one while a white square represents a zero.

Locating the Barcode

Identifiable Features

To locate the barcode we attempt to locate its containing box. To do this we auto-threshold the image based on the average gray level. We then generate a region map to separate the image into individual regions.

Once we have each region isolated we examine the thinness ratio with the equation$\left(\frac{Area}{Perimete\ r^2}\right)$. Additionally we calculate the aspect ratio as $\frac{heig\,ht}{widt\,h}$ and the Euler Number as convexities – concavities. We use these three metrics as a vector $\begin{bmatrix} Thinness \\ Aspect\ Ratio \\ Euler\ Number \end{bmatrix}$ and measure the distance from our ideal values of $\begin{bmatrix} .019 \\ 1 \\ 0 \end{bmatrix}$. When multiple objects have nearly identical distances from this ideal matrix, we rely on vertical distance to locate the barcode.

Corner Detection

To read the barcode, we need to locate each of the four corners. Since the barcode is likely to be regioned into multiple regions and the alternating black and white features generate many corners, we perform our corner detection on the object we've identified as the bounding rectangle.

For corner detection we chose to use a modified version of the Features from Accelerated Segment Test[2] (FAST) corner detection algorithm. This test was chosen because it was simple to implement, it was based on the midpoint circle algorithm we designed in cs200, and it operated fairly quickly.

Our corner algorithm made use of the convolution function created for this project. We create a mask with the center cell containing a 1.0f, and each pixel on the perimeter of the circle about that center specified by a radius also containing a 1.0f. We convolve the mask with the image, only considering white pixels and counting the black pixels on the perimeter. All these values are stored in a map where each cell corresponds to a similarly numbered pixel in the black and white bitmap being convolved, and we will refer to these now as the corner heuristic.

For our purposes, we ignore all pixels that have a corner heuristic lower than 10 as they are more likely to be residual noise along the edge of the barcode, and we provide the algorithm with a radius of 3. This will reveal several groups of pixels that are mostly bunched at the corners of the bounding rectangle. If we have more than four groups, we take the four largest as these are most likely to be located at the corners. The corners we feed farther into our algorithm are determined by the center of mass for each of these corner groups.

---

2 Machine learning for high-speed corner detection - Edward Rosten and Tom Drummond - http://mi.eng.cam.ac.uk/~twd20/Research/Rosten-ECCV06.pdf

Orienting the Barcode

To correctly identify the barcode, it must be oriented correctly when passed into the parser.

The first step to getting the correct orientation is to make sure the points are in a certain order, for our purposes we chose use clockwise order. To do this we took created vectors between the first point and each other point in the list. Then we cross each pair of vectors to get new resultant vectors. We then use the diagonals of these vectors to determine if the points are already in clockwise order, if they aren't we swap them. This process is repeated until there are was run with no swaps. This leaves us with a set of points that are in one of two stats. They are either clockwise or counter clockwise. So there is one final check of the points to make sure they are in clockwise order.

Then, the barcode is preemptively parsed to gather information about the orientation. Basic error correction is then performed to determine if the white corner is in the upper right. If not, all the corners are searched for a white corner, and if one is found the points are rotated clockwise so they each rest on a corner pixel and the upper right pixel of the barcode is white. Then a final reading can be performed for parsing into the application.

Reading the Barcode

Our barcode algorithm is designed to be able to read the barcode given only the four corner of the box around the barcode. The entire code is designed to exist on a grid that is 14 units by 14 units in size, and we use a system of ratio's to determine the approximate location of each pixel.

First, the corners of the data area must be located. To do this we construct the two diagonal vectors that connect opposite points. We step $\frac{5}{28}$ of the way along this diagonal from each corner to get an approximation of the location of the data area. We then construct an 8x8 grid inside of this corner where the intersection of each horizontal and vertical line lies approximately on an individual cell in the barcode. Since the barcode should be correctly oriented at this point, this data can be extracted row by row and translated into an eight character ASCII string.

Observations

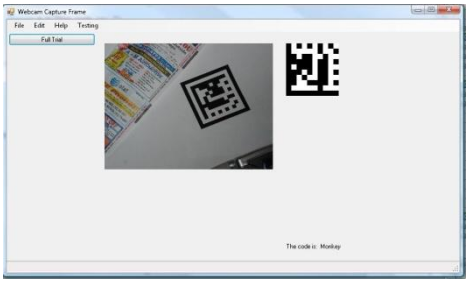
Locating the barcode proved to be a significant challenge. Initially we had relied on identifying specific colors to determine the barcode location, but we changed our method after meeting significant disapproval during the first review of our project. After briefly attempting to identify several various shapes in various orientations, we settled with the hollow rectangle surrounding the barcode. Since we wrote the entire project in c# creating our own image processing libraries, progress was fairly slow and we weren't able to generate all the feature vectors we had initially hoped, eventually settling on thinness ratio and image extents.

Identification with Ideal Conditions: 100% accuracy

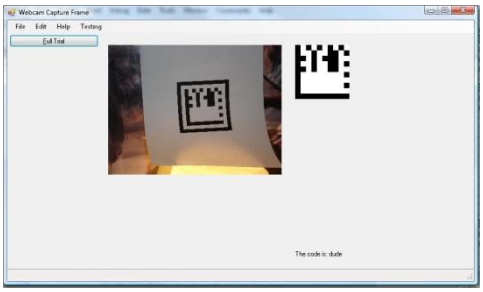
Identification with Simple Rotation: 100% accuracy

Identification with Perspective Rotation: 10% accuracy (failed to locate corners, found shape)

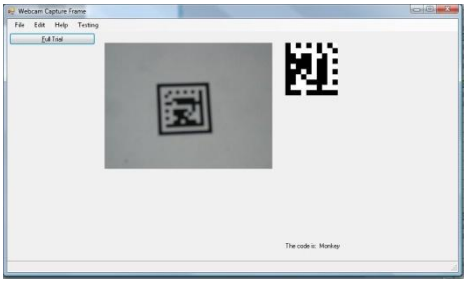
Identification with Prominent Foreground Object: 100% accuracy



Identification with Prominent Background Noise: 100% accuracy



Identification with Blurred Image: 98% accuracy (failed with a single border pixel)



Overall, the process works very well for a camera capturing the barcode as a prominent feature. Due to the low resolution all our tests are performed at (320x240), smaller barcodes will invalidate the corner test. We were also unable to correctly automatically compensate for both perspective shearing and significantly warped images.

Practical Uses

Augmented Reality

Augmented reality involves integrating actual objects with digital systems ranging from print advertisements that contain web links encoded in barcodes that you can visit via your camera phone to games and simulations that utilize actual objects you can manipulate to seed and alter a virtual simulation. Automatic camera based barcode recognition as discussed in this paper can serve to aide computer driven vision systems identify and locate real world objects utilized in augmented reality.

Automated Training of Neural Networks

Labeling objects in the training set for neural network can be a long and tedious process that could be made more efficient if automated. Normally this would be a circular issue, where it is desired to create an automated algorithm to identify objects to train an automated algorithm to identify objects. Using simple barcodes like those discussed in this paper, a simple sign could be constructed containing a barcode that describes the object. When the training function arrives at the image, it can search the image for the barcode, and use that to label the training data rather than forcing a programmer to manually do the same task.

Point of Sale

Many point of sale systems use a form of 1D barcode requiring a special reader to identify products and maintain the in-store inventory whilst only allowing one object to be "read" at a time. Using computer vision based algorithm, multiple objects may be scanned at a time as well as removing the need to purchase special scanning equipment. Additionally, a system could be devised where consumers use the cameras present on most cell phones to easily purchase items by taking a picture of the barcode and paying digitally.

Item Tracking

Parts on assembly lines as well as automatic transit systems and security systems could benefit from a camera based algorithm that quickly identifies many objects in real time. Normal

image processing algorithms may be too slow or cumbersome to design and train to perform the job efficiently. Objects and personnel identified via a visual barcode system read via camera could allow fast batch recognition and tracking of individual objects

Code Overview

Image Processing Library

Convolution Operator

The convolution operator consists of three parts. The actual ConvolutionOperator class is constructed via a ConvolutionMask and a ConvolutionFunction. The ConvolutionOperator will then shift the ConvolutionMask along the image, multiplying each pixel in the image with the corresponding cell in the matrix. This will create a resultant matrix that is passed into a convolution function that will perform any desired operations with this matrix.

The convolution mask is a fairly simple class. It contains a width, a height, and and an array of floats that represent the cell indexes. When the convolution operator moves it along a bitmap, an identically sized mask will be created that has each cell multiplied with the pixel under it.

The convolution operator is actually an interface that specifies a single required function, Convolve. This way specialized classes can be constructed that operate on the data in the matrix in unique way. Included in our project is a partially working mean filter, a median filter, a corner detector, and a Euler number class that generates the Euler number as well as summing up the concavities, convexities, and calculating the perimeter.

Corner Identification

For corner identification, as mentioned above, we use the Features from Accelerated Segment Test detection algorithm. For our project it is implemented via a specialized convolution mask and convolution function defined in the CornerDetector class. Calling the constructor with a specified bitmap and radius (we use a radius of 3 as specified in the paper by Edward Rosten and Tom Drummond) will cause the corner detector to create a mask with a 1 in the center and in the circle generated via the midpoint circle algorithm.

This fills a 2D matrix of floats with a number specifying how likely it is that each pixel is a float as described above. Since the image the mask is applied to is entirely black and white, we simply sum up the black pixels on the circle's perimeter rather than calculating the energy gradient as specified in the linked paper. Although this gives us many false positives as to which pixels are edge pixels, we can easily segment the prospective pixels and find the four largest contiguous areas. These areas are, as far as our testing indicates, the most likely candidates for the corners of the barcode.

Midpoint Circle Algorithm

The midpoint circle algorithm is able to quickly draw a circle by drawing a single octant of the circle and reflecting the changes around the circle's center point. Additionally it speeds computations by storing values as integers rather than floats and removing any square root operations in drawing the circle.

The algorithm starts the x coordinate at radius away from the center, and the y coordinate equal to the center point. It then loops until x is greater than or equal to y, indicating that it has completed the first octant. For this algorithm, it is assumed that y is strictly increasing, and it will be updated every iteration. To determine whether or not to decrease the x coordinate, a midpoint is calculated between the two possible pixels that x could be placed in. Then using a rearrangement of the circle algorithm, $x^2 + y^2 = r^2$ is written as $x^2 + y^2 - r^2 = 0$, with x being the midpoint between the two possible pixels to place x. If the equation is greater than or equal to zero, then the x coordinate is decreased in order to continue approximating the circle.

Barcode Identification

Barcode identification is primarily concerned with reading the data in the barcode assuming that it has been located on the screen. This consists of the classes GeometricalSquare and GeometricalGrid to help isolate the barcode, and the class BarcodeParser to translate the barcode into human readable text.

GeometricalSquare takes four clockwise oriented points indicating the four corners of the barcode region (including the bounding rectangle). It then approximates the position of the data section of the barcode (indicated by the GeometricalGrid) by stepping 2.5 units along a 14 unit long vector diagonally connecting opposite points as described above.

These four interior points are then passed into GeometricalGrid. GeometricalGrid will perform several operations. Its primary operation is to split its image area into a 10x10 grid. This grid indicates the sample points to test for the barcode, which is converted into a 10x10 bitmap for further parsing. Additionally, GeometricalGrid contains an AutoCorrect() function call that will rotate its interior points to place the empty pixel in the upper right as per our barcode specifications.

BarcodeParser is the only component of our project written in c++ due to speed concerns and our desire to directly manipulate memory. BarcodeParser will take either a 10x10 array of boolean values or a 10x10 Bitmap indicating the barcode, and translate the center eight pixels into an ascii string. This class is compiled into its own dll, and can be easily integrated into any managed project despite being written in c++.

Vector Library

The vector library was originally written for a Mat300 assignment, but served as a great aid for this project. It has the ability to represent any sized vector, provides conversion operators between Vectors and PointF's, and provides all the basic vector operations such as addition, subtraction, dot product, normalization, and scalar multiplication and division.

Locked Bitmap

The LockedBitmap class provides a simple interface for locking and unlocking .NET Bitmap objects. Among provided features are direct access to the pixel buffer, buffer swapping, auto gray scaling, and auto thresholding. This class also automatically compensates for stride and converts images to 24Bit RGB format for easier reading and writing.

Region Identification

The region identification was created for the purpose of segmenting the image into distinct regions to be used at multiple steps within the barcode reading process. Region identification is done by looping through all the pixels in the image and marking each pixel with a color value below a given threshold as a region. When a pixel is marked as a region pixel, it's neighborhood is checked for any other regions. If one of the neighboring pixels is already regioned, then this pixel is set to be part of that region. If more that one neighboring pixel has already been regioned then all the corresponding regions are concatenated into one region.

Barcode Generating Program

The barcode generator was written to facilitate the generation of barcodes for testing with our project. It will convert any input ascii string up to eight characters long into a barcode, apply a scale that avoid any blurring or loss of detail, and saves into a bitmap file that can be later printed or utilized in another document.

Geometrical Grid Driver

The Geometrical Grid Driver was designed primarily to test the GeometricalGrid and GeometricalScale classes, and as such is rather crude. One may open any image file supported by DirectX, and then drag the corners of the bounding square to the corners of the barcode area. Lines indicate the individual sections of the barcode, and their intersection indicates the sample locations for generating a barcode. The current barcode image is displayed on the left.

As stated above, this is primarily designed for testing the actual sampling algorithm. It will crash if you drag the grid outside of the texture area, and is primarily of use for testing the accuracy of the barcode reader in various conditions without running the barcode locating code.

cs370FinalProject

This is the main application driver for the barcode reader. By opening an image with a barcode in it and pressing the "Full Trial" button, one may watch the application locate and read the barcode. Alternatively, it is possible to run each step individually using the "Test" menu.

Materials

For development and testing we used a Sony Cyber-shot camera, Model No. DSC-P100, to capture images.

System Requirements

The main project requires Visual Studio .NET 2005 with C# and .NET 2.0 support to build.

The GeometricalGridTestDriver project requires Managed DirectX to successfully build and was developed against the March 2008 SDK, although any DirectX 9.0c SDK (including those present on DigiPen systems) should work if present in your visual studio library path.

References

[1] Edward Rosten and Tom Drummond. *Machine learning for high-speed corner detection*. University of Cambridge. <http://mi.eng.cam.ac.uk/~twd20/Research/Rosten-ECCV06.pdf>

[2] Eisaku Ohbuchi, Hiroshi Hanaizumi, Lim Ah Hock. *Barcode Readers using the Camera Device in Mobile Phones*. Proceedings of the 2004 International Conference on Cyberworlds. Volume , Issue , 18-20 Nov. 2004 Page(s): 260 – 265

[3] Ouaviani, E.; Pavan, A.; Bottazzi, M.; Brunelli, E.; Caselli, F.; Guerrero, M. *A common image processing framework for 2D barcode reading*. Image Processing and Its Applications, 1999. Seventh International Conference on (Conf. Publ. No. 465). Volume 2, Issue , 1999 Page(s):652 - 655 vol.2

[4] Jun Rekimoto and Yuji Ayatsuka. *CyberCode: Designing Augmented Reality Environments with Visual Tags*. Sony Computer Science Laboratories, Inc. <http://www.sonycsl.co.jp/person/rekimoto/papers/dare2000.pdf>

Appendix A: CD Layout

/src

- Contains all project files and sourcecode

/bin

- Contains all compiled drivers and libraries

/img

- Contains sample images

/doc

- Contains documentation

Appendix B: Program Listing

- BarcodeGenerator
  - Generate and save barcodes
- ConvolutionTest
  - Several tests involving the convolution operator
- GeometricalGridTestDriver
  - A testing ground for barcode extraction. Only performs extraction with manually placed points
- Cs370FinalProject
  - Actual project that identifies and extracts the barcode automatically