An Application Program That Interprets Code39 Barcode Images on an iPhone

David Hergert, Ph.D.
hergerd@muohio.edu
Miami University-Hamilton
1601 University Blvd.
Hamilton Ohio 45011

Abstract:

There has been a lot of interest in Smartphone technology over the last few years. Many of these phones are capable of email, internet access, and have a built in camera. Perhaps no Smartphone has generated as much interest as the iPhone. One of the features of the iPhone is that it can be programmed in Objective C using Xcode (the standard programming interface for a MAC).

This paper describes an application of an iPhone that faculty and senior design students in the TAC/ABET accredited B.S. Electromechanical Engineering Technology at Miami University are working on. An iPhone application was written in Objective C that allows the user to take a picture of a bar code displayed on a computer screen using the built in iPhone camera. The software processes the image and determines the corresponding code39 characters. Students are currently working on transmitting the barcode data to a remote data terminal. This system would have many uses for applications that require remote data acquisition, time/date stamping, and lab work verification. An example would be collecting inventory information from products that are separated by large distances from a data collection device.

Keywords: cell phone code 39 barcode scanning

Cell Phone Barcode Scanning

Cell phone barcode scanning has tremendous potential in the marketplace. Companies working on this technology first have a user take a picture with a cell phone camera. Software on the phone then takes users to the web address of a company (or competing companies) for price checks and comparisons. As an example, Scanbuy Shopper uses barcodes and Internet price comparison search engines to tell how much an item is going for online. A picture taken with a cell phone is processed using software on the cell phone. As another example, a recent New York Times article states "House hunters, driving past a for-sale sign, stop and point their cellphone at the sign. With a click, their cellphone screen displays the asking price, the number of bedrooms and baths and lots of other details about the house." [1]

The interest the author has in this technology has more to do with reading laboratory data collection and verification in a distance education program. The B.S. Electromechanical Engineering Technology degree program is offered by distance to eleven community colleges throughout the State of Ohio. A new Industrial Automation course in the program requires students to use lab equipment located at the community college for part of the semester. This has been a problem since some of the campuses have the lab equipment stored in rooms with no internet access or interactive video connection (to date there are three campuses with this situation). This makes it very difficult for the instructor to know if the student has done the lab. Also student have to manually record data collected from the lab.

One solution being considered is for a community college technician to be available to assist the student if they have trouble with the equipment. When the student finishes a lab, the technician opens a binder to a page with a barcode associated with the lab. This barcode would be created by the department and would specifically encode information about the lab and location.

The student would then take a picture of the barcode with their cell phone and use software (described in this paper) to decode the barcode and send the data (along with the date, time and student's name) to a Miami server. This way the instructor would know when the lab has been completed by the student. Once the barcode information is sent, the software would erase the photo on the phone. This way the student could not send the picture to other students.

If the student is in a lab where no 3G connection is available, the student could go somewhere where there is reception and have the software process the photo.

Besides lab work verification, the barcode reader in the phone could also perform data acquisition in a remote (no internet) setting.

Remote Data Acquisition

Remote data acquisition has also become a popular topic in the last few years. There are many methods of implementing this, including data acquisition cards connected to a computer on a network, microcontroller cards such as the TINI that use a TCP/IP stack that are tethered to a PC through a cable, radio or telephone modem links, and Bluetooth connections.

All of the methods described above at some point require an internet connection to send the data out to a server. Many times a machine or instrument is located in a place where there is no internet connection (including wireless and local area connections). For these situations, a 3G network could be used to transmit data from a sensor to a server. Since the iPhone is a 3G device, it would be ideal for remote data logging in situations where an internet connection is not available. Getting data from an outside source into an iPhone is somewhat limited to:

- Receiving data from an internet connection.
- Receiving data from a Bluetooth connected server.
- Receiving data via a coded sound using the iPhone's microphone.
- Receiving data from the camera (this is the method chosen here).

Perhaps the easiest method to record data from a camera is through the use of bar codes. A computer uses Labview to collect data from an instrument or manufacturing process and encode the data in a barcode on the screen (see Figure 1). The IPhone user then takes a picture of the barcodes on the screen and an app converts them to the associated symbols, letters, and numbers.

Finally the data is sent over a 3G network to a remote server. A diagram of this process is shown in Figure 2.

This paper focuses on converting the barcode to data (perhaps the most difficult part of the Figure 2 diagram). While the iPhone has a somewhat low resolution camera (2M pixels on the 3G and 3M pixels on the 3GS) and no flash, tests have shown that code 39 barcode can still be converted with a high degree of accuracy, as long as the bar code is large enough. Even pictures with multiple bar codes can be converted to their associated data. Figure 3 is a photo taken with an iPhone of a computer screen showing four barcodes simulating data received from a remote location.

The upper three barcodes first contains a letter representing a process measurement such as temperature, pressure, or flow. Next the associated reading from the instrument is shown, accurate to one decimal place. The barcode on the bottom of the screen shows a fictitious date. Note all barcodes contain six characters. For Figure 3, the four process readings would be Temperature 100.6, Pressure 48.5, Flow 231.4, and a date of 12/23/09. The applet written for this project requires these steps:

- Take a picture of the barcode.
- Load the applet.
- Once the image appears on the screen, touch the desired barcode.
- A red line is drawn from left to right at the current row. Once the user is satisfied that the red line goes through the entire barcode, she or he presses a button on the screen. The applet will then process the barcode according to the six machine vision principles defined by Fu, Gonzalez, and Lee. [2]

Code 39 Bar Codes

The code 39 bar codes used in this project are often found in manufacturing and process industries. Groover [3] provides a description of the associated letters, numbers, and symbols for standard code 39. Some of the basic rules for code 39 barcodes are:

- All code 39 barcodes start and end with a * character.
- All code 39 barcodes start and end with a black line.
- Black and white lines alternate. A wide line (white or black) represents a 1. A thin line represents a 0.
- A collection of 9 binary numbers form a character.

- There are exactly 3 1s in every character (this is how code 39 got its name, 3 of the 9 bars are 1's).

Figure 4 shows a table for simple code39 barcodes. For example a reading of *P048.5* reading would be converted as follows:

* 010010100	**P** 001010010	**0** 000110100	**4** 000110001	**8** 100100100
. 110000100	**5** 100110000	* 010010100		

Machine Vision Principles

Fu, Gonzalez, and Lee describe six principles of machine vision, namely sensing, preprocessing, segmentation, description, recognition, and interpretation. This project involved all but interpretation. <u>Sensing</u> is associated with obtaining an image. In our case this would be taking a picture with an iPhone. Next is <u>preprocessing</u>. This is handled by first copying the photo to an image space. Preprocessing performs noise reduction by setting a threshold between dark and light pixels. Pixels with an RGB value below the threshold are set to 0 (black) and pixels above the threshold to 1 (white). <u>Segmentation</u> refers to partitioning an image into identifiable objects. This is handled by searching for black and white bars. <u>Description</u> means differentiating between types of objects. In this application, differentiation computes wide and thin black and white bars. <u>Recognition</u> converts the wide and thin bars to a stream of 1s and 0s in groups of nine bits and performs a pattern look up to convert the binary stream to a text message. The process is described in more detail below.

Mapping the Photo

Once the image is received from the camera, the individual pixels must be converted to a barcode. A method to do this is described in by Lamarche [4]. The process begins with reading the pixel from a current location. Assuming a user touches the screen on a barcode, the current location of the touch can be found from in Objective C by:

```
CGPoint centerPoint = [touch locationInView:[self view]];

                float x=centerPoint.x;

                float y=centerPoint.y;
```

Next the photo needs to be reflected to an image space. The code below shows how to do this, assuming imagData is the image space.

```
CGImageRef imageRef = [view1.image CGImage];

    NSUInteger width = CGImageGetWidth(imageRef);

    NSUInteger height = CGImageGetHeight(imageRef);
```

```
        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

        unsigned char *imagData = malloc(height * width * 4);

        NSUInteger bytesPerPixel = 4;

        NSUInteger bytesPerRow = bytesPerPixel * width;

        NSUInteger bitsPerComponent = 8;

        CGContextRef context = CGBitmapContextCreate(imagData, width, height,
bitsPerComponent,

        bytesPerRow, colorSpace, kCGImageAlphaPremultipliedLast |
kCGBitmapByteOrder32Big);

        CGColorSpaceRelease(colorSpace);

        CGContextDrawImage(context, CGRectMake(0, 0, width, height), imageRef);
```

In the above code, CGImageRef imageRef defines the content of view1 reflected in imagData, CGImageGetWidth and CGImageGetHeight get the image width and height respectively. The color space for the reflected image is then defined as a .PNG format RGBA8888 (4 bytes per pixel). RGBA8888 means the red, green, blue, and the alpha intensity value are all 8 bits (1 byte). Therefore there are 4 bytes per pixel. This defines the context. Finally the image is drawn in the context space with the indicated RGB format.

Image Exposure Setting

Once the image is defined in imagData, it can be processed. The image is first converted to black and white. This is done by setting an image exposure. The image exposure is actually a threshold indicator used to determine the cutoff between black and white bars. RGB values and the threshold are stored in a range of 0 to 1. RGB values under the threshold are recorded as black, and RGB values over the threshold are recorded as white. The image exposure threshold indicator is initially set to 0 (all black) and is later incremented until a successful reading is taken. Successful readings are determined by a * character at the beginning and end of the barcodes and a total of eight characters read.

Finding the Start of the Barcode

Figure 5 is a flow chart depicting the process used to find the start of the barcode. After the user touches a barcode, row and column pointers reference the position of the touch. The row defined by the floating point variable centerPoint.y . Another pointer representing the current column starts at the position recorded in centerPoint.x. The algorithm continually decrements centerPoint.x left until no black bars are found. This marks the beginning of the barcode. A similar method is used to find the top and bottom of the barcode using centerPoint.y.

Processing the Barcode

Figure 6 is a flow chart depicting the process used to find the start of the barcode. All code39 barcodes start with a * character. This character is represented in binary as 010010100. Because of this, the barcode will always start with a 0 then a 1. The centerPoint.y axis pointer is set to the top of the barcode. The centerPoint.x axis pointer is incremented until the end of the first black bar is reached. Because the barcode starts with 01, the first black bar is a 0 (meaning a smaller width bar). Next the adjacent white bar is read. This represents a wide bar (or a 1). The two bars are averaged so that future bars can be determined by being larger or smaller than the average between the two widths. Bars are read from left to right until the last bar is reached. The 0 and 1 values are stored in an integer array.

Converting the Barcode to Characters

Figure 7 is a flow chart depicting the process used to find the start of the barcode. A table lookup is performed to compare each set of 9 bits to its associated character. The result of the comparison is sent to an output string. If a reading produces no associated character, the Y axis pointer is incremented to the next pixel down in the barcode, the X pointer is set to the start of the barcode, and the process is repeated. The image exposure threshold varies from 0 to 1 for each row. If the bottom of the barcode is reached and no successful reading is taken, the output string receives an “Error”.

Accuracy of Readings

The iPhone app written for this paper proved to be highly accurate for this application. It was able to successfully read over 20 photos taken with the iPhone camera in a variety of lighting conditions. There are two reasons for the success. First the computer screen provides a light source to give a somewhat even light intensity. Second each barcode contains six characters (eight including the * characters). Therefore a check could be made to be sure all characters were converted. Figure 8 shows a series of iPhone readings. The first is an intentional error where a touch was made outside the barcode. The remainder are from the image in Figure 3. The readings were obtained from taking pictures of a computer screen.

When photos were taken of barcodes printed on paper, at times less than desirable results were achieved. The camera on the iPhone is a fairly low resolution (2M pixels on the 3G and 3M pixels on the 3GS), but similar results were obtained from a 6M pixel Canon camera. The problem lies in a combination of the fact that JPG photo images tend to have a gradient at sharp edges (see Figure 9) and the iPhone camera does not focus well at distances less than one foot. However medium size barcodes could be read with a high degree of accuracy under good lighting conditions, and large size barcodes could even be successfully converted even under shadowy conditions. The code used in this paper can be found at:

www.users.muohio.edu/hergerd/iPhone

Future Work

Students in the Senior Design course are currently writing code to route the data from the iPhone to a server. This will enable the instructor to see live data from student lab experiments.

Also students are working on the image exposure algorithms to improve character recognition for medium size barcodes in dimly lit areas. This is important for the lab verification process where a binder containing barcodes will be used instead of a computer screen. There is no guarantee what lighting conditions will exist in the lab. The code will use an image gradient rather than a specific threshold to separate dark and light pixels.

A local drug manufacturer has expressed in this project to read code 39 inventory barcodes in clean rooms. This will be a future senior design project.

References:

Story, Louise. "New Bar Codes Can Talk With Your Cellphone." New York Times 1 Apr. 2007, natl. ed. : B1+.

Fu, Gonzalez, and Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw Hill, 1987.

Groover, Automation, Production Systems, and Computer-Integrated Manufacturing, Prentice Hall, Second Edition, 2001.

Mark, Lamarche, Beginning iPhone Development: Exploring the iPhone SDK. Apress, 2009.

Tables and Figures

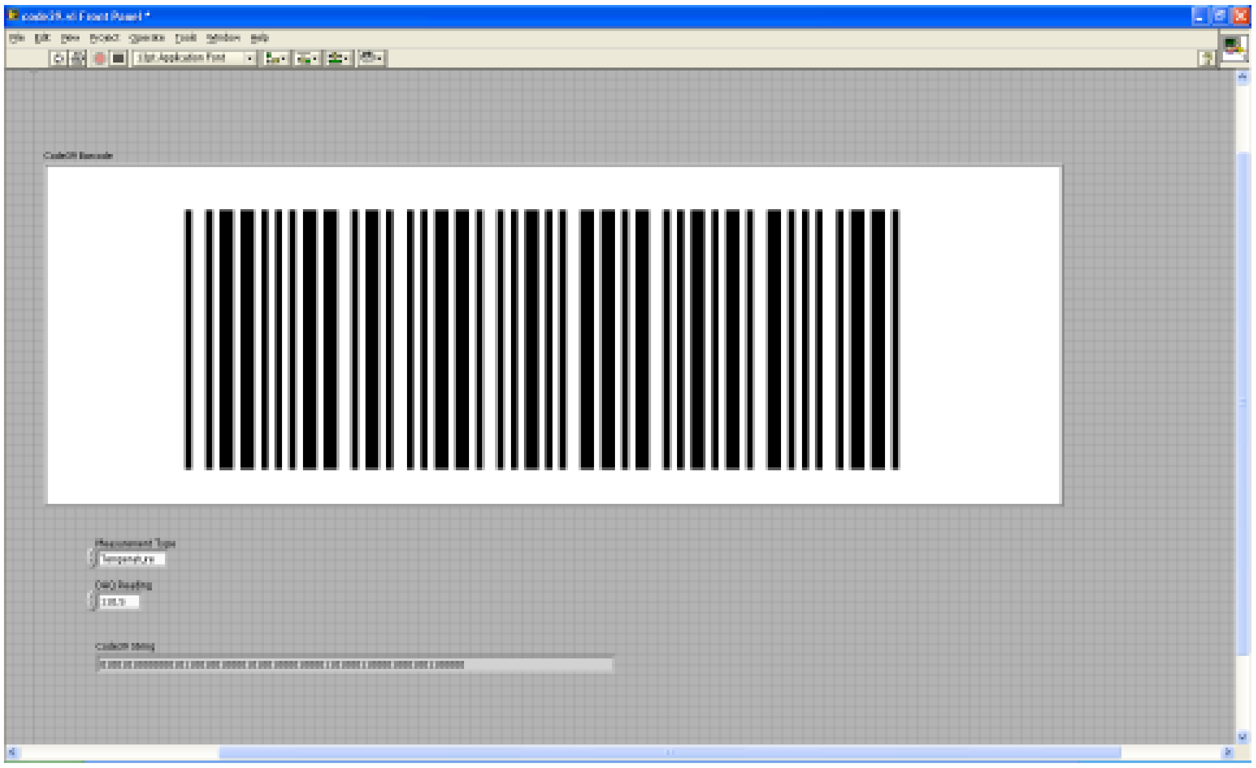

Figure 1. Labview Barcode Generator

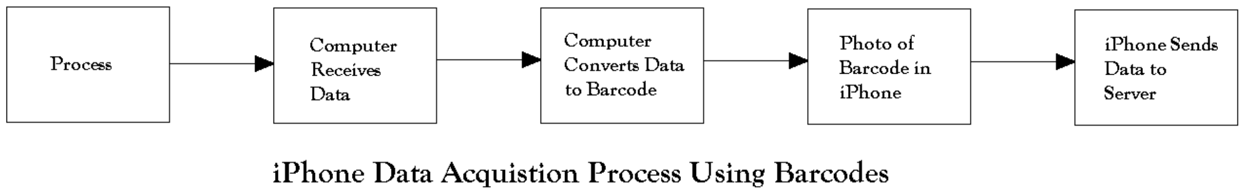
Process
Computer Receives Data
Computer Converts Data to Barcode
Photo of Barcode in iPhone
iPhone Sends Data to Server
iPhone Data Acquistion Process Using Barcodes

Figure 2. Data Acquisition Using Barcodes

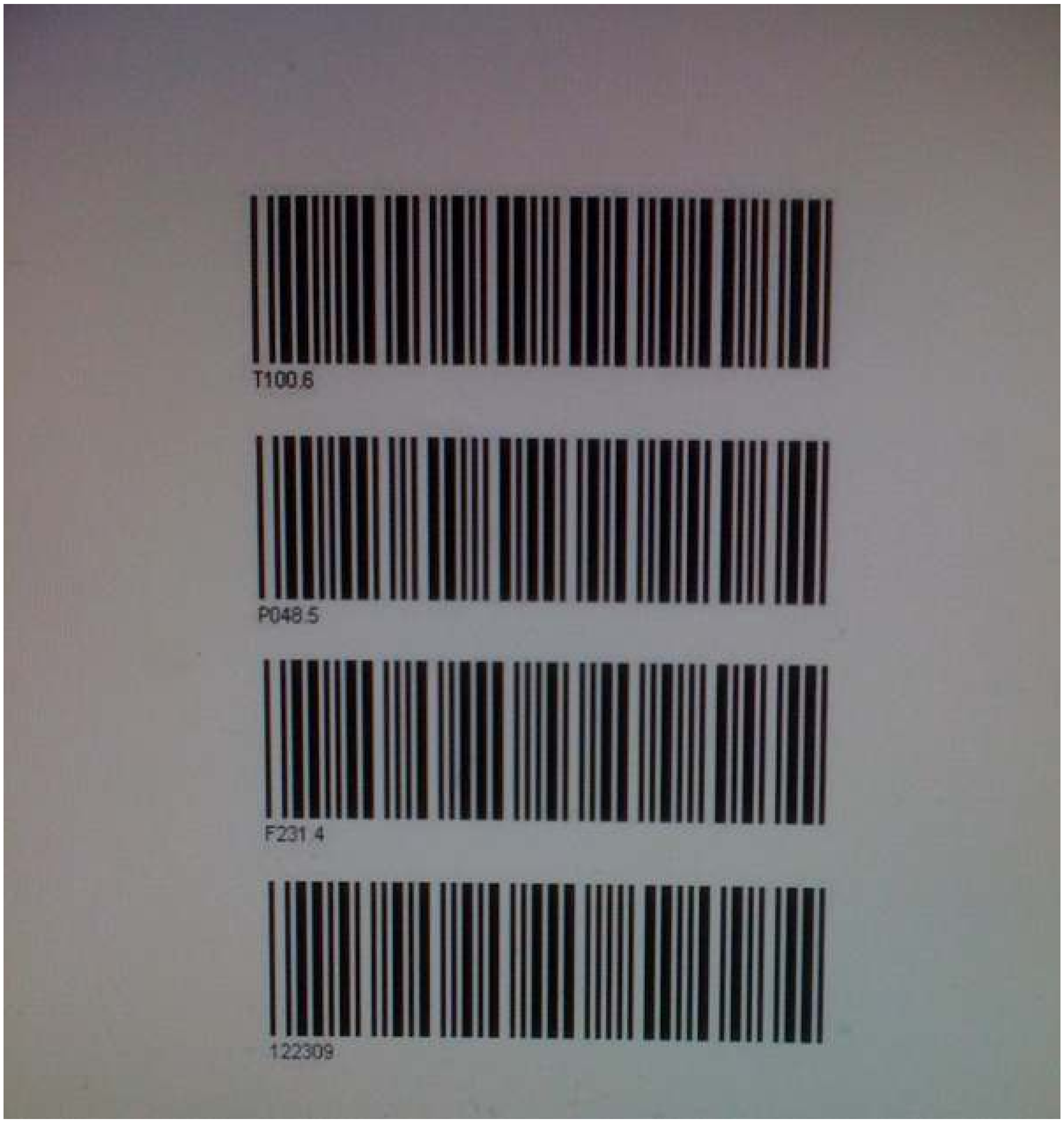

Figure 3. iPhone Photo of Computer Screen

"A"	100001001	"B"	001001001	"C"	101001000	"D"	000011001
"E"	100011000	"F"	001011000	"G"	000001101	"H"	100001100
"J"	000011100	"K"	100000011	"I"	001001100	"L"	001000011
"M"	101000010	"N"	000010011	"O"	100010010	"P"	001010010
"Q"	000000111	"R"	100000110	"S"	001000110	"T"	000010110
"U"	110000001	"V"	011000001	"W"	111000000	"X"	010010001
"Y"	110100000	"Z"	011010000	"0"	000110100	"1"	100100001
"2"	001100001	"3"	101100000	"4"	000110001	"5"	100110000
"6"	001110000	"7"	000100101	"8"	100100100	"9"	001100100
" "	011000100	"-"	010000101	"."	110000100	"*"	010010100

Figure 4. Code 39 Barcode Table

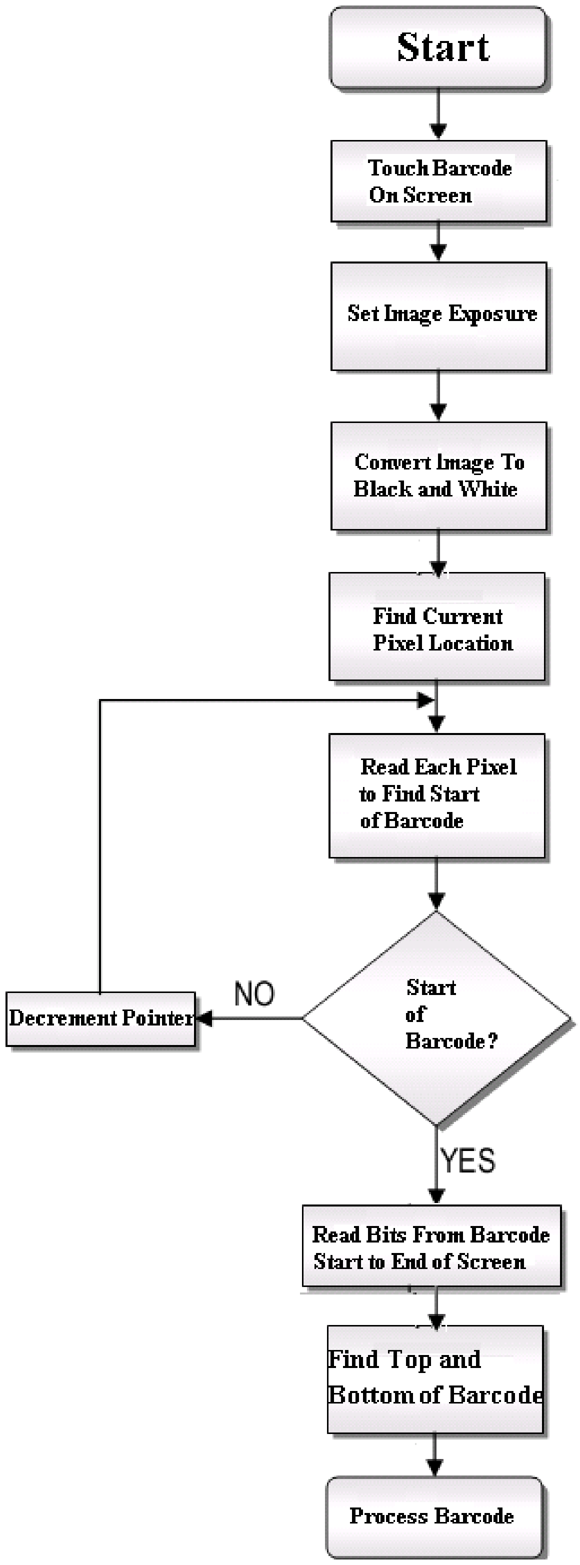
Start
Touch Barcode On Screen
Set Image Exposure
Convert Image To Black and White
Find Current Pixel Location
Read Each Pixel to Find Start of Barcode
Start of Barcode?
NO
Decrement Pointer
YES
Read Bits From Barcode Start to End of Screen
Find Top and Bottom of Barcode
Process Barcode

Figure 5. Flow Chart To Find the Start of the Barcode

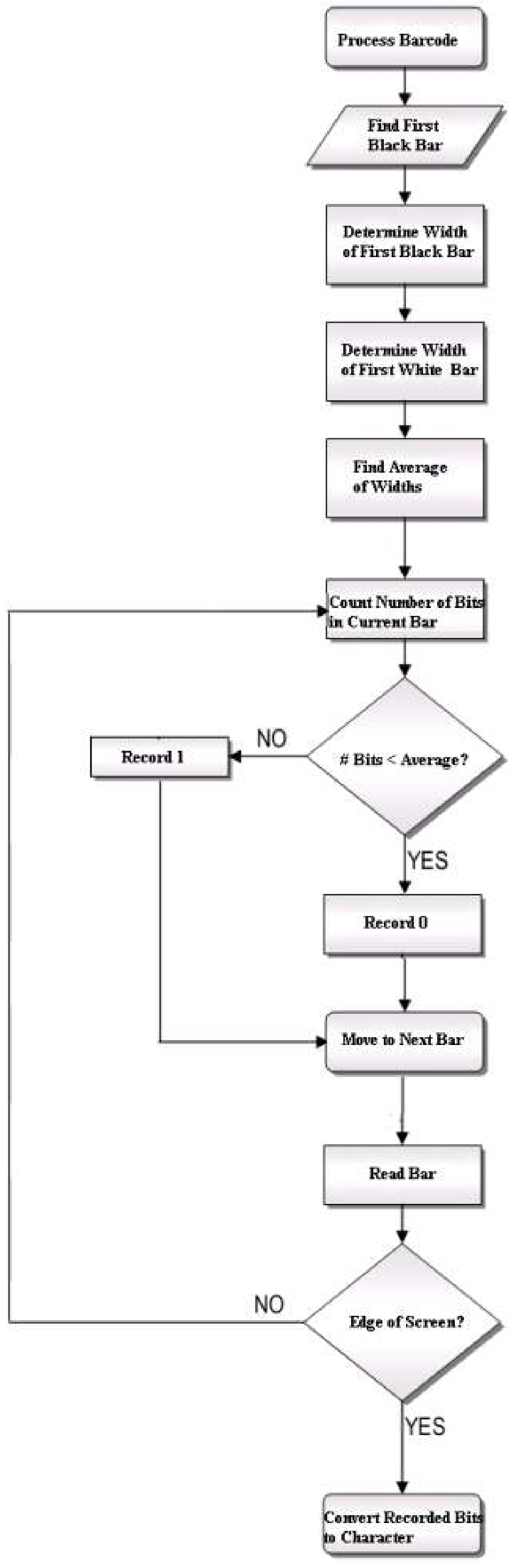

Figure 6. Flow Chart For Processing The Barcode

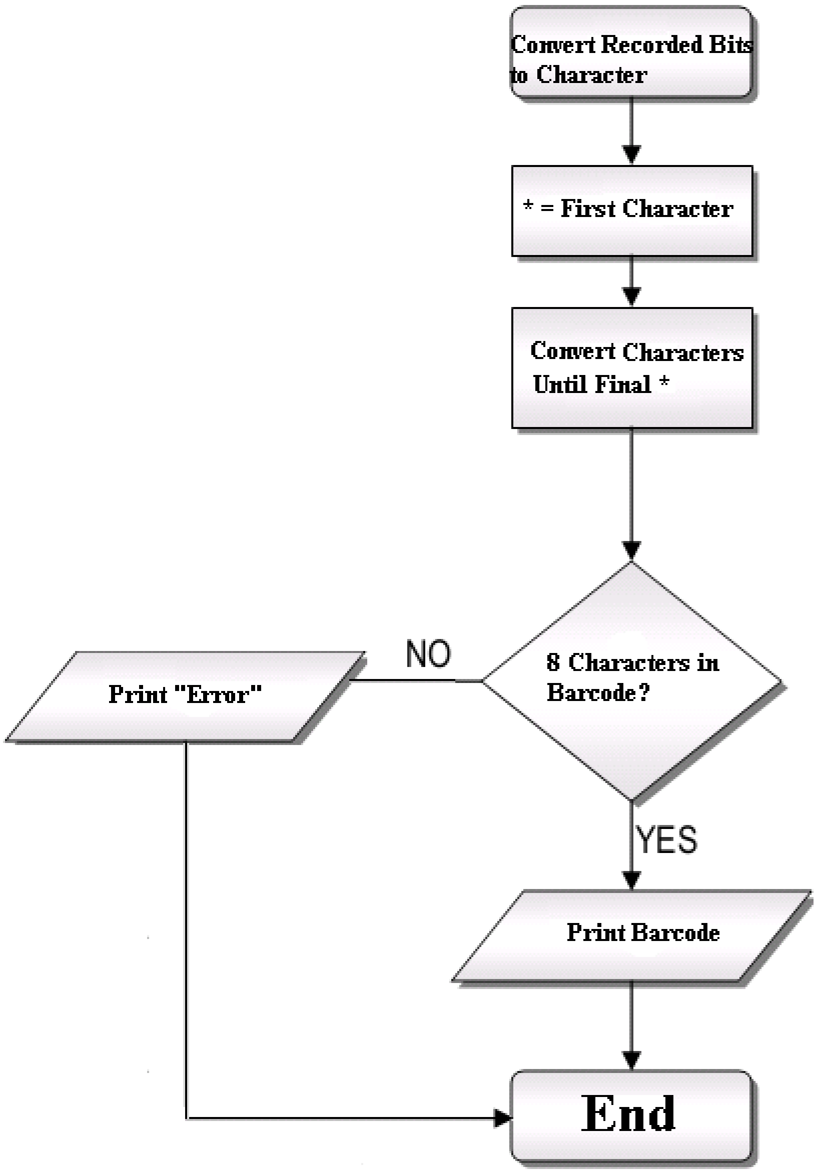

Figure 7. Flow Chart for Converting the Barcode to Characters

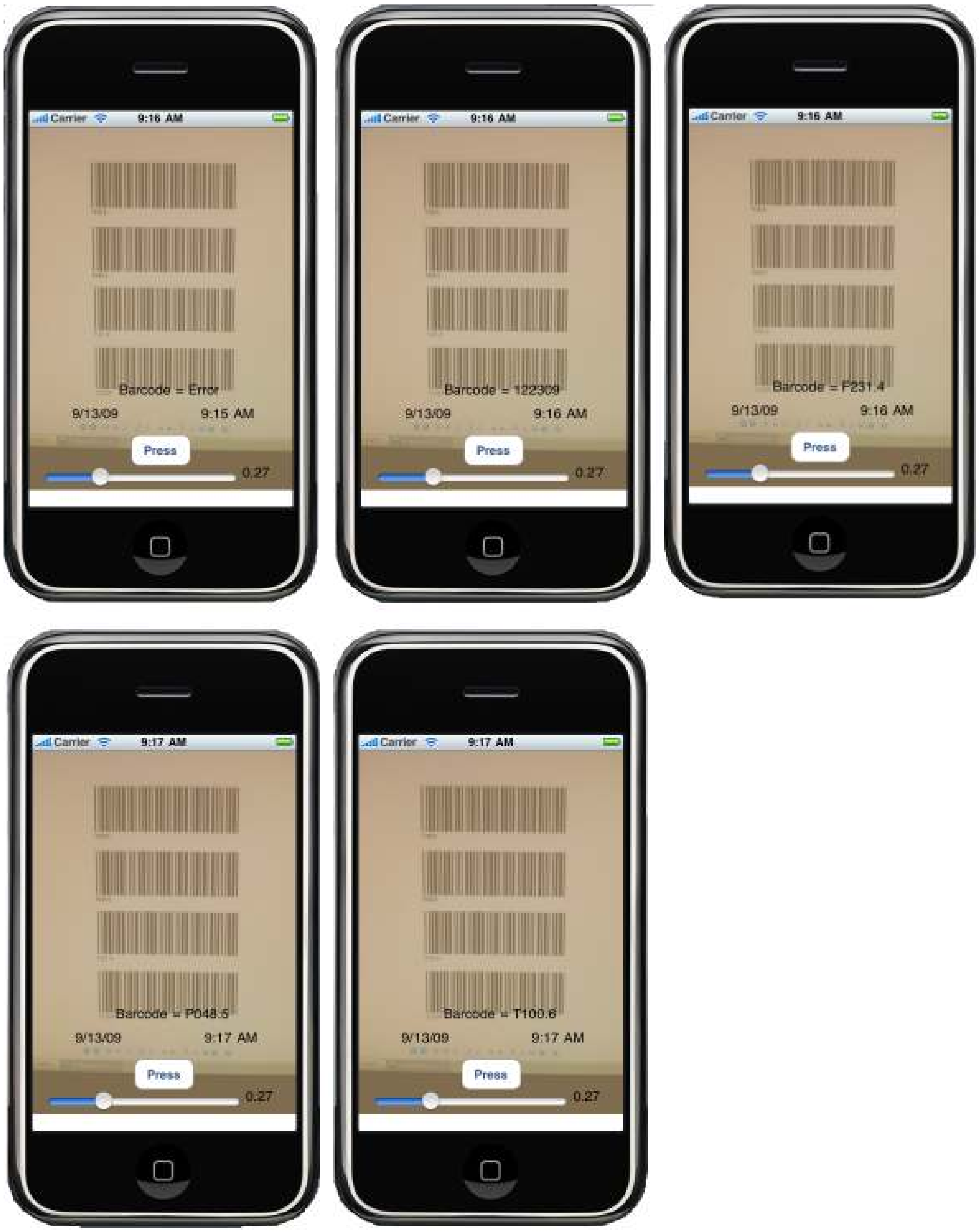

Figure 8. iPhone Images.

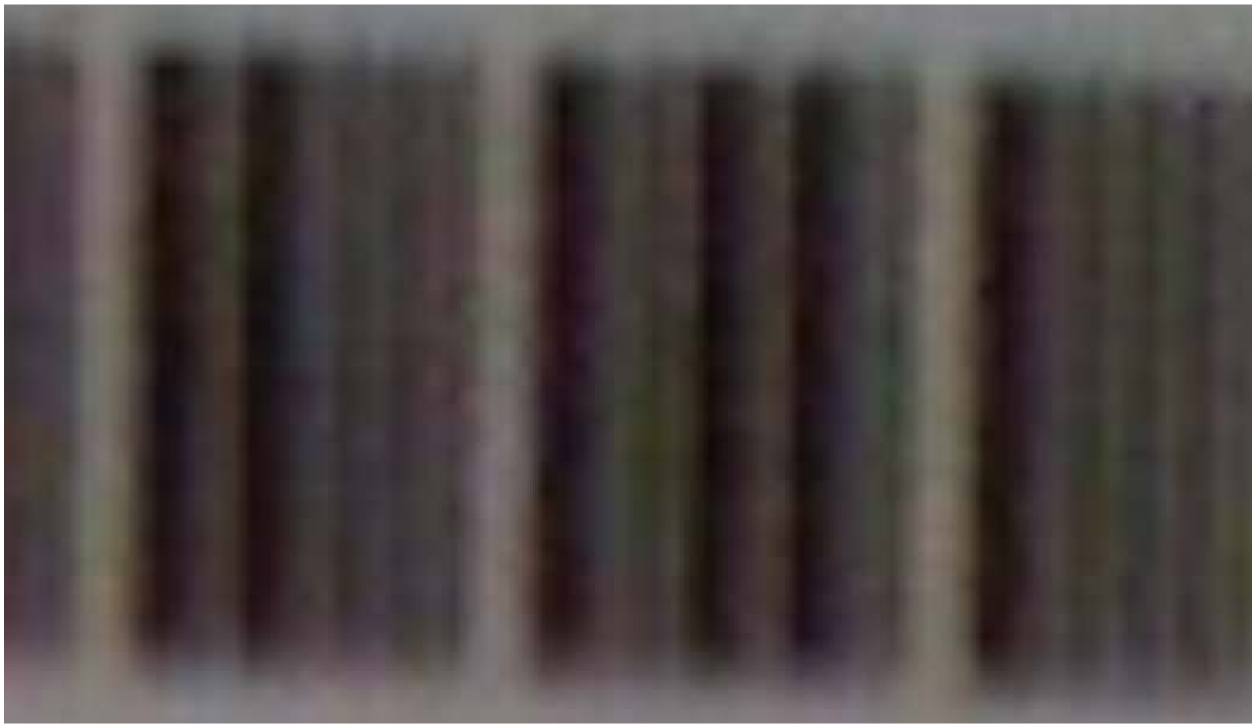

Figure 9. Photo of 1.25*.25 inch Barcode on Paper Taken in Dim Light.