LCD/LED DIGIT RECOGNITION BY IPHONE

by

Xian Li, B.S.E.E.

A Thesis

In

ELECTRICAL ENGINEERING

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

Approved

Dr. Brian Nutter
Chair of Committee

Dr. Sunanda Mitra

Peggy Miller
Dean of the Graduate School

May, 2011

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Dr. Brian Nutter. I would surely not be where I am today without his assistance and guidance. I would also like to thank Dr. Sunanda Mitra for being on my thesis committee. Additionally, to the graduate students of the Computer Vision and Image Analysis Laboratory (CVIAL), I am grateful for the assistance and support that they provided. Finally, I would like to give my special thanks to my parents, whose support and encouragement enabled me to complete this work.

TABLE OF CONTENTS

ACKNOWLEDGMENTS ii

ABSTRACT iv

LIST OF FIGURES v

I INTRODUCTION 1

OVERVIEW 1

Motivation 1

CURRENT CHARACTER RECOGNITION APPS 2

PROPOSED APPROACH 2

II APPLICATION OVERVIEW 5

BACKGROUND OF IOS 5

Software Development Kit 6

Cross-compiling 7

IMAGE PREPROCESSING 8

OpenCV Overview 8

Locating the Device Screen 9

Canny Edge Detection 10

Finding Contours 12

OPTICAL CHARACTER RECOGNITION 19

Tesseract Overview 19

Word Recognition 19

Training Tesseract 21

III RESULTS 24

ERROR ANALYSIS 28

IV CONCLUSIONS AND FUTURE WORK 32

BIBLIOGRAPHY 33

ABSTRACT

Home medical devices with LCD or LED screens are quite common. While they offer many conveniences, the data is usually not retained in any medical database. In this thesis, an iPhone^{TM} application will be introduced that recognizes the digits on LCD or LED screens. The user takes a picture of the desired LCD or LED screen, and the app will convert the image to text in seconds. The app also has an email feature that allows the user to send the picture and the text conveniently to a secure medical logging database. A contour finding algorithm is illustrated in this application for image preprocessing. This algorithm is relatively faster and more efficient than typical image preprocessing techniques. The Tesseract optical character recognition engine will be implemented for recognition of digits. A two-step recognition process improves the accuracy of the recognition. In addition to the image processing techniques, cross-compiling, Objective-C^{TM} coding and Cocoa Touch^{TM} event handling are discussed in this application.

LIST OF FIGURES

1 LED and LCD 2
2 Application Flowchart 4
3 Overview of iOS Layers 5
4 Basic Structure of OpenCV 9
5 Sample Image Taken by iPhone 10
6 Canny Edge Detection 12
7 Test Image Passed to cvFindContours 13
8 Tree Nodes 14
9 Different Contour Retrieval Modes 15
10 Flow Chart for Finding Contour 17
11 Screen Location by Contour Finding Algorithm 18
12 Character Chopping 20
13 Outline Approximation 20
14 Features and Matching 21
15 Image for Training 21
16 Box File 22
17 Box File Correction 22
18 Recognition Results for LED (a) 24
19 Recognition Results for LED (b) 25
20 Recognition Results for LCD (a) 26
21 Recognition Results for LCD (b) 27
22 Mailing GUI 28
23 Rotated Image 30

CHAPTER I
INTRODUCTION

Overview

The iPhone is widely used around the world. This popularity is not only due to its innovative design and powerful built-in hardware support but also greatly relies on the success of the development platform (iOS^{TM} and SDK) and application distribution mode (the App Store^{TM}). Hundreds of new applications come out in the App Store every day, but most of them are related to game, entertainment, social networking, etc. An application with a medical purpose is rather rare. In this thesis, a medical application will be introduced.

Motivation

Medical devices with LCD or LED screens are quite common. While they offer many conveniences, the data is usually not properly recorded. It is extremely useful to have those data recorded and sent to a database, especially for patients who have chronic diseases. This data will allow medical professionals to track the patients between appointments.

This application will be used to read the digits as shown on medical devices that have LCD or LED screens. The user can easily take a picture with the iPhone, and this application will convert the digits as shown in the picture to numbers automatically. The user can then send the text and pictures via email through the application to a medical logging database.

Current Character Recognition Apps

Some character recognition applications have been released on App Store. Most of them offer text recognition, which is relatively easy to implement when text is assumed to consist of white background and dark characters. The noise in these images is assumed to be quite limited.

More complex recognition applications often require an Internet connection to recognize characters in an image. The application will send the image to a server that can do sophisticated and accurate computing in a very short time and return the result to the user. Although this technique greatly increases the accuracy of the recognition, this kind of application usually requires the user to pay a fee.

Proposed Approach

In this application, LCD/LED digits will be the objects to recognize. Because the digit colors for LCD and LED vary (black for LCDs and green, blue or red for LEDs), and the LCD has a low contrast compared to LED, as shown in Figure 1, image preprocessing is quite necessary before successful recognition.

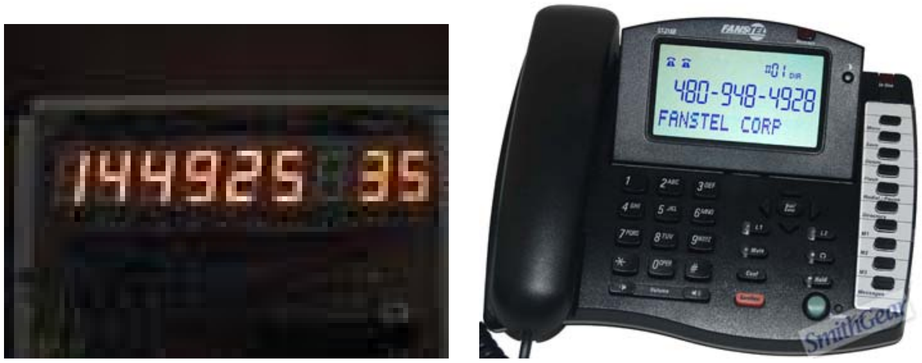


Figure 1 LED and LCD

Though the iPhone 4 is equipped with an ARM Cortex-A8^{TM} CPU, which has a maximum clock rate up to 800 MHz and 512 MB DRAM [2], computing is still quite limited due to the extensive hardware resources allocated to iOS. Any complex image processing technique might kill the application instantly. Even if a complex algorithm is successful running on the iPhone, the processing speed should also be considered. Five seconds for processing in real environment is perfectly acceptable, ten seconds is also tolerable, but twenty seconds or longer might not useful.

This application will not only address the recognition problem but also consider the processing time. The contour finding preprocessing algorithm is a good choice for the processing speed. Two open source libraries, OpenCV and Tesseract, make this application totally free of royalties. The overall flowchart for this application is shown in Figure 2.

The thesis is organized as follows. Chapter II first introduces background information about iOS, SDK and cross-compiling techniques, and then the image preprocessing and optical character recognition are discussed. Chapter III discusses the results, and Chapter IV contains the conclusion and future work.

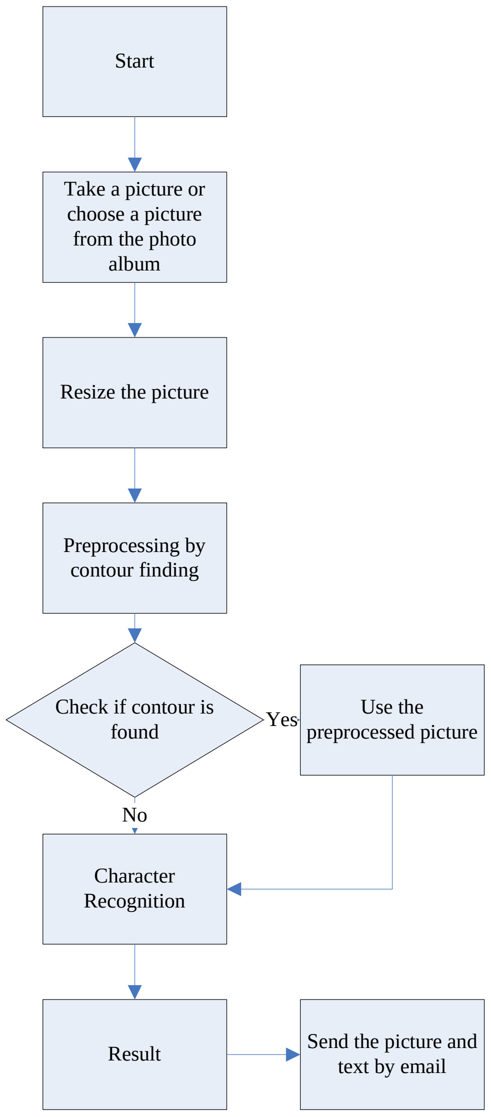

Start
Take a picture or choose a picture from the photo album
Resize the picture
Preprocessing by contour finding
Check if contour is found
Yes
Use the preprocessed picture
No
Character Recognition
Result
Send the picture and text by email

Figure 2 Application Flowchart

CHAPTER II

APPLICATION OVERVIEW

In this chapter, iOS will be introduced, and the difference between desktop coding and iPhone coding will be discussed. With the help of cross-compiling, we are able to use a workstation to include libraries for the iPhone, which has an ARM architecture. The image preprocessing stage will greatly decrease the noise, and the Tesseract library will be used for fast recognition.

Background of iOS

The mobile operating system developed by Apple^{TM} is called iOS (also known as iPhone OS). It is derived from Mac OS X^{TM}, which shares the Darwin foundation [3]. Apple's iOS is a UNIX^{TM} based operating system. Although iOS has a lot in common with Mac OS X, iOS is specifically designed to meet the needs of a mobile environment, where the users' needs are different. The multi-touch interface and accelerometer support are provided.

The iOS kernel has four layers to implement applications on the platform. Figure 3 shows an overview of these layers [3].

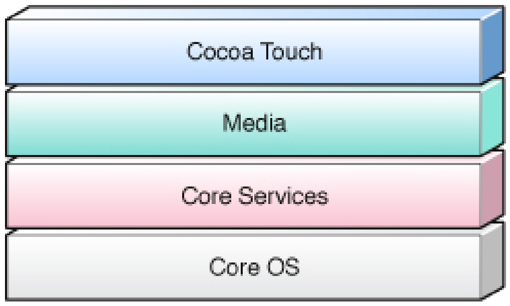


Figure 3 Overview of iOS Layers

The lowest two layers at the bottom are Core OS and Core Services, which contain the fundamental interfaces for iOS [3]. These interfaces are mainly used for access of files, low-level data types, Bonjour^{TM} services (service discovery protocol), network sockets, etc. These interfaces such as Core Foundation, CFNetwork, SQLite^{TM} are generally C-based [3].

The third layer is the Media layer. It contains interfaces based on C and Objective-C. For example, this layer contains the fundamental interfaces such as OpenGL^{TM} ES, Quartz^{TM}, and Core Audio that support 2D and 3D drawing, audio, and video [3]. It also has a powerful animation engine called Core Animation.

The top layer is the Cocoa Touch layer, where most of the interfaces use Objective-C. These interfaces provide the fundamental infrastructure used by an application. For example, the Foundation framework provides object-oriented support for collections, file management, network operations, and more [3]. The UIKit framework provides the visual support for applications, including classes for windows, views, and the controllers that manage those objects. Other frameworks can be used to access the user’s contacts, photo album, accelerometers and other hardware of the device.

When creating a new application, we always start from the Cocoa Touch layer, because the frameworks in the Cocoa Touch layer provide handy functions to build standard system behaviors.

Software Development Kit

The iOS software development kit (SDK) supports the creation of applications that run natively in iOS; it contains abundant development tools like XCode^{TM} for general coding, Interface Builder^{TM} for graphical user interface design, and iPhone Simulator for application testing on the Mac^{TM}.

Although the iOS SDK comes with these convenient tools, the differences between desktop application coding and iPhone coding are significant.

The iPhone can only execute one application at any given time in addition to the iOS itself. The application itself will not be able to do anything that is not in response to user interaction.

Only one physical switch is present on the iPhone, the Home button. In place of the traditional keyboard, most of the user operations will be implemented by touch events. These touches can be used to detect simple interactions with content, such as selecting or dragging items, or they can be used to detect complex gestures and interactions, such as swipes or the pinch-open and pinch-close gestures [3].

Unlike desktop programs in which the user has access to almost everything, the iPhone restricts the application's access. They can only read and write files from a portion of the iPhone file system called the 'sandbox'.

The Objective-C implementation used to program on the iPhone is also quite different from the one used on Mac OS. The iPhone has no garbage collection. Developers must take care of every object they have created and release them after the usage of these objects, or memory leakage will occur.

Cross-compiling

In this application, two open source libraries will be used (OpenCV and Tesseract). Because the processor of the iPhone is ARM-based, cross-compiling for a different platform is quite important.

The first requirement for successful cross-compiling is to set up the correct environment variables: the desired iPhone SDK and the CPPFLAGS, CFLAGS etc. These flags will tell the compiler to target the ARM architecture.

The following two command lines set the environment variables for developer tools and iPhone SDK.

```
export DEVROOT=/Developer/Platforms/iPhoneOS.platform/Developer [4]
export SDKROOT=$DEVROOT/SDKs/iPhoneOS4.0.sdk [4]
```

Then we define the compiler flags for the ARM architecture:

export CPPFLAGS= "-I$SDKROOT/usr/lib/gcc/arm-apple-darwin9/4.0.1/ include / -I$SDKROOT/usr/include/ -miphoneos-version-min=2.2"
export CFLAGS="$CPPFLAGS -arch armv6 -pipe -no-cpp-precomp -isysroot $SDKROOT" [4]
export CPP="$DEVROOT/usr/bin/cpp $CPPFLAGS"
export CXXFLAGS="$CFLAGS"

The following command will be added to generate the static library:

LIBPATH_static=$LIBFILE.a [4]
LIBNAME_static=`basename $LIBPATH_static` [4]

We also must target the i386 architecture to build libraries that are able to run under the iPhone simulator environment.

Finally, a library will be generated containing both the ARM and i386 variants.

Image Preprocessing

OpenCV Overview

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real time computer vision, developed by Intel^{TM} and now supported by Willow Garage [5]. It is free for use under the open source BSD license.

OpenCV consists of five libraries; four of them are shown in Figure 4 [5]. The CV component contains the basic image processing and computer vision algorithms [5]. MLL is the machine learning library that contains many statistical classifiers and clustering tools [5]. HighGUI contains I/O routines and functions for storing and loading video and images [5]. CXCore contains the basic data structures and content [5].

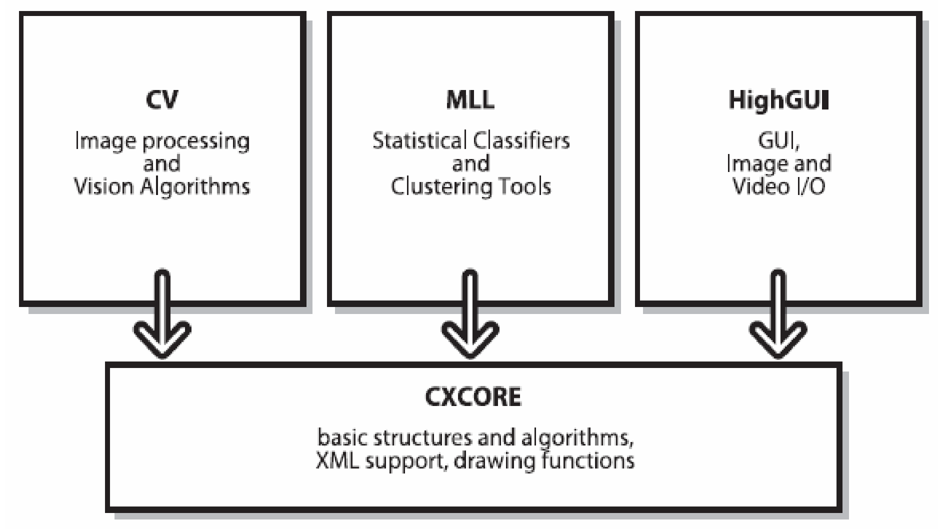

CV
Image processing and Vision Algorithms
MLL
Statistical Classifiers and Clustering Tools
HighGUI
GUI, Image and Video I/O
CXCORE
basic structures and algorithms, XML support, drawing functions

Figure 4 Basic Structure of OpenCV

The CvAux library is not shown above. It contains algorithms that have already been deprecated or are still in the experimental stage [5].

Locating the Device Screen

Due to the screen size differences between medical devices, the image has a substantial portion of background that contains structures in which we are not interested, as shown in Figure 5. The image is standard RGB format, with dimensions of 1936*2592.

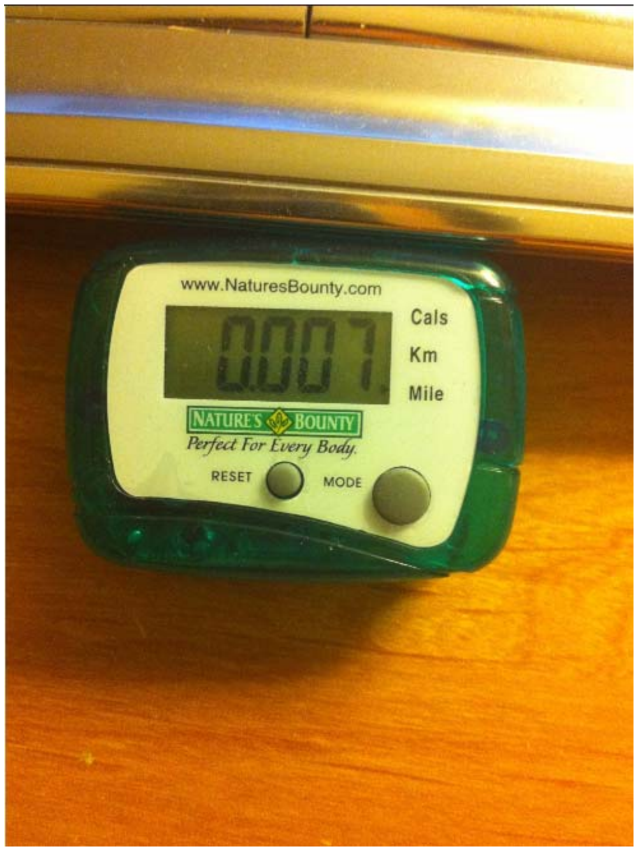


Figure 5 Sample Image Taken by iPhone

Because this application will be used with medical devices that have LCD or LED screens, the rectangle area is obviously our region of interest (ROI).

Canny Edge Detection

Different screens and their digits come in a variety of colors (black for LCDs and green, blue or red for LEDs), which makes it difficult to rely on color cues. On the other hand, we can see that most screens have an obvious boundary compared to the background, so edge detection for the ROI is a better solution because it significantly reduces the amount of data, while identifying and preserving the important structural properties in the image.

OpenCV provides a function for Canny edge detection, cvCanny [5]. It takes a grayscale image as input and outputs a binary image with detected edges.

```
void cvCanny( IplImage* img,
IplImage* edges,
double lowThresh,
double highThresh,
int apertureSize=3 );
```

The cvCanny edge function includes a four-stage algorithm [5]:

1. The image data is first smoothed by a Gaussian function.
2. Next, the smoothed image is differentiated with respect to the directions x and y. From the computed gradient values x and y, the magnitude and the angle of the gradient can be calculated using the arctangent function.
3. After the gradient has been calculated at each point of the image, the edges can be located at the points of local maximum gradient magnitude.
4. The final stage is edge thresholding, also called hysteresis thresholding. If there is only one single threshold limit, if the edge values fluctuate above and below this value, the line appears broken, which is commonly referred as ‘streaking’. The Canny edge detector solves this problem by setting an upper and lower edge value limit. If a value lies above the upper threshold limit, it is immediately accepted. If the value lies below the low threshold, it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels that are accepted. Thus the streaking phenomenon is greatly reduced because the points must fluctuate above the upper limit and below the lower limit for streaking to occur. A ratio of high:low threshold between 2:1 and 3:1 is usually recommended.

Figure 6 shows the image after the Canny edge detection is applied.

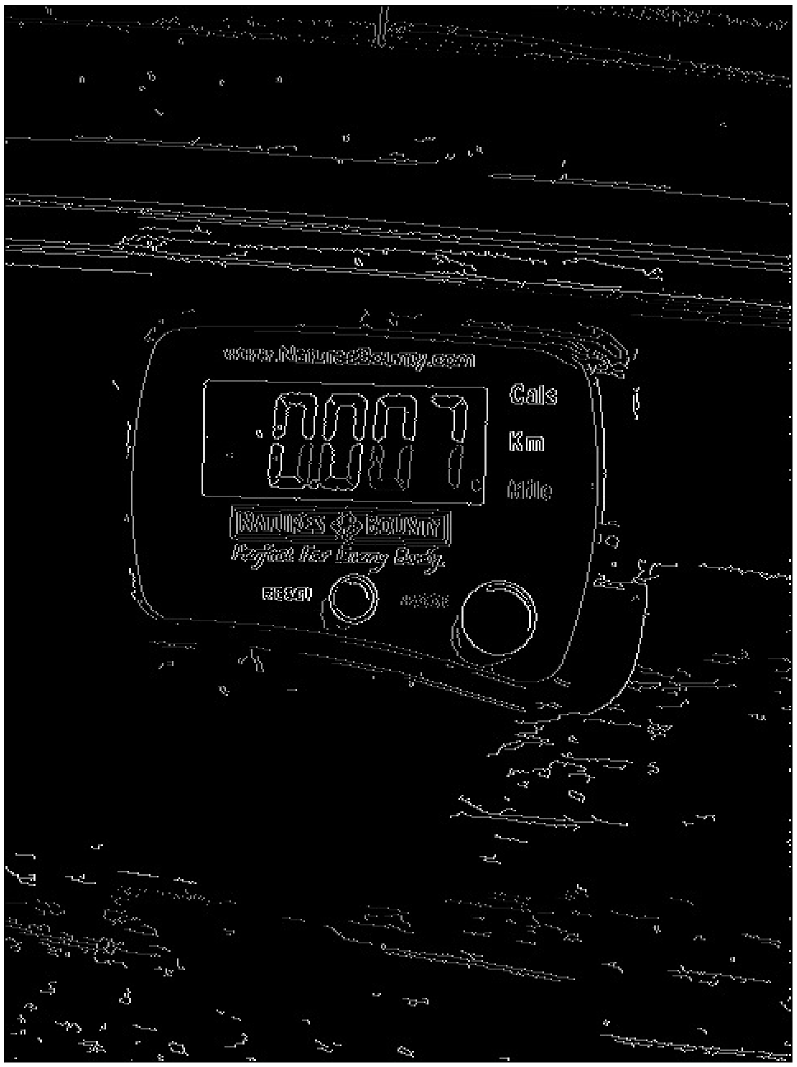


Figure 6 Canny Edge Detection

Finding Contours

Although the Canny edge detector can find the edge pixels, it does not tell us anything about those edges as entities in themselves. The next step is assemble those edge pixels into contours.

OpenCV has a function called cvFindContours [5] that computes contours from binary images.

```
int cvFindContours (
IplImage* img,
CvMemStorage* storage,
CvSeq** firstContour,
int headerSize = sizeof(CvContour),
CvContourRetrievalMode mode = CV_RETR_LIST,
CvChainApproxMethod method = CV_CHAIN_APPROX_SIMPLE
);
```

A test image is passed to cvFindContours as shown in Figure 7.

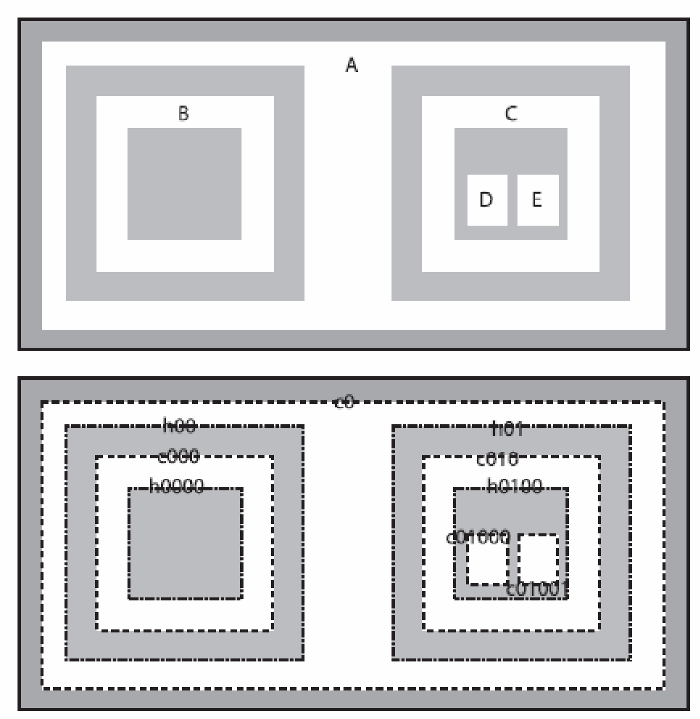


Figure 7 Test Image Passed to cvFindContours

We can see from the above figure that contours are labeled as cX and hX, where ‘c’ stands for ‘contour’, ‘h’ stands for ‘hole’, and X is the number of each specific instance. It is not generally useful to only find these contours. Instead, OpenCV can be asked to assemble the found contours into contour trees. CvContourRetrievalMode is a key argument here; it indicates to cvFindContours exactly which contours we would like to find and how we would like the result to be presented to us. OpenCV provides us four variable pointers linked to each given contour (h_next, h_prev, v_next, v_prev). The h_next and h_prev provide horizontal links; the v_next and v_prev provide vertical links. Figure 8 illustrates these links for contour c0.

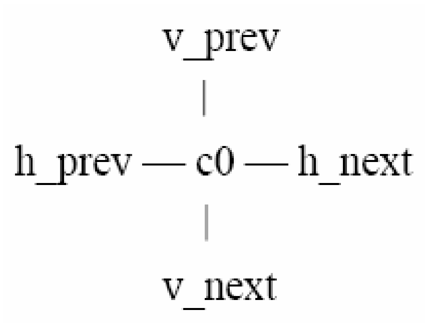
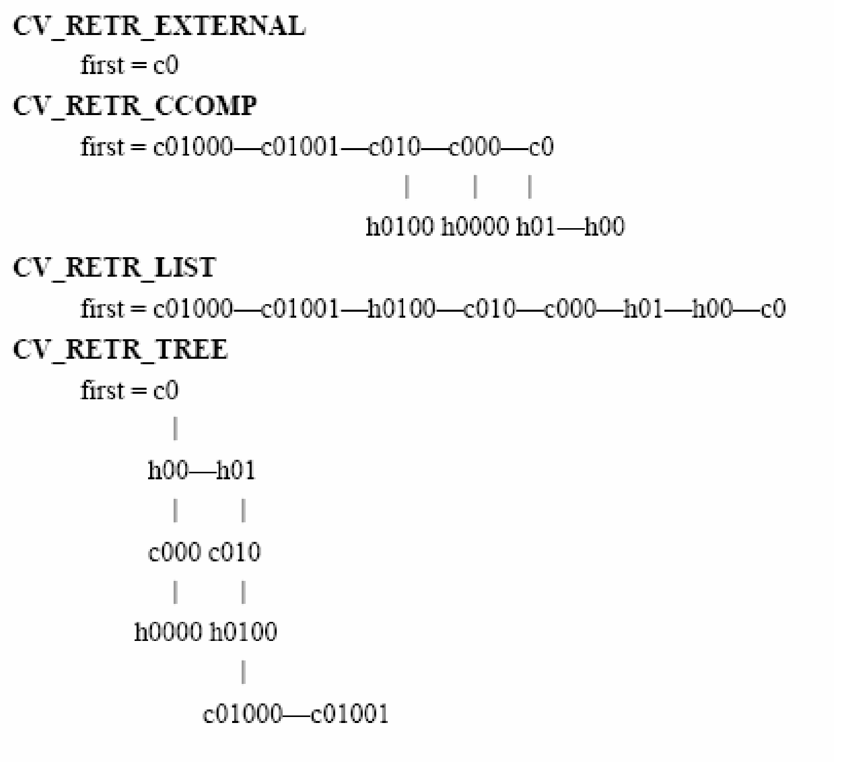


Figure 8 Tree Nodes

The mode can be set to any of these four values: CV_RETR_EXTERNAL, CV_RETR_LIST, CV_RETR_CCOMP, or CV_RETR_TREE. A simple illustration of different retrieval modes is shown below in Figure 9.



```
CV_RETR_EXTERNAL
        first = c0
CV_RETR_CCOMP
        first = c01000—c01001—c010—c000—c0
                                     |      |     |
                                h0100 h0000 h01—h00
CV_RETR_LIST
        first = c01000—c01001—h0100—c010—c000—h01—h00—c0
CV_RETR_TREE
        first = c0
                  |
               h00—h01
                  |      |
               c000 c010
                  |      |
             h0000 h0100
                         |
                   c01000—c01001
```

Figure 9 Different Contour Retrieval Modes

In this application, the CV_RETR_TREE mode will be used as the retrieval mode due to the completeness of the image contour information and easy handling for future algorithm modifications.

After we have obtained the contour tree, the contour finding algorithm will be applied to locate the device screen. From the Canny edge detection in Figure 2.4, we can see that the screen boundary is always a contour itself; it has a relatively fixed area range if the screen is properly located in the image, the aspect ratio for the screen contour is also in a certain range, and the screen contour will have a certain number of sub-contours, representing the digits inside it.

The contour finding algorithm can be illustrated through the following steps:

1. The pointer starts at the first contour of the contour tree and checks whether the first contour has any children. If the first contour has

children, then set the pointer to its direct child and repeat from step 1.

2. If no children are found, then check if the first contour has neighbors. If it has neighbors, then goes to its neighbor and repeat from step 1.

3. If no neighbors are found, then check if the first contour has parents. If no parents are found, then this contour should be the root of the contour tree, and the contour finding routine will return to this point.

4. If the first contour has parents, this contour is our candidate for the screen contour. We should first check its contour area to see whether it is in our predefined area range ([1000, 15000]). If the area satisfies the range requirement, the number of its total children will be computed, and if the number of its children is also in the predefined range (has more than 5 children). The candidate contour will be the final result only if it meets the above two criteria. Contours that fail any of these three criteria will set the pointer to its neighbor.

The area range is calculated by centered the screen within the image and the screen should take up to 1/10 of the whole image. The number of the children's range is defined bigger than 5, because there are always at least 5 digit contours and noise contours within the screen.

Figure 10 shows the flow chart of the algorithm to find the contour of the screen after the CV_RETR_TREE mode is applied.

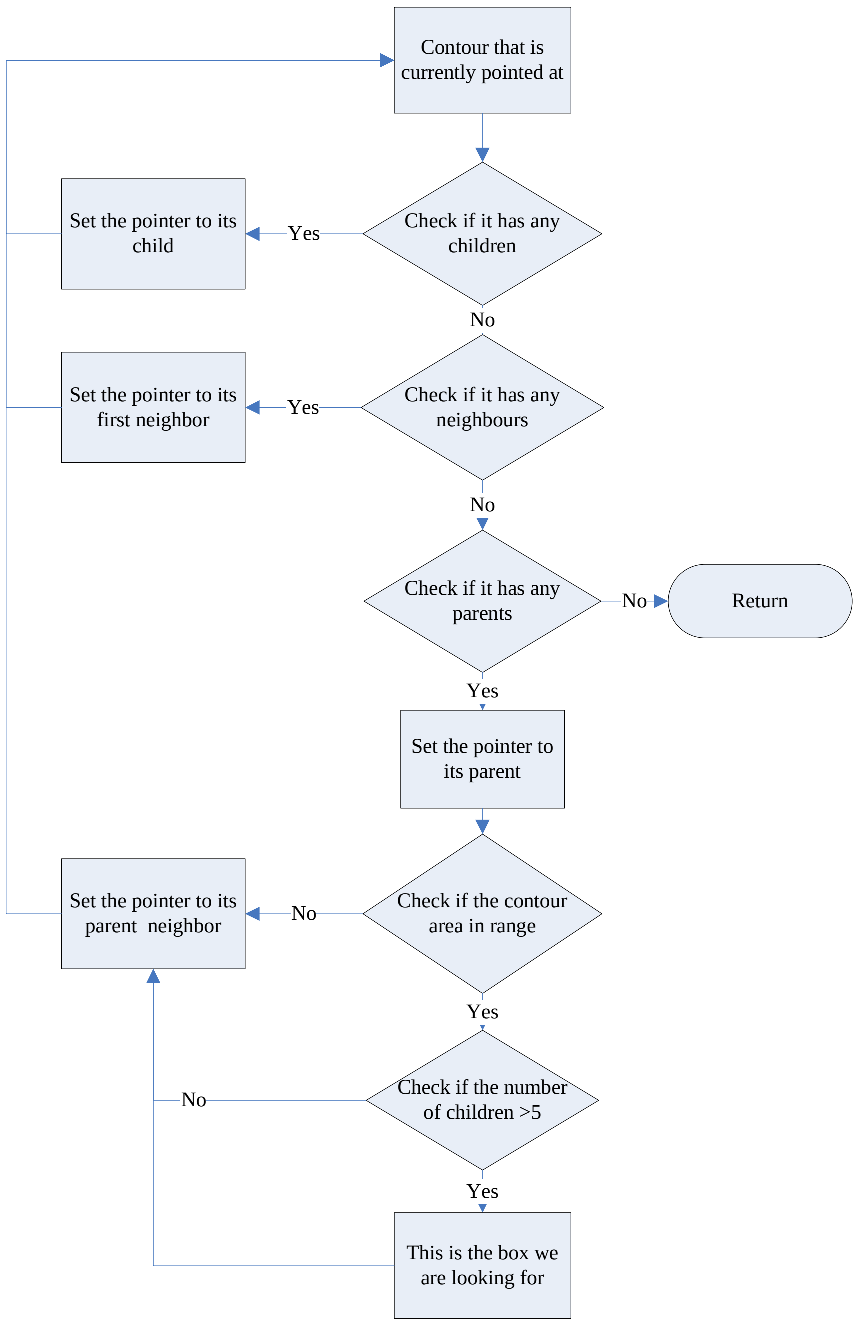

Contour that is currently pointed at
Check if it has any children
Yes
Set the pointer to its child
No
Check if it has any neighbours
Yes
Set the pointer to its first neighbor
No
Check if it has any parents
No
Return
Yes
Set the pointer to its parent
Check if the contour area in range
No
Set the pointer to its parent neighbor
Yes
Check if the number of children >5
No
Yes
This is the box we are looking for

Figure 10 Flow Chart for Finding Contour

Figure 11 shows the result after the contour finding algorithm.

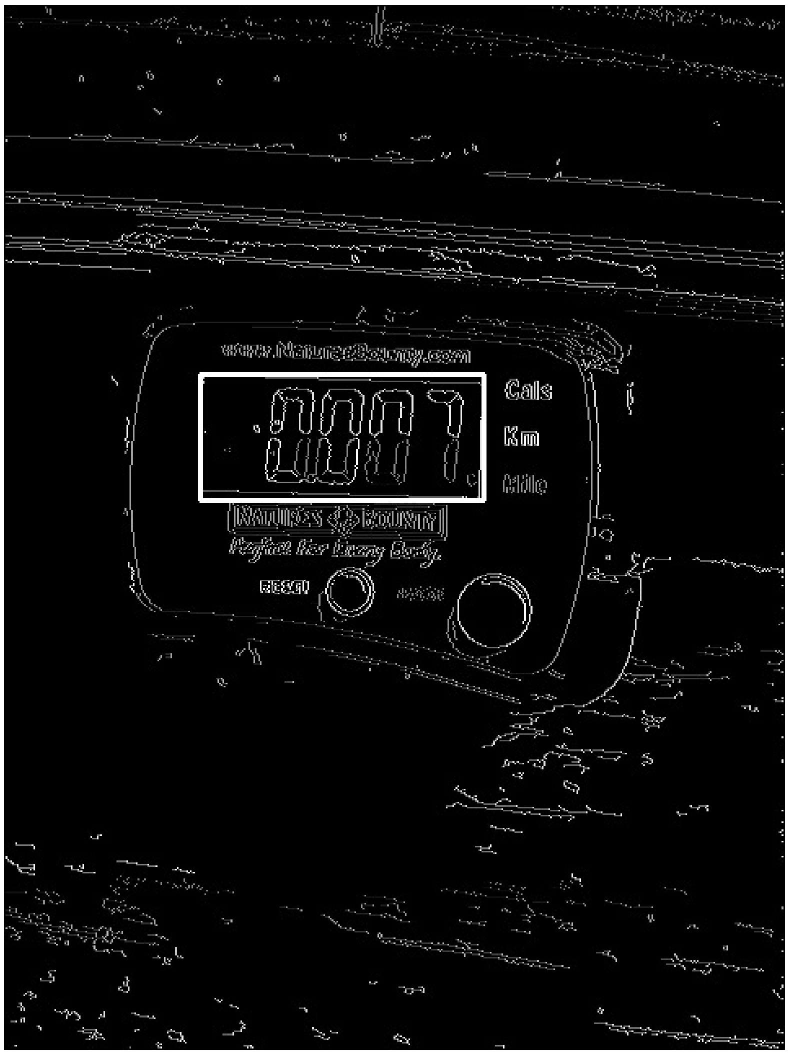


Figure 11 Screen Location by Contour Finding Algorithm

After the screen contour is found, we set the rectangle area as the ROI. OpenCV has a function called cvSetImageROI to set the region of interest.

Optical Character Recognition

Tesseract Overview

Tesseract OCR is an open source optical character recognition engine that was originally developed by Hewlett-Packard between 1985 and 1995 [6]. After ten years without any development taking place, Hewlett Packard^{TM} and UNLV released it as open source in 2005 [6]. Tesseract is currently developed by Google^{TM} [6].

The OCR process for Tesseract can be summarized in the following procedures [6]:

1. Outlines are stored by connected component analysis.
2. Outlines are gathered into blobs.
3. Blobs are organized into text lines and broken into words.
4. The first pass of the recognition procedure attempts each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data.
5. Previous words that were learned by the classifier will be used for a second pass to improve recognition for the words that are not properly recognized in the first pass.

Word Recognition

During the recognition process, the Tesseract will chop the joined characters by picking some of the concave vertices as shown in Figure 12 [5].

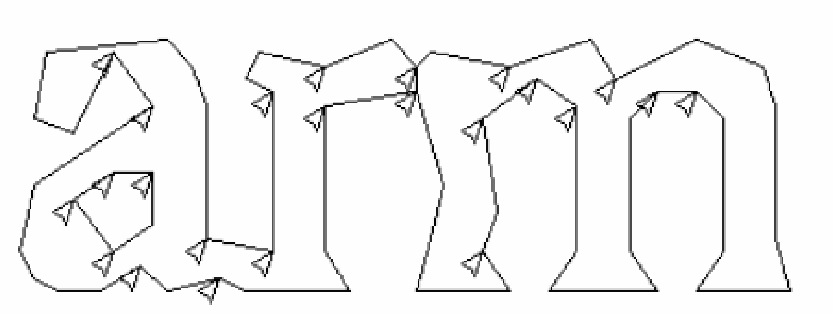


Figure 12 Character Chopping

From Figure 2.11, we can see that most of the chopping points chop the characters into pieces; Tesseract will associate the broken characters by making a search of the segmentation graph, which will consider the possible combinations of the chopped blobs as candidate characters.

The feature extraction will be done by outline approximation; any pieces from the chopping step will be approximated as a polygon, as shown in Figure 13 [5].

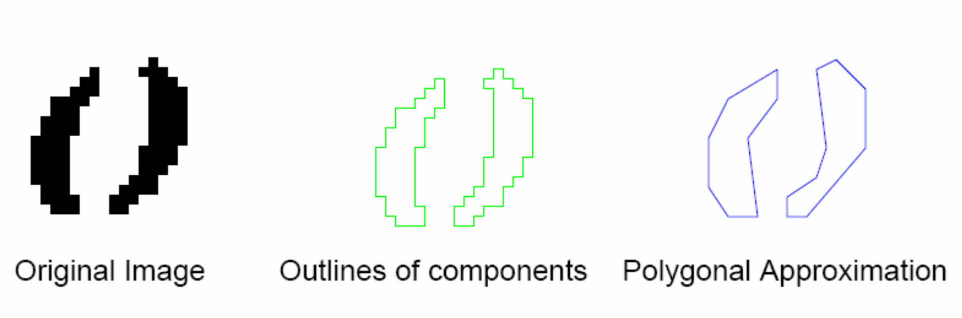


Figure 13 Outline Approximation

The classification proceeds as a two-step process. In the first step, Tesseract will create a short list of character class prototypes that might match the unknown characters. If the prototypes match the previously extracted features from earlier, then the second step will be applied by checking whether the extracted features also match the prototypes. The classification procedure is shown in Figure 14 [5].

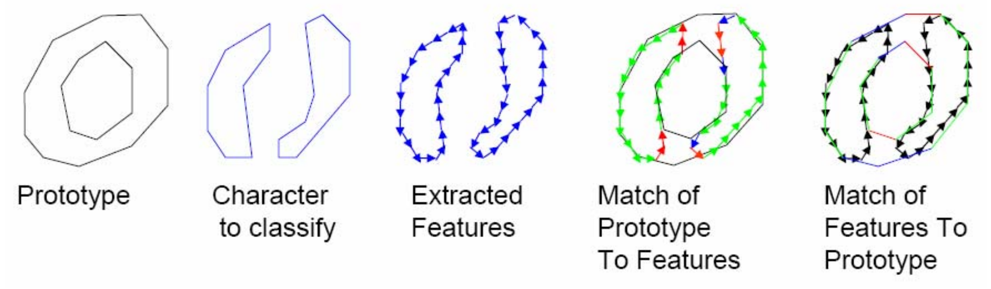

Prototype
Character to classify
Extracted Features
Match of Prototype To Features
Match of Features To Prototype

Figure 14 Features and Matching

Training Tesseract

The first step for training Tesseract is generating a training image. The training image should be a binary image with a ‘tif’ image type.

Figure 15 shows the image used for seven-segment display training.

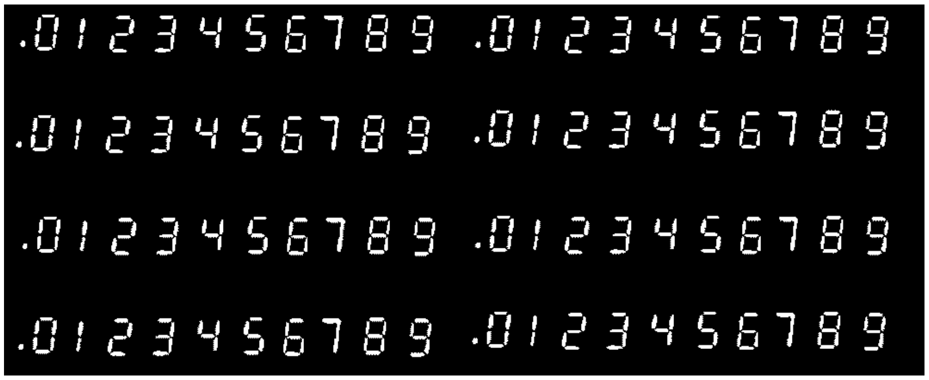
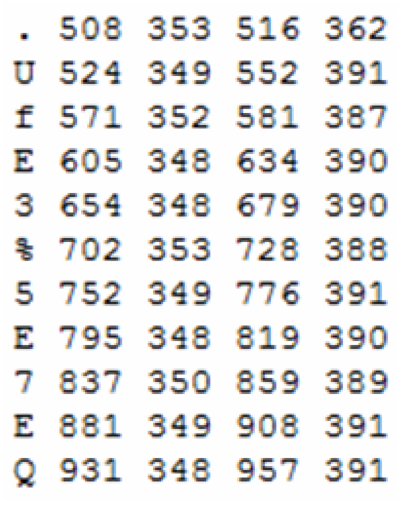


Figure 15 Image for Training

After the training image is created, a box file must be generated to go with the training image. The box file is a text file that lists the characters in the training image, in order, one per line, with the coordinates of the bounding box around the image. After the box is generated, we must manually edit the box file to substitute the correct character for each incorrect instance.

Figure 16 shows the box file after the above training image is applied.

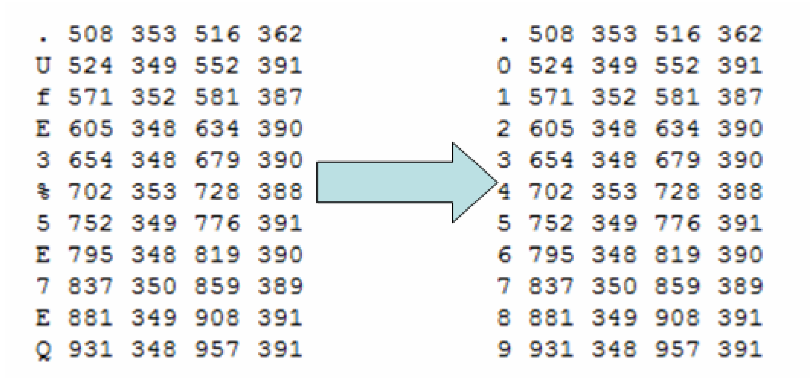


```
. 508 353 516 362
U 524 349 552 391
f 571 352 581 387
E 605 348 634 390
3 654 348 679 390
% 702 353 728 388
5 752 349 776 391
E 795 348 819 390
7 837 350 859 389
E 881 349 908 391
Q 931 348 957 391
```

Figure 16 Box File

The box file contains five columns. The first one lists the characters recognized by Tesseract, and the following four columns give the bounding box of each character, i.e., left, bottom, right and top.

From the above box file, we can see that many characters are incorrect. Figure 17 shows an example of box file correction.



```
. 508 353 516 362        . 508 353 516 362
U 524 349 552 391        0 524 349 552 391
f 571 352 581 387        1 571 352 581 387
E 605 348 634 390        2 605 348 634 390
3 654 348 679 390        3 654 348 679 390
% 702 353 728 388        4 702 353 728 388
5 752 349 776 391        5 752 349 776 391
E 795 348 819 390        6 795 348 819 390
7 837 350 859 389        7 837 350 859 389
E 881 349 908 391        8 881 349 908 391
Q 931 348 957 391        9 931 348 957 391
```

Figure 17 Box File Correction

After correcting all the characters, we pass the new box file to Tesseract again, and it will generate new character features. When the character features of the training image have been extracted, we cluster them to create the prototypes by using the ‘mftraining’ and ‘cntraining’ functions.

```
mftraining fontfile_1.tr fontfile_2.tr
cntraining fontfile_1.tr fontfile_2.tr
```

After all the language data have been generated, we merge these files together and name it ‘seg’. We add the folder path to the application bundle, and then Tesseract is ready to recognize digits.

From the previous illustration, we can see that the contour finding algorithm provides a quick way to de-noise the image while consume modest hardware resources. The Tesseract library is also easy to implement for iPhone programming.

Final recognition results and error rates will be illustrated and analyzed in the next chapter.

CHAPTER III

RESULTS

In this section, LED and LCD recognition results will be shown, and the analysis of the recognition accuracy will be provided.

Figure 18 and Figure 19 show example results for LED screens.

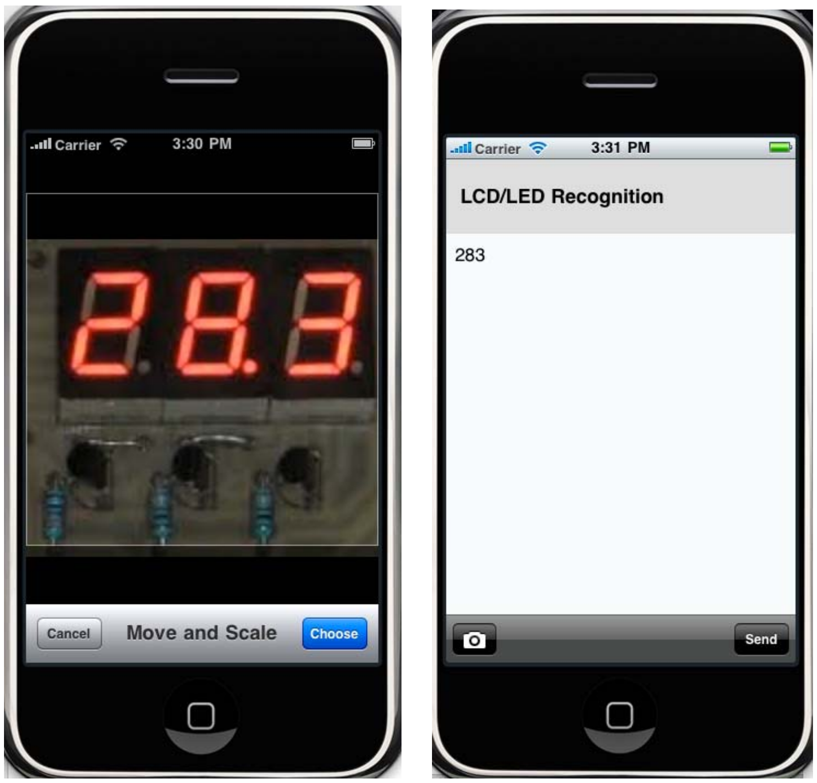
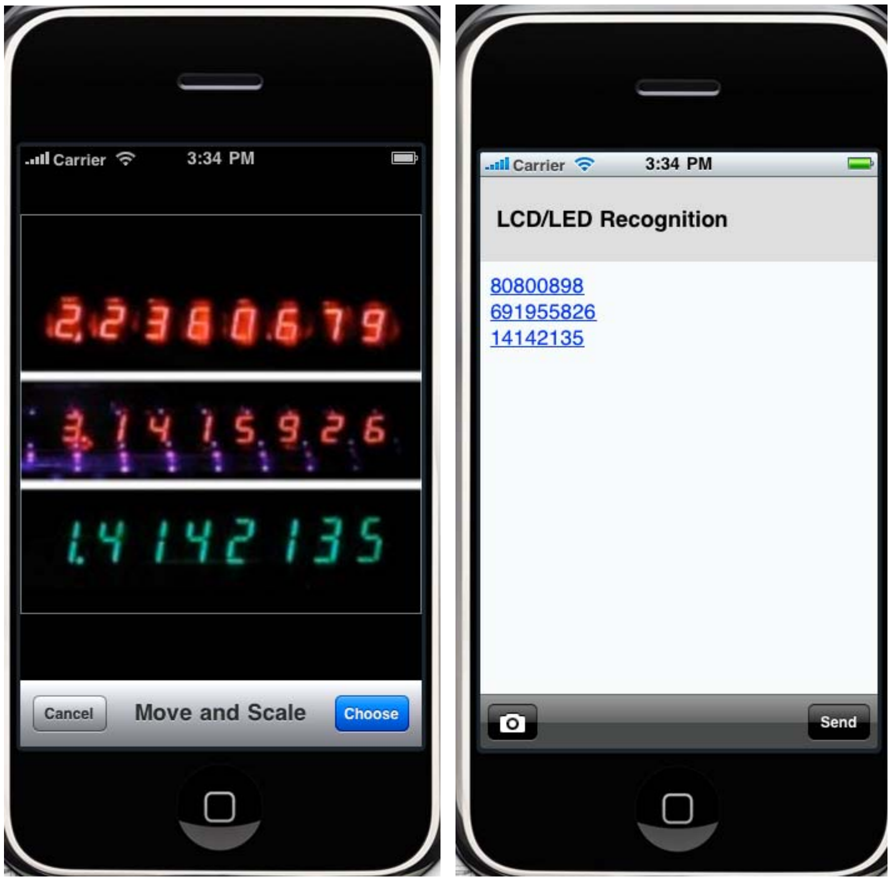


Figure 18 Recognition Results for LED (a)

Figure 18 is quite simple to recognize. The digits are in the same line and the numbers have high contrast compared to the background. The decimal point is missing here mainly due to the lack of eight-segment display training. The decimal point is part of the digit rather than an individual digit.



LCD/LED Recognition

80800898
691955826
14142135

Figure 19 Recognition Results for LED (b)

The results for Figure 19 are not as good as results for the previous one, mainly because the first lines of the digits are blurred. After thresholding, the blurred digits could appear as large white spots, so that the recognition algorithm takes the digits as ‘0’ or ‘8’. The second line has noise in the background; the recognition algorithm could include that noise as part of the digits.

Figure 20 and Figure 21 show example results for LCD screens.

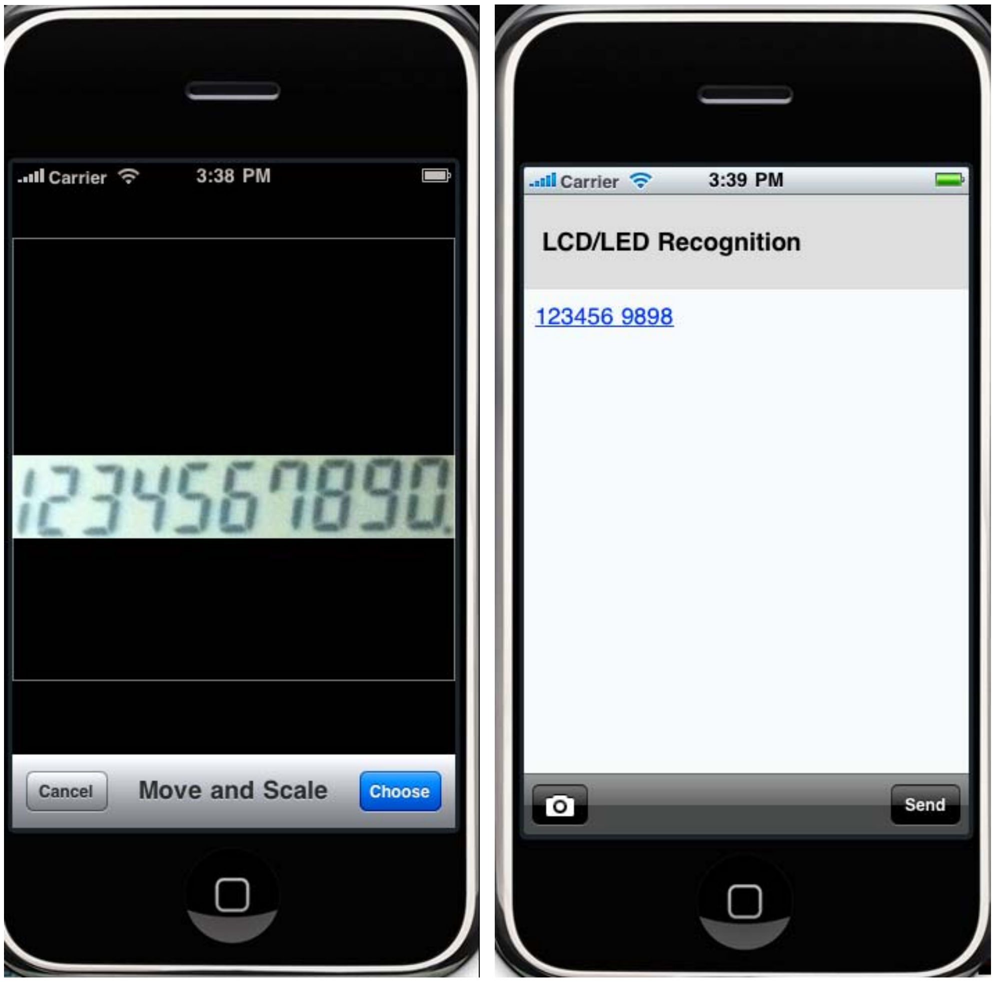
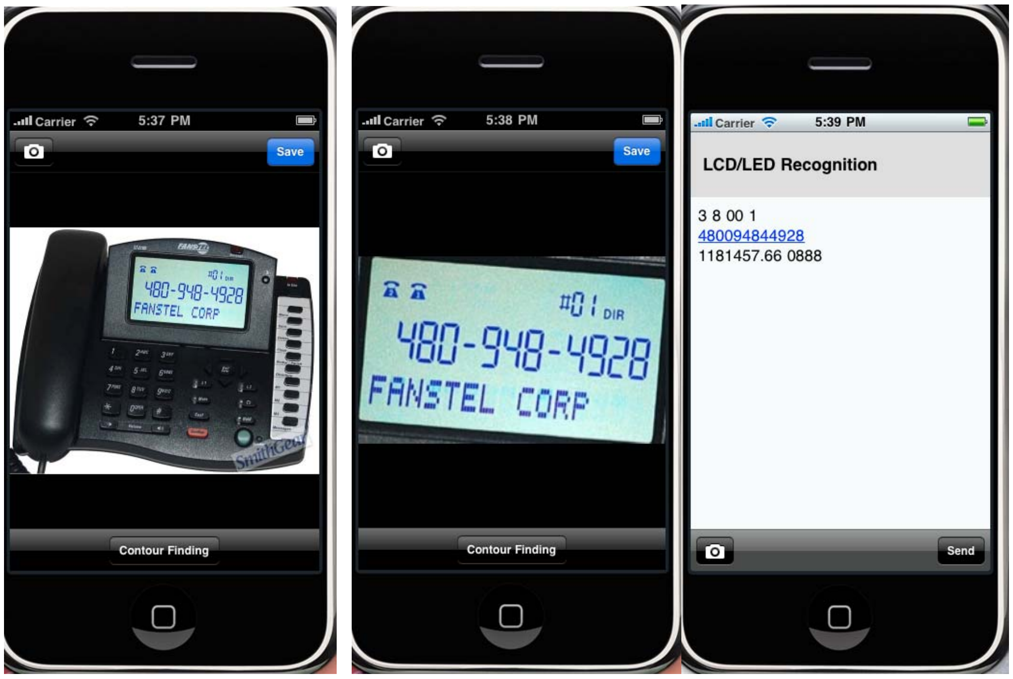


Figure 20 Recognition Results for LCD (a)

Figure 19 is a series of digits taken from a calculator; the ‘7’ and ‘0’ are incorrectly recognized as ‘9’ and ‘8’, respectively.

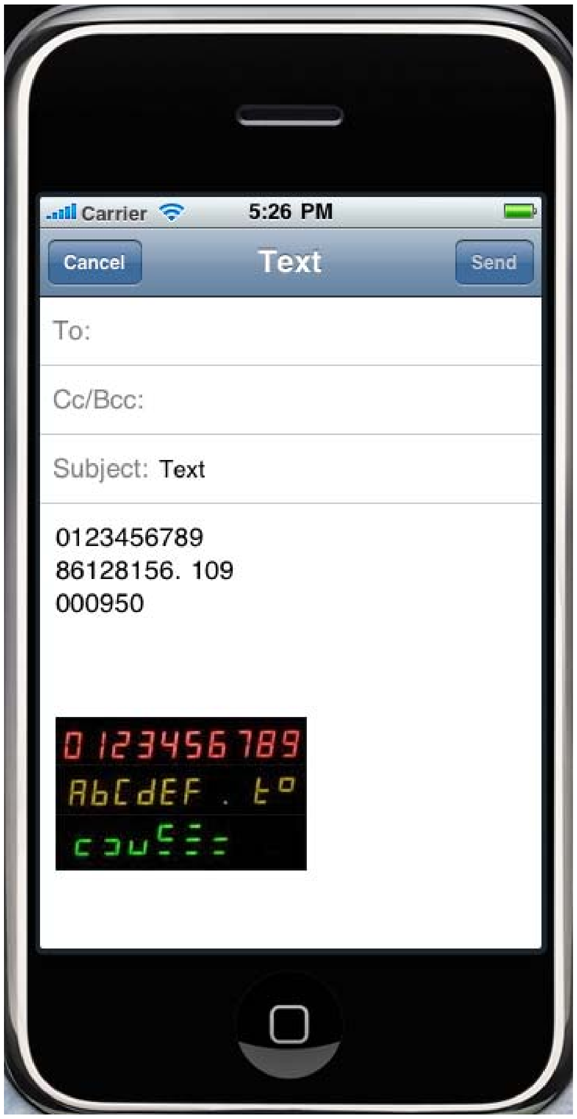


```
3 8 00 1
480094844928
1181457.66 0888
```

Figure 21 Recognition Results for LCD (b)

Figure 21 has a very good result for screen location, although there is a rotation of the image. The recognition is also good; the dashed line between the digits is recognized as random numbers. Because the contour finding algorithm can not perform noise reduction within the screen area, the first line and the third line are also considered as part of the recognition objects. They are recognized as digits due to the lack of training for English characters.

Figure 22 shows the mailing GUI. This is the user interface for e-mailing the recognized text combined with the picture.



0123456789
86128156. 109
000950

Figure 22 Mailing GUI

Error Analysis

The error rate for digit recognition is shown in Table 1.The row is the digit to be recognized and the column is the result after recognition.

Table 1 Error Rate for Digit Recognition
Row: Digit to be recognized Col: Result after recognition

Row Col	0	1	2	3	4	5	6	7	8	9
0	82.22%	0.00%	0.00%	5.26%	0.00%	0.00%	0.00%	0.00%	3.77%	5.77%
1	0.00%	81.36%	0.00%	0.00%	0.00%	0.00%	0.00%	5.56%	0.00%	0.00%
2	0.00%	0.00%	92.59%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
3	0.00%	0.00%	0.00%	78.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
4	0.00%	0.00%	0.00%	0.00%	90.57%	0.00%	0.00%	0.00%	0.00%	0.00%
5	0.00%	0.00%	0.00%	0.00%	0.00%	83.67%	4.17%	0.00%	0.00%	0.00%
6	0.00%	0.00%	0.00%	0.00%	5.66%	10.20%	83.33%	0.00%	11.32%	0.00%
7	0.00%	10.17%	0.00%	0.00%	0.00%	0.00%	0.00%	72.22%	0.00%	0.00%
8	17.78%	0.00%	3.70%	8.77%	0.00%	0.00%	12.50%	0.00%	75.47%	9.62%
9	0.00%	0.00%	0.00%	7.02%	0.00%	6.12%	0.00%	22.22%	9.43%	84.62%
No result	0.00%	8.47%	3.70%	0.00%	3.77%	0.00%	0.00%	0.00%	0.00%	0.00%
Total counts	45	59	54	51	53	49	48	36	53	52

The test for the recognition accuracy is made by recognizing a Casio fx-350TL calculator LCD, 10 digits at a time. 500 digits of the constant π are used. The most common errors are the mixing between ‘7’ and ‘9’, ‘8’ ,‘6’ and ’0’, ‘5’ and ‘6’, with only one segment difference within these sets. ‘1’ is a very special digit because it takes only half of the width compared to other digits; it is a relatively smaller object for the recognition stage than other digits. The recognition could easily take it as part of the noise and therefore cannot correctly recognize it. The recognition of decimal points does not work well because the application can not reliably find them on LCDs. The LCDs have relatively lower contrast than LEDs, and the decimal points are very small.

On the other hand, we can see from the above images that LEDs usually have better recognition results than the LCDs, even for the recognition of decimal points, mainly due to the high contrast level that LEDs offer.

Another factor that affects the final result is the rotation of the image. Because the image is taken from a handheld camera in an iPhone, the text lines may skewed from their original orientation, as shown in Figure 23.

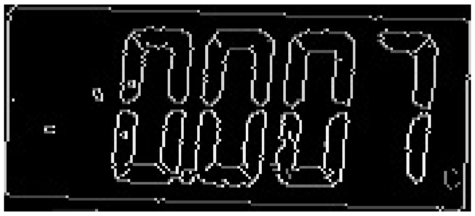


Figure 23 Rotated Image

The user technique in taking the picture also greatly influences the final result. Because the screen is located based on the contour finding algorithm, the contour finding will first determine the size of the contour area, so it would be difficult to recognize the screen if the user takes the picture from too great a distance. The best image would have the screen centered in the picture. Any light reflection when taking the picture is also an important factor. Avoiding the use of flash and placing the camera at a proper angle to minimize the light reflection would greatly increase the recognition accuracy.

Conclusions and future work will be illustrated in the next chapter.

CHAPTER IV

CONCLUSIONS AND FUTURE WORK

An automated software application that is able to read and interpret LED/LCD screens would be very practical for convenient daily use. The contour finding algorithm is not difficult to implement on the iPhone, and it consumes modest computing resources. The Tesseract library is also very handy for iPhone programming. We can even recognize other objects like vehicle license plates with appropriate training. Because vehicle license plates usually have the same size and contain numbers and letters, the contour finding algorithm would be a good solution [7].

For future work, the current preprocessing algorithm still can be improved, because the contour finding algorithm can only be used with devices that have a screen with obvious boundary. Any LED/LCD without obvious boundary might be difficult to analyze. The contour finding algorithm also has disadvantages because of the use of ‘magic numbers’, which greatly reduces the robustness of the program. Skew detection and correction should also be considered, because any rotation of the image will affect the accuracy of the recognition stage. A possible solution for de-skew can be baseline detection and angle correction based on center of gravity of the characters [8].

It will also be useful to improve the character recognition algorithm. Otsu’s method for thresholding is a good choice, because after the preprocessing stage, we always have a screen image with two histogram clusters. One of them is the screen background and the other one is the digits. This will lead a more precise and robust recognition results. We can also establish a database to receive these recognized numbers in order to construct the whole system.

BIBLIOGRAPHY

[1] SmithGear^{TM} .Fanstel ST-218B 2-Line Corded Phone. Available: http://www.smithgear.com/fan-st218.html

[2] Wikipedia. Introduction of iPhone. Available: http://en.wikipedia.org/wiki/IPhone

[3] Apple. iOS Overview. Available: http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//apple_ref/doc/uid/TP40007592

[4] iPhone Programming Tips: building UNIX software. Available: http://latenitesoft.blogspot.com/2008/10/iphone-programming-tips-building-unix.html

[5] Gary Bradski, Adrian Kaehler, "Learning OpenCV: Computer Vision with the OpenCV Library", O'Reilly Media, 1st edition, September, 2008.

[6] Ray Smith, "An Overview of the Tesseract OCR Engine", Google Inc.

[7] Wikipedia.Vehicle registration plates of the United States. Available: http://en.wikipedia.org/wiki/Vehicle_registration_plates_of_the_United_States

[8] Atallah Mahmoud Al-Shatnawi and Khairuddin Omar. "Skew Detection and Correction Technique for Arabic Document Images Based on Centre of Gravity", University Kebangsaan Malaysia, Selangor, Malaysia, 2009.