Lab 5: Decoding an EAN-13 Barcode

TRC3500 Sensors and Artificial Perception

## Lab members

|  |  |
| --- | --- |
| Kun Zhang | 22701478 |
| Sheng Lun Seow | 22724281 |
| Jiapeng Zou | 24531308 |

# 1. Introduction

In this laboratory practice, details of EAN-13 barcode have been studied. With that information, a C program with OpenCV was created for decoding it using some computer vision techniques. The program started by capturing image of the barcode using the USB camera provided and then converted it to a binary image. Later on, it scans across centre half height of the image to produce a binary scan of the barcode that was used for decoding the information carried in the barcode.

# 2. Equipment and Software

Equipment used in the lab:

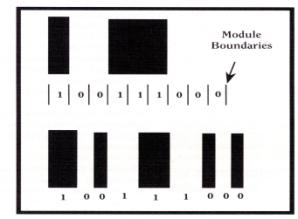
1. 720p USB Webcam
2. a PC

Software used in the lab:

1. Microsoft Visual Studio 2010 Professional
2. OpenCV
3. The supplied skeleton program

# 3. Theoretical Background

Barcode is presented in binary codes together with its encoded numbers. The black bar represented a “1” while a white space represented a “0” that each of the character should be of fixed length and stand-alone.



**Figure 1: Barcode representation in binary codes**

## EAN-13 Barcode

An EAN-13 barcode is a 13-digit barcoding system that consists of 12 data digits and 1 modulo check digit. It is a superset of the original 12-digit Universal Product Code (UPC) system developed in the United States. It is widely used in marking products such as book, cd and etc. [1]



**Guard Bars**

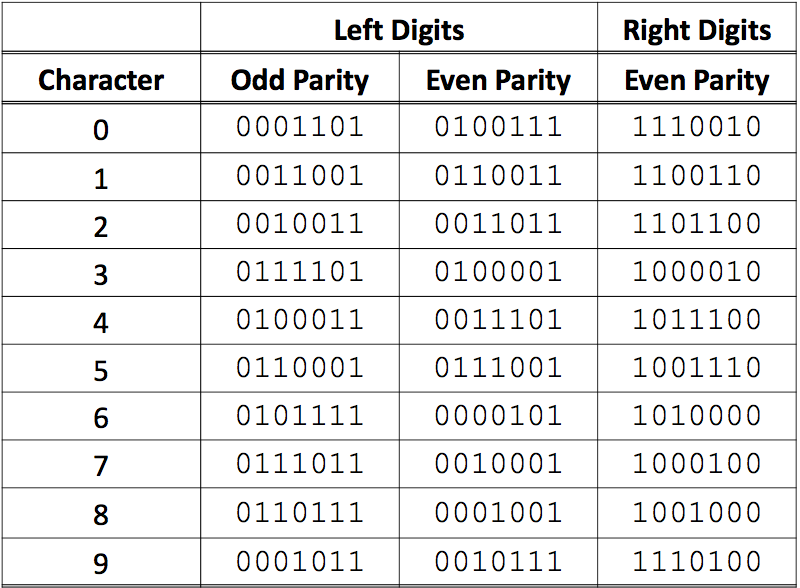
**Figure 2: EAN-13 barcode example**

In EAN-13 barcode, the first two digits are normally carried the information of manufacturer’s country code. Next 4, 5 or 6 digits carried company code and the next 2 to 6 digits carried item number. The last digit is the modulo check digit.

The barcode is divided into 2 groups: left-hand side group and right-hand side group, which separated by 3 sets of guard bars as shown in figure 2 above. The left-most digit is not coded as barcode; it is encoded in the parity of the left-hand side group of digits. Table 1 and Table 2 below show the EAN-13 parity encoding and EAN-13 character encoding respectively

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **First Digit** | **Second Digit** | **Third Digit** | **Fourth Digit** | **Fifth Digit** | **Sixth Digit** | **Seventh Digit** |
| **0** | Odd | Odd | Odd | Odd | Odd | Odd |
| **1** | Odd | Odd | Even | Odd | Even | Even |
| **2** | Odd | Odd | Even | Even | Odd | Even |
| **3** | Odd | Odd | Even | Even | Even | Odd |
| **4** | Odd | Even | Odd | Odd | Even | Even |
| **5** | Odd | Even | Even | Odd | Odd | Even |
| **6** | Odd | Even | Even | Even | Odd | Odd |
| **7** | Odd | Even | Odd | Even | Odd | Even |
| **8** | Odd | Even | Odd | Even | Even | Odd |
| **9** | Odd | Even | Even | Odd | Even | Odd |

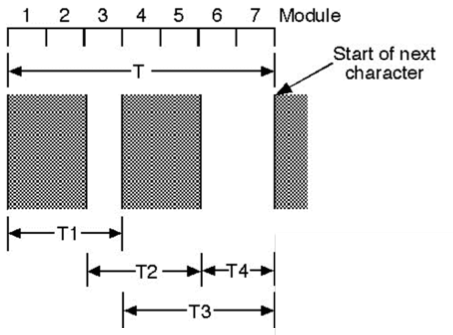
**Table 1: EAN Parity Encoding**



**Table 2: EAN-13 Character Encoding**

## Decoding

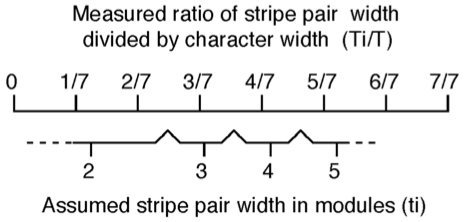
In EAN-13 barcode, each character has T-values with 7 modules inside. To decode the character, value of T1, T2 and T4 need to be measured according to the bar width within the 7 modules.



**Figure 3: T-values**

However, before measuring the T-values, the position of the first bar needs to be determined by checking the rising edge and falling edge in the 7 modules. For example, if a binary “0” comes first, followed by binary “1” then a rising edge is detected. Otherwise, a falling edge is detected. After finding the position of the first bar, T1, T2 and T4 can now be measured. T1 is measured from the start of the first black bar to the beginning of the second black bar. As for T2, it is measured from the first black bar falling edge till the end of the falling edge of second black bar and same goes to T4, measured from the second black bar falling edge till the beginning of next rising edge or at the end of the 7 modules. Nonetheless, T4e value is usually measured to prevent the error caused by the bar code ink spreading. It is measured from the second black bar falling edge till the falling edge of the first black bar in the next character module. Therefore, width of the next character first black bar needs to be determined before finding T4e value. [2]

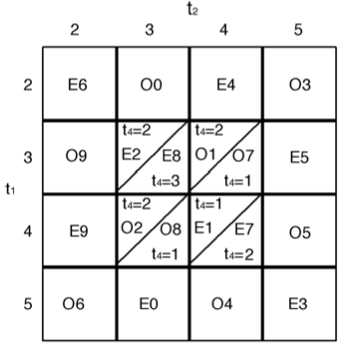
When all the required T-values are obtained from the barcode, they need to be converted to integer module values as shown in figure 4 below.



**Figure 4: Converting measurements to modules**

The measured T-values will be divided by the character width to get the corresponding ratios. The character width is ranged from 0 to 7 for each character and then the ratios are evaluated as follows [2]:

Finally, the evaluated T1, T2 and T4 values are decoded accordingly using the barcode decoding character shown in figure 5 below.



**Figure 5: Decoding character**

# 4. Procedures

In this section, the procedures of the code will be introduced.

### Modify the skeleton code provided so that it could grab image using the USB camera.

Initialize camera setting to check if the USB camera is successfully connected to PC

CvCapture\* capture = 0;

IplImage\* processedImage = 0;

cout<<"Andy's USB camera program"<<endl<<"Press any key to capture image for scan"<<endl;

cout<<"Press and hold 'q' to quit"<<endl;

// Initializes capturing video from camera

capture = cvCaptureFromCAM( -1 );

if( !capture ) {

fprintf(stderr,"Could not initialize capturing...\n");

return -1;

}

Create a window to display image and printout the width and height of the image

// Creates window

cvNamedWindow( "Camera image", 1 );

// Camera image

IplImage\* frame = 0;

// Grabs and returns a frame from camera

frame = cvQueryFrame( capture );

// Print details of image

cout<<"image width ="<<frame->width<<" height ="<<frame->height;

cout<<" depth ="<<frame->depth<<" channels ="<<frame->nChannels<<endl;

### Convert the image to binary image and find out the threshold of the image.

The three consecutive element in the array represent one pixel, thus assign these three element to grey colour

row[ x\*3 ] = gray;

row[ x\*3 + 1 ] = gray;

row[ x\*3 + 2 ] = gray;

### Scan across the middle of the image and convert it to binary values.

Each element in the middle array is assigned to either 0 or 1

if (y == (frame->height/2)){ //apply thresholding to image

if (gray <= threshold )

{

scanrow[x]=1;

gray = 0;

}else{

scanrow[x]=0;

gray = 1;}

### Identify rising edges and falling edges of each bar and store the location of the edges of each bar in an array

Whenever an edge is detected, its location is store in an array that only stores two locations: the location of the rising edge of that bar, and the location of the falling edge of that bar.

for(i = 0 ;i<640 ; i++)

{

if (((scanrow[i]==0)&&(scanrow[i-1]==1))|((scanrow[i]==1)&&(scanrow[i-1]==0))){

falledge+=1;

EDGE\_LINE\_GROUP[falledge] = i;

}

}

### Find out all the T-values (T1, T2, T4, T4e) using the data obtained in previous step and convert it to integer module values.\

Two functions were written to find out T values: one for left digits and the other for right digits. This is because the left digits and right digits use different rules in determining T values.

void LeftDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int beginP){

float T1,T2,T4;

float T = LINE\_2\_F-beginP;

T1 = LINE\_2\_F - LINE\_1\_F;

T2 = LINE\_2\_R - LINE\_1\_R;

T4 = LINE\_1\_R - beginP;

int T1\_width=T\_Width(T1,T);

int T2\_width=T\_Width(T2,T);

int T4\_width=T\_Width(T4,T);

COMPARE\_TABLE(T1\_width, T2\_width, T4\_width , &LEFT\_NUMBER[number], &parity\_left[number]);

}

void RightDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int endP){

float T1,T2,T4;

float T = endP - LINE\_1\_R;

T1 = LINE\_2\_R - LINE\_1\_R;

T2 = LINE\_2\_F - LINE\_1\_F;

T4 = endP - LINE\_2\_F;

int T1\_width=T\_Width(T1,T);

int T2\_width=T\_Width(T2,T);

int T4\_width=T\_Width(T4,T);

COMPARE\_TABLE(T1\_width, T2\_width, T4\_width , &RIGHT\_NUMBER[number], &parity\_right[number]);}

The T\_width function is used to convert the actual width (based on array numbers) into width values (1, 2, 3, 4 and 5) which can be used for look up table. The width values were found using formulae given by [1]

int T\_Width(float Ti, float T)

{

int width=0;

if((Ti/T)<=(3.0/14)){

width=1;

}

if((3.0/14)<(Ti/T)&&(Ti/T)<=(5.0/14)){

width=2;

}

if((5.0/14)<(Ti/T)&&(Ti/T)<=(7.0/14)){

width=3;

}

if((7.0/14)<(Ti/T)&&(Ti/T)<=(9.0/14)){

width=4;

}

if((9.0/14)<(Ti/T)&&(Ti/T)<=(11.0/14)){

width=5;

}

return width;

}

### Use the width value to look up corresponding digit

The look up table was coded with multiple IF conditions

void COMPARE\_TABLE(int T1\_width, int T2\_width, int T4\_width ,int \*character, char \*parity){

if(T1\_width==2&&T2\_width==2){

\*character=6;

\*parity='E';

}

if(T1\_width==2&&T2\_width==3){

\*character=0;

\*parity='O';

…

}

if(T1\_width==5&&T2\_width==4){

\*character=4;

\*parity='O';

}

if(T1\_width==5&&T2\_width==5){

\*character=3;

\*parity='E';

}

}

### 7) Using the parity of the left digits to determine the 13th digit

According to the table for the 13th digit, we could deduce the 13th digit by the following code:

int digit13(char par1,char par2,char par3,char par4,char par5,char par6)

{

//the 1st parity will always be zero, so ignore 1st parity

if((par2=='O')&&(par3=='O')&&(par4=='O')&&(par5=='O')&&(par6=='O'))

{

return 0;

}

……

if((par2=='E')&&(par3=='E')&&(par4=='E')&&(par5=='E')&&(par6=='E'))

{

return 10;

}

}

### 8) Print out all the values and their parity on the console window.

Print out the result while showing image on the screen

printf("LEFT\_NUMBER : %d,%d,%d,%d,%d,%d\n",LEFT\_NUMBER[1],LEFT\_NUMBER[2],LEFT\_NUMBER[3],LEFT\_NUMBER[4],LEFT\_NUMBER[5],LEFT\_NUMBER[6]);

printf("parity\_left : %c,%c,%c,%c,%c,%c\n",parity\_left[1],parity\_left[2],parity\_left[3],parity\_left[4],parity\_left[5],parity\_left[6]);

printf("RIGHT\_NUMBER : %d,%d,%d,%d,%d,%d\n",RIGHT\_NUMBER[1],RIGHT\_NUMBER[2],RIGHT\_NUMBER[3],RIGHT\_NUMBER[4],RIGHT\_NUMBER[5],RIGHT\_NUMBER[6]);

printf("parity\_right : %c,%c,%c,%c,%c,%c\n",parity\_right[1],parity\_right[2],parity\_right[3],parity\_right[4],parity\_right[5],parity\_right[6]);

}

printf("#############################################################\n");

printf("13 number : %d\n",NUM\_13);

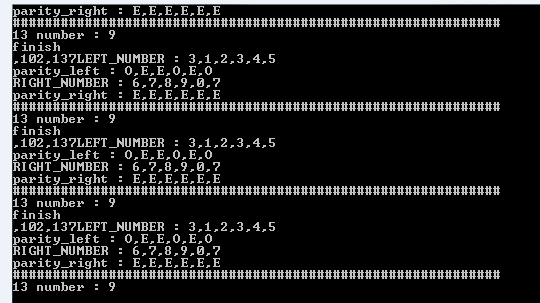
cvShowImage( "Camera image", frame );

}while ('q'!=cvWaitKey(10));

# 5. Results

We tested the programming using the two bar codes found in the lab note. Six tests were conducted and their results are shown in this section.

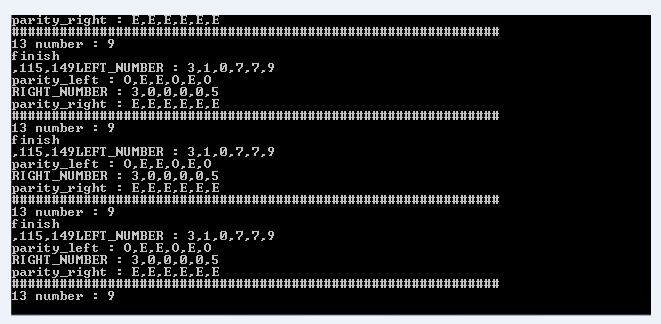
## Test 1



**Figure 6: Results of Test 1**

In test 1, the printed barcode is 9 312345 678907, as shown above. The program was able to read the barcode correctly. The number and their parities were calculated and displayed on the monitor screen as indicated by the right picture above. The parities of the right bits, i.e. 678907, were all even. However, the parities of the left bits vary from barcode to barcode and they are used to determine the first digit of the barcode. In this test, the parities for left digits are ‘OEEOEO’, in which character ‘O’ represents odd parity and character ‘E’ represents even parity. If we look up the table of barcode parity for EAN-13, this combination corresponds to the number 9.

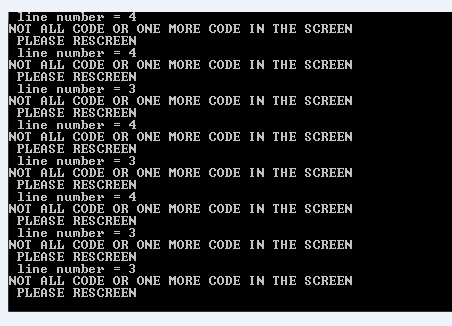
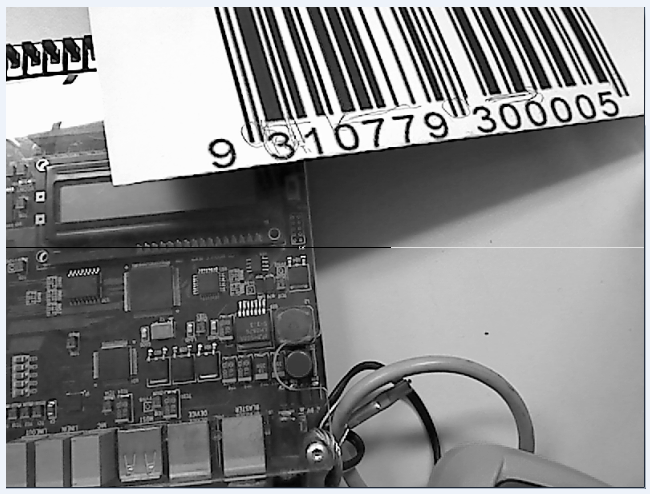
## Test 2



**Figure 7: Results of Test 2**

In test 2, the printed barcode is 9 310779 300005 shown on the top right picture. The program correctly displayed the barcode on the monitor screen. In a similar manner, the number and their parities were calculated and displayed. The parity combination was the same as the one in test and hence the first digit is 9.

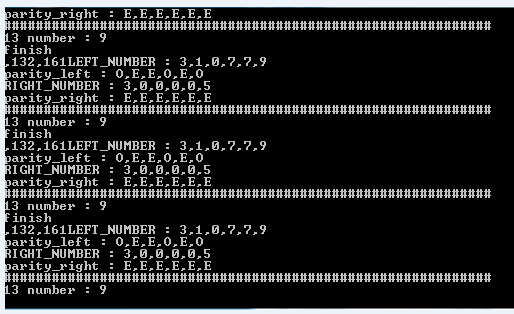
## Test 3



**Figure 8: Results of Test 3**

The purpose of test 3 is to show that the barcodes have to be placed within certain region in order to be read by the USB camera. As can be seen from the pictures above, the bar code was placed way above the central line. As a result, the program could not read the barcode and advised the user to relocate the barcode. Having tried multiple locations, we found the barcode should be placed in regions where the central line of the USB can cut through all bars of the barcode.

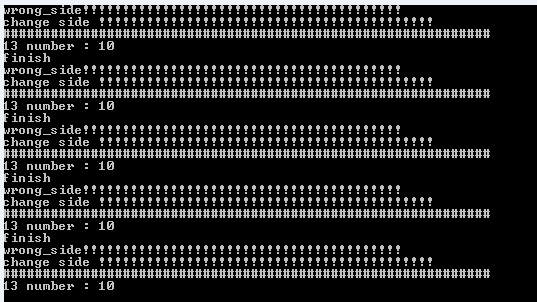
## Test 4



**Figure 9: Results of Test 4**

Following test 3, we would like to investigate whether orientations of the barcode have any effect on readings. With the central line cutting through all bars, we rotated the barcode paper counter clockwise to approximately 60 degrees. It had been found that during the rotation process the reading did not change.

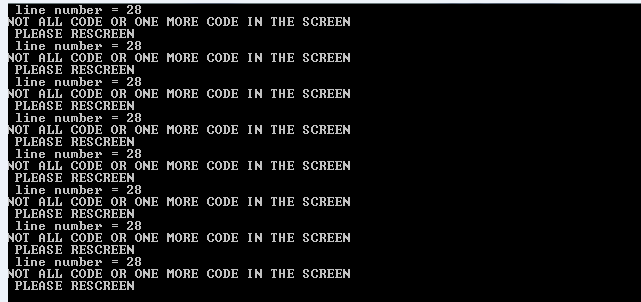
## Test 5



**Figure 11: Results of Test 5**

We kept increase the rotation angle until the barcode was reversed in orientation. The program was able to detect this reversal and advised the user to ‘change side’.

## Test 6

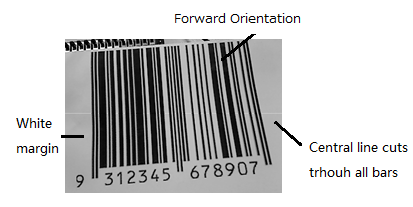
 

**Figure 12: Results of Test 6**

The last test was done to demonstrate the importance of the left most guard code and the white margin to the left of the whole barcode. As shown in the figure, the USB camera was unable to sense the guard code and the white margin. As a consequence, the program could not read the barcode and it warned the reposition the barcode.

# 6. Discussion

The barcode detecting program samples the barcode image from the middle, so the middle line should cut all the bars in a barcode. The reason of sampling this way is to increase the resistivity of interference that may present on the code, such as dirt. Specifically, if there is a blot on the barcode, we can simply reposition the barcode into a location that all its bars are scanned by the central line. Nevertheless, it does not mean that the barcode can be put randomly under the camera, as demonstrated by test 3, 5 and 6. The acceptable way to position the barcode is illustrated below



**Figure 13: Acceptable barcode positions**

The white margin to the left of the barcode is needed because we coded the system in a way that we detect positive edges (or 0-> 1 transition) for the left digits. If the image starts with black (e.g. a blot of dirt or ink), the first 0 ->1 transition will not be detected and we will therefore miss the first bar. This problem can be addressed by measuring the width of the black blot. If the width is wider than a pre-set threshold value, the detecting system will regard it as interference and replace it with 0 instead of 1. In this way, the program will see the black blot as if it does not exist.

It has been shown in the test results that the EAN-13 barcode is relative resilient to the skewness of barcodes. However, the barcodes still need to be put within certain areas under the USB camera so as to be read correctly. We can conclude that as long as the barcode is not in the reversed orientation, the reading will not change even if the barcodes are tilted. If it is in the reverse orientation, the program will recognise it by detecting the parities of the right digits to be a mix of odd and even, which violated the EAN-13 rule that states all the right bits should be even

The importance of camera barcode detection has grown these days with the increases in possession of mobile phones. With the camera on mobile phones, people can easily scan a barcode and acquire the information it carries. The more advanced and prevalent technique today involves 2D matrices that carry more data than their 1D counterpart [3].

# 7. Conclusion

In this lab practice, a modified program was composed to read barcodes using an USB camera. The program was written in C++ with the aid of a skeleton code and OpenCV library. The code functioned well with the barcodes in the lab notes. It can correctly read the 10 digits with their parities and calculate the 13th bit basing on the parities of the left digits. In addition, several tests had been conducted to verify the validity and to explore the characteristic of the detecting program. It has been found that barcodes need to be place in certain positions in order to be read correctly by the camera.

# Reference

[1] A. Russell, “Monash TRC3500 Lecture notes”, Monash University.

[2] A. Russell, “TRC3500 Sensors and Artificial Perception laboratory manual”, Monash University

[3] David L. Hecht. "Printed Embedded Data Graphical User Interfaces". Xerox Palo Alto Research Center. IEEE Computer March 2001.

# Appendix

////////////////////////////////////////////////////////////////

// Skeleton program for TRC3500

// Grabs images from a USB camera using OpenCV

// Written by Andy Russell 09th February 2006

// Modified by Michael Curtis 2011-2012 - updated for newer OpenCV

/////////////////////////////////////////////////////////////////

#include "cv.h"

#include "highgui.h"

#include <iostream>

#include <stdio.h>

#include <math.h>

using namespace std;

int scanrow[640] = {0},

EDGE\_LINE\_GROUP[60] = {0},

code\_left\_1[4] = {0},

code\_left\_2[4] = {0},

code\_left\_3[4] = {0},

code\_left\_4[4] = {0},

code\_left\_5[4] = {0},

code\_left\_6[4] = {0},

code\_right\_1[4] = {0},

code\_right\_2[4] = {0},

code\_right\_3[4] = {0},

code\_right\_4[4] = {0},

code\_right\_5[4] = {0},

code\_right\_6[4] = {0},

LEFT\_NUMBER[6] = {0},

RIGHT\_NUMBER[6] = {0};

//array initilization

char parity\_left[6],parity\_right[6];

int digit13(char par1, char par2,char par3,char par4,char par5,char par6);

void COMPARE\_TABLE(int T1\_width, int T2\_width, int T4\_width ,int \*character, char \*parity);

void RightDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int endP);

void LeftDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int beginP);

int T\_Width(float Ti, float T);

// main - initialises OpenCV and captures an image and changes it

int main( )

{

CvCapture\* capture = 0;

IplImage\* processedImage = 0;

cout<<"Andy's USB camera program"<<endl<<"Press any key to capture image for scan"<<endl;

cout<<"Press and hold 'q' to quit"<<endl;

// Initializes capturing video from camera

capture = cvCaptureFromCAM( -1 );

if( !capture ) {

fprintf(stderr,"Could not initialize capturing...\n");

return -1;

}

// Creates window

cvNamedWindow( "Camera image", 1 );

// Camera image

IplImage\* frame = 0;

// Grabs and returns a frame from camera

frame = cvQueryFrame( capture );

// Print details of image

cout<<"image width ="<<frame->width<<" height ="<<frame->height;

cout<<" depth ="<<frame->depth<<" channels ="<<frame->nChannels<<endl;

int threshold=130;

int black=0;

int white=255;

do {

cvWaitKey(100);

// Grabs and returns a frame from camera

frame = cvQueryFrame( capture );

if( !frame ) {

break;

}

int x;

int y;

// Convert half of the image to gray

for( y = 0; y < frame->height; y++) {

for( x = 0; x < frame->width; x++) {

// This is a pointer to the start of the current row.

// Note: The image is stored as a 1-D array which is mapped back

// into 2-space by multiplying the widthStep (the image width rounded to

// a "nice" value, eg a multiple of 4 or 8 depending on the OS and CPU)

// by the row number.

uchar \*row = (uchar\*)(frame->imageData + frame->widthStep \* y );

int gray = ( row[ x\*3 ] + row[ x\*3+1 ] + row[ x\*3+2 ] ) / 3;

if (y == (frame->height/2)){ //apply thresholding to image

if (gray <= threshold )

{

scanrow[x]=1;

//printf(",%d",x);

gray = 0;

}else{

//printf("%d",0);

scanrow[x]=0;

gray = 1;.

}

}

row[ x\*3 ] = gray;

row[ x\*3 + 1 ] = gray;

row[ x\*3 + 2 ] = gray;

}

}

int i = 0;

int falledge = 0;

for(i = 0 ;i<640 ; i++)

{

if (((scanrow[i]==0)&&(scanrow[i-1]==1))|((scanrow[i]==1)&&(scanrow[i-1]==0))){

falledge+=1;

EDGE\_LINE\_GROUP[falledge] = i;

}

}

if (falledge!=60){

int RECREEN = 1;

printf("line number = %d\n",falledge/2);

printf("NOT ALL CODE OR ONE MORE CODE IN THE SCREEN\n ");

printf("PLEASE RESCREEN\n ");

cvShowImage( "Camera image", frame );

continue;

}else int RECREEN = 0;

printf("finish\n");

int a = 5;

////////////////////left/////////////////////////////////

for(i=0; i<4; i++){

code\_left\_1[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_left\_2[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_left\_3[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_left\_4[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_left\_5[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_left\_6[i]=EDGE\_LINE\_GROUP [a];

a++;

}

a = 33;

////////////////right///////////////////////////////////////

for(i=0; i<4; i++){

code\_right\_1[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_right\_2[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_right\_3[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_right\_4[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_right\_5[i]=EDGE\_LINE\_GROUP [a];

a++;

}

for(i=0; i<4; i++){

code\_right\_6[i]=EDGE\_LINE\_GROUP [a];

a++;

}

RightDigit(1, code\_right\_1[0], code\_right\_1[1], code\_right\_1[2], code\_right\_1[3], code\_right\_2[0]);

RightDigit(2, code\_right\_2[0], code\_right\_2[1], code\_right\_2[2], code\_right\_2[3], code\_right\_3[0]);

RightDigit(3, code\_right\_3[0], code\_right\_3[1], code\_right\_3[2], code\_right\_3[3], code\_right\_4[0]);

RightDigit(4, code\_right\_4[0], code\_right\_4[1], code\_right\_4[2], code\_right\_4[3], code\_right\_5[0]);

RightDigit(5, code\_right\_5[0], code\_right\_5[1], code\_right\_5[2], code\_right\_5[3], code\_right\_6[0]);

RightDigit(6, code\_right\_6[0], code\_right\_6[1], code\_right\_6[2], code\_right\_6[3], EDGE\_LINE\_GROUP [57]);

LeftDigit(1, code\_left\_1[0], code\_left\_1[1], code\_left\_1[2], code\_left\_1[3], EDGE\_LINE\_GROUP[4] );

LeftDigit(2, code\_left\_2[0], code\_left\_2[1], code\_left\_2[2], code\_left\_2[3], code\_left\_1[3] );

LeftDigit(3, code\_left\_3[0], code\_left\_3[1], code\_left\_3[2], code\_left\_3[3], code\_left\_2[3] );

LeftDigit(4, code\_left\_4[0], code\_left\_4[1], code\_left\_4[2], code\_left\_4[3], code\_left\_3[3] );

LeftDigit(5, code\_left\_5[0], code\_left\_5[1], code\_left\_5[2], code\_left\_5[3], code\_left\_4[3] );

LeftDigit(6, code\_left\_6[0], code\_left\_6[1], code\_left\_6[2], code\_left\_6[3], code\_left\_5[3] );

int NUM\_13 = digit13(parity\_left[1],parity\_left[2],parity\_left[3],parity\_left[4],parity\_left[5],parity\_left[6]);

if (NUM\_13 == 10){

printf("wrong\_side!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");

printf("change side !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");

}else{

printf("LEFT\_NUMBER : %d,%d,%d,%d,%d,%d\n",LEFT\_NUMBER[1],LEFT\_NUMBER[2],LEFT\_NUMBER[3],LEFT\_NUMBER[4],LEFT\_NUMBER[5],LEFT\_NUMBER[6]);

printf("parity\_left : %c,%c,%c,%c,%c,%c\n",parity\_left[1],parity\_left[2],parity\_left[3],parity\_left[4],parity\_left[5],parity\_left[6]);

printf("RIGHT\_NUMBER : %d,%d,%d,%d,%d,%d\n",RIGHT\_NUMBER[1],RIGHT\_NUMBER[2],RIGHT\_NUMBER[3],RIGHT\_NUMBER[4],RIGHT\_NUMBER[5],RIGHT\_NUMBER[6]);

printf("parity\_right : %c,%c,%c,%c,%c,%c\n",parity\_right[1],parity\_right[2],parity\_right[3],parity\_right[4],parity\_right[5],parity\_right[6]);

}

printf("#############################################################\n");

printf("13 number : %d\n",NUM\_13);

cvShowImage( "Camera image", frame );

}while ('q'!=cvWaitKey(10));

// Releases the CvCapture structure

cvReleaseCapture( &capture );

// Destroys all the HighGUI windows

cvDestroyAllWindows( );

return 0;

}

void LeftDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int beginP){

float T1,T2,T4;

float T = LINE\_2\_F-beginP;

T1 = LINE\_2\_F - LINE\_1\_F;

T2 = LINE\_2\_R - LINE\_1\_R;

T4 = LINE\_1\_R - beginP;

int T1\_width=T\_Width(T1,T);

int T2\_width=T\_Width(T2,T);

int T4\_width=T\_Width(T4,T);

COMPARE\_TABLE(T1\_width, T2\_width, T4\_width , &LEFT\_NUMBER[number], &parity\_left[number]);

}

void RightDigit(int number,int LINE\_1\_R,int LINE\_1\_F,int LINE\_2\_R,int LINE\_2\_F,int endP){

float T1,T2,T4;

float T = endP - LINE\_1\_R;

T1 = LINE\_2\_R - LINE\_1\_R;

T2 = LINE\_2\_F - LINE\_1\_F;

T4 = endP - LINE\_2\_F;

int T1\_width=T\_Width(T1,T);

int T2\_width=T\_Width(T2,T);

int T4\_width=T\_Width(T4,T);

COMPARE\_TABLE(T1\_width, T2\_width, T4\_width , &RIGHT\_NUMBER[number], &parity\_right[number]);

}

void COMPARE\_TABLE(int T1\_width, int T2\_width, int T4\_width ,int \*character, char \*parity){

if(T1\_width==2&&T2\_width==2){

\*character=6;

\*parity='E';

}

if(T1\_width==2&&T2\_width==3){

\*character=0;

\*parity='O';

}

if(T1\_width==2&&T2\_width==4){

\*character=4;

\*parity='E';

}

if(T1\_width==2&&T2\_width==5){

\*character=3;

\*parity='O';

}

if(T1\_width==3&&T2\_width==2){

\*character=9;

\*parity='O';

}

if(T1\_width==3&&T2\_width==3&&T4\_width==2){

\*character=2;

\*parity='E';

}

if(T1\_width==3&&T2\_width==3&&T4\_width==3){

\*character=8;

\*parity='E';

}

if(T1\_width==3&&T2\_width==4&&T4\_width==2){

\*character=1;

\*parity='O';

}

if(T1\_width==3&&T2\_width==4&&T4\_width==1){

\*character=7;

\*parity='O';

}

if(T1\_width==3&&T2\_width==5){

\*character=5;

\*parity='E';

}

if(T1\_width==4&&T2\_width==2){

\*character=9;

\*parity='E';

}

if(T1\_width==4&&T2\_width==3&&T4\_width==2){

\*character=2;

\*parity='O';

}

if(T1\_width==4&&T2\_width==3&&T4\_width==1){

\*character=8;

\*parity='O';

}

if(T1\_width==4&&T2\_width==4&&T4\_width==1){

\*character=1;

\*parity='E';

}

if(T1\_width==4&&T2\_width==4&&T4\_width==2){

\*character=7;

\*parity='E';

}

if(T1\_width==4&&T2\_width==5){

\*character=5;

\*parity='O';

}

if(T1\_width==5&&T2\_width==2){

\*character=6;

\*parity='O';

}

if(T1\_width==5&&T2\_width==3){

\*character=0;

\*parity='E';

}

if(T1\_width==5&&T2\_width==4){

\*character=4;

\*parity='O';

}

if(T1\_width==5&&T2\_width==5){

\*character=3;

\*parity='E';

}

}

int T\_Width(float Ti, float T)

{

int width=0;

if((Ti/T)<=(3.0/14)){

width=1;

}

if((3.0/14)<(Ti/T)&&(Ti/T)<=(5.0/14)){

width=2;

}

if((5.0/14)<(Ti/T)&&(Ti/T)<=(7.0/14)){

width=3;

}

if((7.0/14)<(Ti/T)&&(Ti/T)<=(9.0/14)){

width=4;

}

if((9.0/14)<(Ti/T)&&(Ti/T)<=(11.0/14)){

width=5;

}

return width;

}

int digit13(char par1,char par2,char par3,char par4,char par5,char par6)

{

//the 1st parity will always be zero, so ignore 1st parity

if((par2=='O')&&(par3=='O')&&(par4=='O')&&(par5=='O')&&(par6=='O'))

{

return 0;

}

if((par2=='O')&&(par3=='E')&&(par4=='O')&&(par5=='E')&&(par6=='E'))

{

return 1;

}

if((par2=='O')&&(par3=='E')&&(par4=='E')&&(par5=='O')&&(par6=='E'))

{

return 2;

}

if((par2=='O')&&(par3=='E')&&(par4=='E')&&(par5=='E')&&(par6=='O'))

{

return 3;

}

if((par2=='E')&&(par3=='O')&&(par4=='O')&&(par5=='E')&&(par6=='E'))

{

return 4;

}

if((par2=='E')&&(par3=='E')&&(par4=='O')&&(par5=='O')&&(par6=='E'))

{

return 5;

}

if((par2=='E')&&(par3=='E')&&(par4=='E')&&(par5=='O')&&(par6=='O'))

{

return 6;

}

if((par2=='E')&&(par3=='O')&&(par4=='E')&&(par5=='O')&&(par6=='E'))

{

return 7;

}

if((par2=='E')&&(par3=='O')&&(par4=='E')&&(par5=='E')&&(par6=='O'))

{

return 8;

}

if((par2=='E')&&(par3=='E')&&(par4=='O')&&(par5=='E')&&(par6=='O'))

{

return 9;

}

if((par2=='E')&&(par3=='E')&&(par4=='E')&&(par5=='E')&&(par6=='E'))

{

return 10;

}

}