**======The proposed real code of IEE process=====**

**Code is provided in five sections:**

**1. Call header files**

**2. The adaptive Threshold function and neighboring boundaries definitions are clearly mentioned.**

**3. Thresholding implementation to the image is provided.**

**4. The Main function; related frames/ windows of obtained images to be defined are available.**

**5. IEE process is mentioned, mask movement, and conditions applied.**

//Image Edges Exaction (IEE) Process

//detecting vertical edges, two pixel width for odd edges and one pixel width for even edges

#include <stdlib.h>

#include <stdio.h>

#include <cv.h>

#include <highgui.h>

#include <sys/types.h>

#include <time.h>

#define T (0.15f)

void adaptiveThreshold(unsigned char\* input, unsigned char\* bin, int widthOriginal, int heightOriginal)

{

unsigned long\* integralImg = 0;

int i, j;

long sum=0;

int count=0;

int index;

int x1, y1, x2, y2;

int S=widthOriginal/8;

int s2 = S/2;

// create the integral image

integralImg = (unsigned long\*)malloc(widthOriginal\*heightOriginal\*sizeof(unsigned long\*));

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

{

// reset this column sum

sum = 0;

for (j=0; j<heightOriginal; j++)

{

index = j\*widthOriginal+i;

sum += input[index];

if (i==0)

integralImg[index] = sum;

else

integralImg[index] = integralImg[index-1] + sum;

}

}

// perform thresholding

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

{

for (j=0; j<heightOriginal; j++)

{

index = j\*widthOriginal+i;

// set the SxS region

x1=i-s2; x2=i+s2;

y1=j-s2; y2=j+s2;

// check the border

if (x1 < 0) x1 = 0;

if (x2 >= widthOriginal) x2 = widthOriginal-1;

if (y1 < 0) y1 = 0;

if (y2 >= heightOriginal) y2 = heightOriginal-1;

count =S\*S; //(x2-x1)\*(y2-y1);

// I(x,y)=s(x2,y2)-s(x1,y2)-s(x2,y1)+s(x1,x1)

sum = integralImg[y2\*widthOriginal+x2] -

integralImg[y1\*widthOriginal+x2] -

integralImg[y2\*widthOriginal+x1] +

integralImg[y1\*widthOriginal+x1];

if ((long)(input[index]\*count) < (long)(sum\*(1.0-T)))

// --count-- is replaced by --(S\*S-0.5\*S\*S-2\*S)--

bin[index] = 0;

//else bin[index] = 255;

}

}

free (integralImg);

}

main ()

{

IplImage\* cvFrame;

IplImage\* binImg;

int key;

cvFrame = cvLoadImage("9n.jpg", 0);

//binImg = cvCreateImage(cvSize(IMAGE\_WIDTH, IMAGE\_HEIGHT), 8, 1);

// Clone, and make binImg(output image) white

binImg=cvCloneImage(cvFrame);

uchar \*dataOP=(uchar \*)binImg->imageData;

int stepOP=binImg->widthStep;

for(int a=0; a<binImg->height; a++)

for( int b=0; b<binImg->width; b++)

dataOP[a\*stepOP+b]=255;

cvNamedWindow("Gray scale image", 1);

cvNamedWindow("Adaptive", 1);

adaptiveThreshold((unsigned char\*)cvFrame->imageData,(unsigned char\*)binImg->imageData, cvFrame->width, cvFrame->height);

cvShowImage("Gray scale image", cvFrame);

cvShowImage("Adaptive", binImg);

cvNamedWindow("Output2", 1);

cvShowImage("Output2", binImg);

IEE\_process();

}

//end main

IEE\_process() {

cvNamedWindow("IEE", 1); // *IEE* is a window on which the edges extracted images is placed

int rght=1,lft=1;

for(int a=0; a<binImg->height; a++)

for( int b=0; b<binImg->width; b++) {

cntr=1; rght=1; lft=1;

if ( !dataOP[a\*stepOP+b] & !dataOP[a\*stepOP+(b+1)] & !dataOP[(a+1)\*stepOP+b] & !dataOP[(a+1)\*stepOP+(b+1)] ) //main col

cntr=0;

if ( (dataOP[a\*stepOP+(b+2)] + dataOP[(a+1)\*stepOP+(b+2)])==0 ) //Next col

rght=0;

if( (dataOP[a\*stepOP+(b-1)] + dataOP[(a+1)\*stepOP+(b-1)])==0 ) //Previous col

lft=0;

if(!cntr &!rght &!lft){

data3[a\*step3+b]=255;

data3[(a+1)\*step3+b]=255;

}

}

cvShowImage("VEDA", Img3);

cvWaitKey();

}

// end of IEE\_process function