**IMAGE SHOW**

#include "stdafx.h"

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <stdio.h>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\picture000.jpg", CV\_LOAD\_IMAGE\_COLOR);

namedWindow("my window", CV\_WINDOW\_AUTOSIZE);

imshow("my window", image);

waitKey();

return 0;

}

**RGB**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat image(200, 300, CV\_8UC3, Scalar(0, 0, 255));

char win[] = "Striped Image";

int location;

for (int i = 0; i<image.rows; i++)

{

for (int j = 0; j<image.cols; j++)

{

image.at<Vec3b>(i, j)[0] = 0;

image.at<Vec3b>(i, j)[1] = 0;

image.at<Vec3b>(i, j)[2] = 0;

if (i<image.rows / 3)

image.at<Vec3b>(i, j)[2] = 255;

else if (i >= (2 \* image.rows) / 3)

image.at<Vec3b>(i, j)[0] = 255;

else

image.at<Vec3b>(i, j)[1] = 255;

}

}

namedWindow(win, CV\_WINDOW\_FULLSCREEN);

imshow(win, image);

waitKey(0);

return 0;

}

**RGB TO BINARY**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\picture000.jpg", CV\_LOAD\_IMAGE\_COLOR);

char win[] = "Black and White";

int t = 382;

for (int i = 0; i<image.rows; i++)

{

for (int j = 0; j<image.cols; j++)

{

if ((image.at<Vec3b>(i, j)[0] + image.at<Vec3b>(i, j)[1] + image.at<Vec3b>(i, j)[2])>t)

{

image.at<Vec3b>(i, j)[0] = 255;

image.at<Vec3b>(i, j)[1] = 255;

image.at<Vec3b>(i, j)[2] = 255;

}

else

{

image.at<Vec3b>(i, j)[0] = 0;

image.at<Vec3b>(i, j)[1] = 0;

image.at<Vec3b>(i, j)[2] = 0;

}

}

}

namedWindow(win, CV\_WINDOW\_FULLSCREEN);

imshow(win, image);

waitKey(0);

return 0;

}

**RGB TO BINARY USING UCHAR**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\picture001.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat image1 (image.rows,image.cols,CV\_8UC1,Scalar (0,10,20));

char win[] = "Black and White";

int t = 382;

for (int i = 0; i<image.rows; i++)

{

for (int j = 0; j<image.cols; j++)

{

if ((image.at<Vec3b>(i, j)[0] + image.at<Vec3b>(i, j)[1] + image.at<Vec3b>(i, j)[2])>t)

{

image1.at<uchar>(i, j)= 255;

}

else

{

image1.at<uchar>(i, j)= 0;

}

}

}

namedWindow(win,WINDOW\_AUTOSIZE);

imshow(win, image1);

waitKey(0);

return 0;

}

**EDGE DETECTION**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\picture002.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat image1(image.rows, image.cols, CV\_8UC1, Scalar(0));

Mat image2(image.rows, image.cols, CV\_8UC1, Scalar(0));

int t = 100, i, j, max, min, k, l;

for (int i = 0; i < image.rows; i++)

{

for (int j = 0; j<image.cols; j++)

{

if ((image.at<Vec3b>(i, j)[0] + image.at<Vec3b>(i, j)[1] + image.at<Vec3b>(i, j)[2])>t)

{

image1.at<uchar>(i, j) = 255;

}

else

{

image1.at<uchar>(i, j) = 0;

}

}

}

for (i = 1; i < (image1.rows - 1); i++)

{

for (j = 1; j < (image1.cols - 1); j++)

{

max = 0; min = 255;

for (k = (i - 1); k <= (i + 1); k++)

{

for (l = (j - 1); l <= (j + 1); l++)

{

if (k!= l)

{

if (max < image1.at<uchar>(k, l))

max = image1.at<uchar>(k, l);

if (min > image1.at<uchar>(k, l))

min = image1.at<uchar>(k, l);

}

}

}

if ((max - min) > 100)

image2.at<uchar>(i, j) = 255;

else

image2.at<uchar>(i, j) = 0;

}

}

namedWindow("Original", WINDOW\_AUTOSIZE);

imshow("Original", image);

namedWindow("Binary Image1", WINDOW\_AUTOSIZE);

imshow("Binary Image1", image1);

namedWindow("Edge Detection", WINDOW\_AUTOSIZE);

imshow("Edge Detection", image2);

waitKey();

return 0;

}

**Track Bar**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include <iostream>

using namespace std;

using namespace cv;

int main()

{

Mat image = imread("F:\picture001.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat grey(image.rows, image.cols, CV\_8UC1);

Mat binary(image.rows, image.cols, CV\_8UC1);

int i, j, thres;

int b, g, r;

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

b = image.at<Vec3b>(i, j)[0];

g = image.at<Vec3b>(i, j)[1];

r = image.at<Vec3b>(i, j)[2];

grey.at<uchar>(i, j) = (b + g + r) / 3;

}

}

namedWindow("Input", CV\_WINDOW\_AUTOSIZE);

namedWindow("Grey", CV\_WINDOW\_AUTOSIZE);

imshow("Input", image);

imshow("Grey", grey);

namedWindow("Binary", CV\_WINDOW\_AUTOSIZE);

cvCreateTrackbar("Threshold", "Binary", &thres, 255);

while (1)

{

for (i = 0; i < image.rows; ++i)

{

for (j = 0; j < image.cols; ++j)

{

if (grey.at<uchar>(i, j)>thres)

binary.at<uchar>(i, j) = 255;

else

binary.at<uchar>(i, j) = 0;

}

}

imshow("Binary", binary);

int ikey = waitKey(50);

if (ikey == 27)

break;

}

waitKey(0);

return 0;

}

**ADAPTIVE:**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\picture001.jpg", CV\_LOAD\_IMAGE\_GRAYSCALE);

Mat image1(image.rows, image.cols, CV\_8UC1, Scalar(0));

char win[] = "Black and White"; //String with the name of the window

int i, j;

for (int i = 1; i<(image.rows - 1); i++)

{

for (int j = 1; j<(image.cols - 1); j++)

{

if (((image.at<uchar>(i - 1, j) + image.at<uchar>(i - 1, j - 1) + image.at<uchar>(i - 1, j + 1) + image.at<uchar>(i, j - 1)) + image.at<uchar>(i, j + 1) + image.at<uchar>(i + 1, j - 1) + image.at<uchar>(i + 1, j) + image.at<uchar>(i + 1, j + 1)) / 8> image.at<uchar>(i, j))

{

image1.at<uchar>(i, j) = 255;

}

else

{

image1.at<uchar>(i, j) = 0;

}

}

}

for (i = 0; i < image.rows; i = i + (image.rows - 1))

{

for (int j = 1; j < (image.cols - 1); j++)

{

if (((image.at<uchar>(i, j + 1) + image.at<uchar>(i, j - 1) + image.at<uchar>(i + 1, j + 1) + image.at<uchar>(i + 1, j - 1)) + image.at<uchar>(i + 1, j)) / 5 > image.at<uchar>(i, j))

{

image1.at<uchar>(i, j) = 255;

}

else

{

image1.at<uchar>(i, j) = 0;

}

}

}

for (j = 0; j < image.cols; j = j + (image.cols - 1))

{

for (int i = 1; i < (image.rows - 1); i++)

{

if (((image.at<uchar>(i + 1, j) + image.at<uchar>(i - 1, j) + image.at<uchar>(i + 1, j + 1) + image.at<uchar>(i - 1, j + 1)) + image.at<uchar>(i, j + 1)) / 5 > image.at<uchar>(i, j))

{

image1.at<uchar>(i, j) = 255;

}

else

{

image1.at<uchar>(i, j) = 0;

}

}

}

namedWindow(win, WINDOW\_AUTOSIZE);

imshow(win, image1);

waitKey();

return 0;

}

**VIDEOREAD**

#include "stdafx.h"

#include <iostream>

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace std;

using namespace cv;

int main()

{

int ikey;

VideoCapture vid(0);

while (1)

{

Mat temp;

vid.read(temp);

namedWindow("Video", WINDOW\_AUTOSIZE);

imshow("Video", temp);

ikey = waitKey(50);

if (ikey == 27)

break;

}

}

**VIDEO SEEKER**

#include "stdafx.h"  
#include<opencv2\core\core.hpp>  
#include<opencv2\highgui\highgui.hpp>  
#include<opencv2\imgproc\imgproc.hpp>  
#include<iostream>  
#include<stdio.h>  
#include<algorithm>  
using namespace std;  
using namespace cv;  
int main()  
{  
VideoCapture vid("F:\\alcoholic.mp4");  
namedWindow("Output", CV\_WINDOW\_AUTOSIZE);  
int thres;  
createTrackbar("Track", "Output", &thres,60\*1000\*3);  
while (1)  
{  
Mat image;  
vid.read(image);  
imshow("Output", image);  
double thres1 = thres;  
double pos = vid.get(CV\_CAP\_PROP\_POS\_MSEC);  
setTrackbarPos("Track", "Output", thres);  
vid.set(CV\_CAP\_PROP\_POS\_MSEC,thres1);  
if (waitKey(30) == 27)  
break;  
  
}  
return 0;  
}

**COLUR DETECTION**

#include "stdafx.h"

#include<opencv2\core\core.hpp>

#include<opencv2\highgui\highgui.hpp>

#include<iostream>

#include<stdio.h>

using namespace std;

using namespace cv;

int main()

{

Mat image;

image = imread("F:\\picture001.jpg", CV\_LOAD\_IMAGE\_COLOR);

Mat image1(image.rows, image.cols, CV\_8UC1, Scalar(0));

int t1 =100, t2 = 100, i, j;

for (i = 0; i < image.rows; i++)

{

for (j = 0; j < image.cols; j++)

{

if ((image.at<Vec3b>(i, j)[2]>t1) && (image.at<Vec3b>(i, j)[1] < t2) && (image.at<Vec3b>(i, j)[0] < t2))

image1.at<uchar>(i, j) = 255;

else

image1.at<uchar>(i, j) = 0;

}

}

namedWindow("Colour", WINDOW\_AUTOSIZE);

imshow("Colour", image1);

waitKey(0);

return 0;

}

**HSV**

#include"stdafx.h"  
#include<opencv2\core\core.hpp>  
#include "opencv2/imgproc/imgproc.hpp"  
#include<opencv2\highgui\highgui.hpp>  
#include <iostream>  
  
using namespace std;  
using namespace cv;  
  
  
int main()  
{  
Mat image = imread("F:\\picture000.jpg", CV\_LOAD\_IMAGE\_COLOR);  
Mat hsv(image.rows, image.cols, CV\_8UC3);  
Mat color(image.rows, image.cols, CV\_8UC1);  
cvtColor(image, hsv, CV\_BGR2HSV);  
int h,s,v, i, j;  
for (i = 0; i < image.rows; i++)  
{  
for (j = 0; j < image.cols; j++)  
{  
v = hsv.at<Vec3b>(i, j)[2];  
h = hsv.at<Vec3b>(i, j)[0];  
s = hsv.at<Vec3b>(i, j)[1];  
if ( h>150 && h<170 && s>200 )  
color.at<uchar>(i, j) = 255;  
else  
color.at<uchar>(i, j) = 0;  
}  
}  
namedWindow("hsv", WINDOW\_AUTOSIZE);  
imshow("hsv", hsv);  
namedWindow("colordetect", WINDOW\_AUTOSIZE);  
imshow("colordetect", color);  
namedWindow("original", WINDOW\_AUTOSIZE);  
imshow("original", image);  
waitKey(0);  
  
}

**BLOB DETECTION**

#include "stdafx.h"  
#include<opencv2\core\core.hpp>  
#include<opencv2\highgui\highgui.hpp>  
#include<opencv2\imgproc\imgproc.hpp>  
#include <iostream>  
#include<queue>  
  
using namespace std;  
using namespace cv;  
  
typedef struct {  
int x, y;  
}point;  
  
Mat binary(Mat image)  
{  
Mat image2(image.rows, image.cols, CV\_8UC1);  
int i, j;  
for (i = 0; i < image.rows; i++)  
{  
for (j = 0; j < image.cols; j++)  
{  
if ((image.at<Vec3b>(i, j)[0] + image.at<Vec3b>(i, j)[1] + image.at<Vec3b>(i, j)[2]) / 3 < (60))  
image2.at<uchar>(i, j) = 0;  
else  
{  
image2.at<uchar>(i, j) = 255;  
}  
  
}  
  
}  
  
return image2;  
}  
  
  
  
int main()  
{  
Mat image = imread("F:\\picture000.jpg", CV\_LOAD\_IMAGE\_COLOR);  
Mat bimage(image.rows, image.cols, CV\_8UC1);  
Mat help(image.rows, image.cols, CV\_8UC1, Scalar(0));  
Mat check(image.rows, image.cols, CV\_8UC1, Scalar(0));  
bimage = binary(image);  
namedWindow("binary", WINDOW\_AUTOSIZE);  
imshow("binary", bimage);  
int r = image.rows, c = image.cols, count = 0, a, b;  
queue<point> q;  
for (int i = 1; i < r - 1; i++)  
{  
for (int j = 1; j < c - 1; j++)  
{  
if (bimage.at<uchar>(i, j) == 0)  
help.at<uchar>(i, j) = 0;  
  
else  
{  
if (check.at<uchar>(i, j) == 0)  
{  
  
point p;  
p.x = i;  
p.y = j;  
q.push(p);  
help.at<uchar>(i, j) = count + 1;  
check.at<uchar>(i, j) = 1;  
while (!q.empty())  
{  
point pnt = q.front();  
a = pnt.x;  
b = pnt.y;  
for (int k = a - 1; k <= a + 1; k++)  
{  
for (int l = b - 1; l <= b + 1; l++)  
{  
if (bimage.at<uchar>(k, l) == 255)  
{  
if (check.at<uchar>(k, l) == 0)  
{  
point p;  
p.x = k;  
p.y = l;  
  
q.push(p);  
help.at<uchar>(k, l) = count + 1;  
check.at<uchar>(k, l) = 1;  
}  
}  
}  
}  
q.pop();  
}  
count++;  
}  
}  
}  
}  
  
  
cout << count << "\n";  
waitKey(0);  
return 0;  
}

**HOUGH TRANSFORM**

void HoughTransform(Mat image)  
{  
int i;  
Mat edges(image.rows, image.cols, CV\_8UC1,Scalar(0));  
Canny(image, edges, 50, 150);  
namedWindow("Edges", CV\_WINDOW\_NORMAL);  
imshow("Edges", edges);  
int rho = 4, theta = 4;  
namedWindow("Detected Lines", CV\_WINDOW\_NORMAL);  
createTrackbar("Rho (0.25\*x)", "Detected Lines", &rho, 20);  
createTrackbar("\nTheta (.25\*x)", "Detected Lines", &theta, 40);  
while (1)  
{  
vector<Vec2f> lines;  
HoughLines(edges, lines, 0.25\*rho, .25\*theta\*(CV\_PI / 180), 100);  
Mat linesimg(image.rows, image.cols, CV\_8UC1, Scalar(0));  
for (i = 0; i < lines.size(); i++)  
{  
float rho = lines[i][0];  
float theta = lines[i][1];  
double a = cos(theta), b = sin(theta);  
double x0 = a\*rho, y0 = b\*rho;  
Point pt1(cvRound(x0 + 1000 \* (-b)),  
cvRound(y0 + 1000 \* (a)));  
Point pt2(cvRound(x0 - 1000 \* (-b)),  
cvRound(y0 - 1000 \* (a)));  
line(linesimg, pt1, pt2, Scalar(255, 255, 255), 2, 6);  
  
}  
  
imshow("Detected Lines", linesimg);  
  
int key = waitKey(33);  
if (key == 27)  
break;  
}  
}

**SETMOUSE CALL BACK**

#include "stdafx.h"

#include "opencv2/core/core.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <iostream>

#include <string>

#include <queue>

#include <math.h>

using namespace std;

using namespace cv;

void CallBackFunc(int event, int x, int y, int flags, void\* userdata)

{

Mat\* rgb = (Mat\*)userdata;

if (event == EVENT\_LBUTTONDOWN)

{

cout << "Left button of the mouse is clicked - position (" << x << ", " << y << ")" << endl;

cout << "Red=" << (int)(\*rgb).at<Vec3b>(y, x)[2] << endl;;

cout << "Green=" << (int)(\*rgb).at<Vec3b>(y, x)[1] << endl;

cout << "Blue=" << (int)(\*rgb).at<Vec3b>(y, x)[0] << endl;

}

/\*else if (event == EVENT\_RBUTTONDOWN)

{

cout << "Right button of the mouse is clicked - position (" << x << ", " << y << ")" << endl;

}

else if (event == EVENT\_MBUTTONDOWN)

{

cout << "Middle button of the mouse is clicked - position (" << x << ", " << y << ")" << endl;

}

else if (event == EVENT\_MOUSEMOVE)

{

cout << "Mouse move over the window - position (" << x << ", " << y << ")" << endl;

}\*/

}

int main()

{

Mat img = imread("F:\\picture000.jpg");

namedWindow("My Window", CV\_WINDOW\_AUTOSIZE);

setMouseCallback("My Window", CallBackFunc, &img);

imshow("My Window", img);

waitKey(0);

return 0;

}