```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

Mat frameC,frameT1,frameT2;
int th = 10, rMin = 0, rMax = 255, gMin = 0, gMax = 255, bMin = 0, bMax = 255;
void detEdge(Mat frame,int width,int height,int threshold,int dir=1)
{
	int rT, gT, bT;
	rT = gT = bT = 0;
	int rD, gD, bD;
	frameC = frame.clone();
	for (int i = 1; i < (dir==1?height:width); i++)
		for (int j = 1; j < (dir==1?width:height); j++)
		{
			Vec3b pixel=frame.at<Vec3b>(Point((dir==1?j:i),(dir==1?i:j)));
			int r, g, b;

			r = pixel.val[2];//Reads the values of Red Green And Blue Colours,
Structure is reversed
			g= pixel.val[1];
			b= pixel.val[0];

			rD = r - rT;//Calculates the difference between the last pixel value
for color reference
			gD = g- gT;
			bD = b- bT;

			rD *= rD;//Squares the values of difference getting rid of -ve values
			gD *= gD;
			bD *= bD;
			rD = sqrt(rD);
			gD = sqrt(gD);
			bD = sqrt(bD);
			if ((rD>threshold) || (gD > threshold) || (bD > threshold))
			{
				pixel.val[2] = pixel.val[0] = pixel.val[1] = 200;
			}
			else
			{
				pixel.val[0] = pixel.val[1] = pixel.val[2] = 0;
			}
			frame.at<Vec3b>(Point((dir==1?j:i), (dir==1?i:j))) = pixel;

			rT = r;//Stores the values of RGB Channels
			gT = g;
			bT = b;
		}
}

void reduceNoise(Mat frame, int width, int height)
{
	int count = 0,threshold=0;
	Mat frameTmp = frame;
	for (int i = 0; i < height; i++)
		for (int j = 0; j < width; j++)
		{
			Vec3b pixel = frame.at<Vec3b>(Point(j, i));
			if (count <= threshold && pixel.val[2] == 0)
			{
```

```
                    for (int k = 1; k <= count; k++)
                    {
                        pixel = frame.at<Vec3b>(Point(j - k, i));
                        pixel.val[0] = pixel.val[1] = pixel.val[2] = 0;
                        frame.at<Vec3b>(Point(j - k, i)) = pixel;
                    }
                }
                if (pixel.val[0] != 0)
                    count++;
                else
                    count = 0;
            }
}

void blur(Mat frame,int width,int height,int r)
{
    for (int i = 0; i < height; i ++)
        for (int j = 0; j+1 < width; j ++)
        {
            int x = j - r;
            int y = i - r;
            int rS, gS, bS;
            rS = gS = bS = 0;
            for (int k = i - r; k <= i + r; k+=1)
                for (int l = j - r; l <= j + r; l+=1)
                {
                    if (l > 0 && l < width&&k>0 && k < height)
                    {
                        Vec3b pixel = frame.at<Vec3b>(Point(l, k));
                        rS += pixel.val[2];
                        gS += pixel.val[1];
                        bS += pixel.val[0];
                    }
                }

            Vec3b pixC;
            pixC.val[0] = bS / ((2 * r + 1)*(2 * r + 1));
            pixC.val[1] = gS / ((2 * r + 1)*(2 * r + 1));
            pixC.val[2] = rS / ((2 * r + 1)*(2 * r + 1));
            frameC.at<Vec3b>(Point(j,i))=pixC;
        }
}

void thinEdge(Mat frame, int width, int height,int dir=1)
{
    Vec3b pixReset;
    pixReset.val[0]=pixReset.val[1]=pixReset.val[2] = 0;
    bool pixDet = false;
    int start;
    for (int i = 1; i < (dir==1?height:width); i++)
    {
        for (int j = 1; j < (dir==1?width:height); j++)
        {
            Vec3b pix = frame.at<Vec3b>(Point((dir==1?j:i), (dir==1?i:j)));
            frame.at<Vec3b>(Point((dir == 1 ? j : i), (dir == 1 ? i : j))) =
pixReset;
            if (!pixDet&&pix.val[0] != 0)
            {
                pixDet = true;
                start = j;
            }
            if (pixDet&&pix.val[0] == 0)
```

```
                {
                        pixDet = false;
                        for (int k = start; k <= j; k++)
                        {
                                pix.val[0] = pix.val[2] = 255; pix.val[1] = 255;
                                if (dir==1)
                                        frame.at<Vec3b>(Point(start + (j - start) / 2,i)) =
pix;
                                else if (dir==2)
                                        frame.at<Vec3b>(Point(i, start + (j - start) / 2)) =
pix;
                        }
                }
        }
        pixDet = false;
    }
}

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

    if (event == EVENT_LBUTTONDOWN)
    {
        system("cls");
        th++;
        cout << "Threshold:  "<<th;
    }
    else if (event == EVENT_RBUTTONDOWN)
    {
        system("cls");
        if (th>0)
        th--;
        cout << "Threshold:  " << th;
    }
    else if (event == EVENT_MBUTTONDOWN)
    {
    }
    else if (event == EVENT_MOUSEMOVE)
    {
    }
}

void detColor(Mat frame, int width, int height)
{
    for (int i = 1; i < height; i++)
        for (int j = 0; j < width; j++)
        {
            Vec3b p = frame.at<Vec3b>(Point(j, i));
            if (p.val[0] >=
rMin&&p.val[0]<rMax&&p.val[1]>=gMin&&p.val[1]<gMax&&p.val[2]>=bMin&&p.val[2] < bMax);
            else
            {
                p.val[0] = p.val[1] = p.val[2] = 255;
            }
            frame.at<Vec3b>(Point(j, i)) = p;

        }
}

void edgeStuff(Mat frame, int width, int height, int threshold)
{
    frameT1 = frame.clone();
```

```
	frameT2 = frame.clone();
	detEdge(frameT1, width, height, th, 1);		//Horizontal Scan
	thinEdge(frameT1, width, height, 1);
	detEdge(frameT2, width, height, th, 2);  //Vertical Scan
	thinEdge(frameT2, width, height, 2);

	Vec3b pixW, pixB;
	pixW.val[0] = pixW.val[1] = pixW.val[2] = 200;
	pixB.val[0] = pixB.val[1] = pixB.val[2] = 0;

	for (int i = 0; i < width; i++)		//Performs union of both of them
		for (int j = 0; j < height;j++)
			if (frameT1.at<Vec3b>(Point(i, j)).val[0] != 0 ||
frameT2.at<Vec3b>(Point(i, j)).val[0] != 0)
				frame.at<Vec3b>(Point(i, j))=pixW;
			else
				frame.at<Vec3b>(Point(i, j))=pixB;

}

int main(int argc, char* argv[])
{
	VideoCapture cap(0); // open the video camera no. 0
	double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width of frames of
the video
	double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the height of frames
of the video

	//cvNamedWindow("Original", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("Result", CV_WINDOW_AUTOSIZE);
	cvNamedWindow("Controls", CV_WINDOW_AUTOSIZE);

	createTrackbar("rMin", "Controls", &rMin, 255);
	createTrackbar("rMax", "Controls", &rMax, 255);
	createTrackbar("gMin", "Controls", &gMin, 255);
	createTrackbar("gMax", "Controls", &gMax, 255);
	createTrackbar("bMin", "Controls", &bMin, 255);
	createTrackbar("bMax", "Controls", &bMax, 255);

	while (1)
	{
		Mat frame;
		cap.read(frame); // feeds in the frame form webcam
		//imshow("Original", frame);
		detColor(frame, dWidth, dHeight);
		frameC = frame.clone();
		blur(frame, dWidth, dHeight, 1);
		frame = frameC.clone();

		edgeStuff(frame,dWidth,dHeight,th);
		imshow("Result", frame);

		setMouseCallback("Original", CallBackFunc, NULL);
		if (waitKey(1) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is
pressed, break loop
			break;

	}
	return 0;

}
```