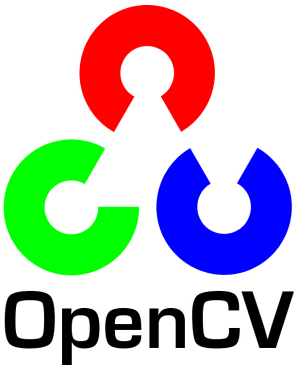
OpenCV Report 1
Robotics and Automation 282 762

Marc Alexander Sferrazza
12164165 *†

August 10, 2017



Abstract

A brief report on OpenCV edge image detection without using Canny function. Each step was tested and an out image saved to the local folder when running the program to ensure a best rendered results; these results can be tuned with the given functions in the program.

*This work was not supported by any organization

†Faculty of Mechatronics Engineering, Massey University, Albany, Auckland, New Zealand `Progress of project: https://github.com/alex1v1a/Robotics-and-Automation/`

Contents

1 INTRODUCTION

OpenCV (Open Source Computer Vision Library) is an open source machine learning library for visual components. It has been widely accepted as one of the worlds most used image processing tools, and has the capability of static and dynamic comprehension. Some of the basic features available in OpenCV are capable mapping environments to be used in demonstration such as self driving cars lane detection.

The process in this project to achieve is to design a program in the Visual Studio environment, using the OpenCV library that will render an images edges. OpenCV is a powerful tool for image processing with more then 2500 algorithms available in its library; by using several functions within the OpenCV library excluding Canny, a processed image will produce an output of the edges.

2 METHOD

An image has been provided of different colour tennis balls on a table. The process will involve deconstructing this image to the rendered edges only through several filters to provide the best matched edge.

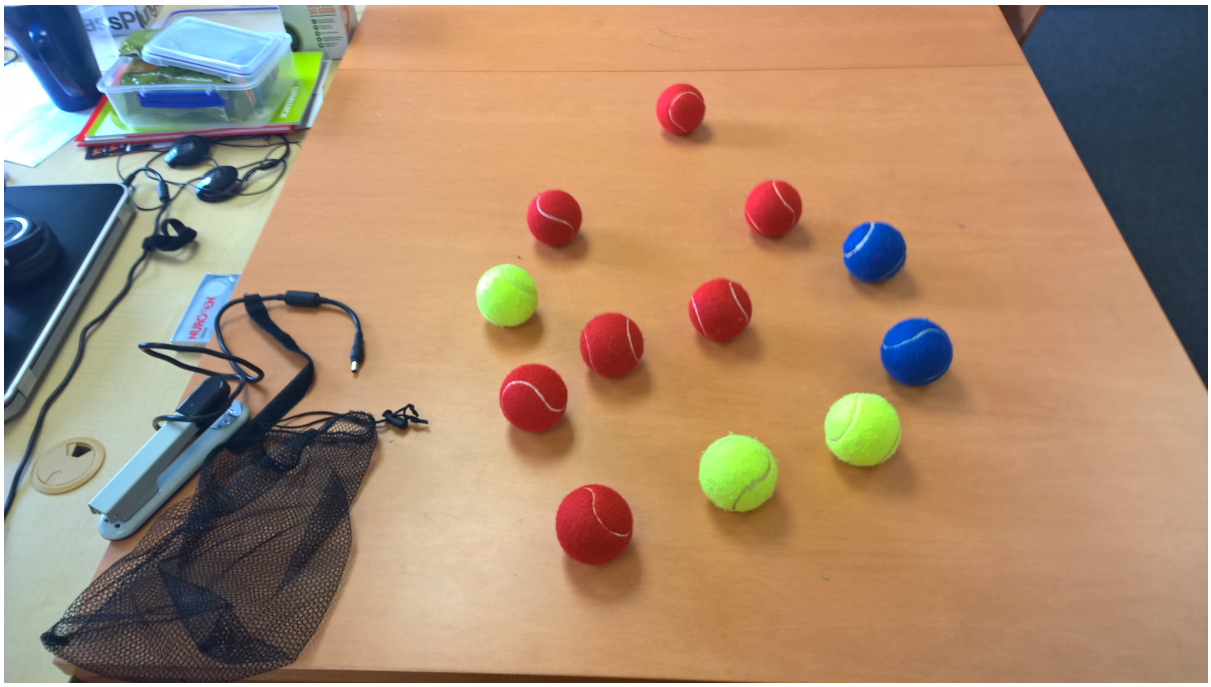


Figure 1: Original Image

A full break down of the process has been attached and can be found in the code in the appendix.

2.1 Greyscale

Using the greyscale function to convert the image to black and white will help reduce the processing errors.

```
//Convert to greyscale
cvtColor(image, greyscale, CV_BGR2GRAY);
imwrite("Grey Scale.jpg", greyscale); //Save greyscale
```

Figure 2: Greyscale filter

2.2 Gaussian Blur

The Gaussian blur and smoothing is used to reduce the noise of the image to help eliminate any external influences. The function will transform the image using convolution matrix, and will give a result based on each pixel and it's surrounding pixels to contour and blur while maintaining the edges integerity.

From the naked eye it may be hard to recognise this from the image below and the previous observed; but if the image is enlarged you can clearly see the edges and pixels blended slightly.

```
//Noise reduction
GaussianBlur(greyscale, blurred, Size(5, 5), 1.4);
imwrite("Blurred.jpg", greyscale);
```

Figure 3: Noise reduction with Gaussian Blur

2.3 Sobel operator

The Sobel operator combines the Gaussian blur and differentiation, and is used to find an approximation of the gradient; then process and detect the edges and produce an edge map. This is a two stage process in respect to calculating two derivatives, and may only be done correctly if the original image is converted to greyscale. The result of the first step will give two outputs, each convolution gradient in the x and y directions respectfully; in the second step it will combine the two gradient direction's from the outputs in step one and produce a final output.

```
//Grad X
Sobel(blurred, gradx, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);

//Grad Y
Sobel(blurred, grady, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);
```

2.4 Magnitude & Angle

```
//Magnitude, Angle
cartToPolar(gradx, grady, mag, angle, true);
```

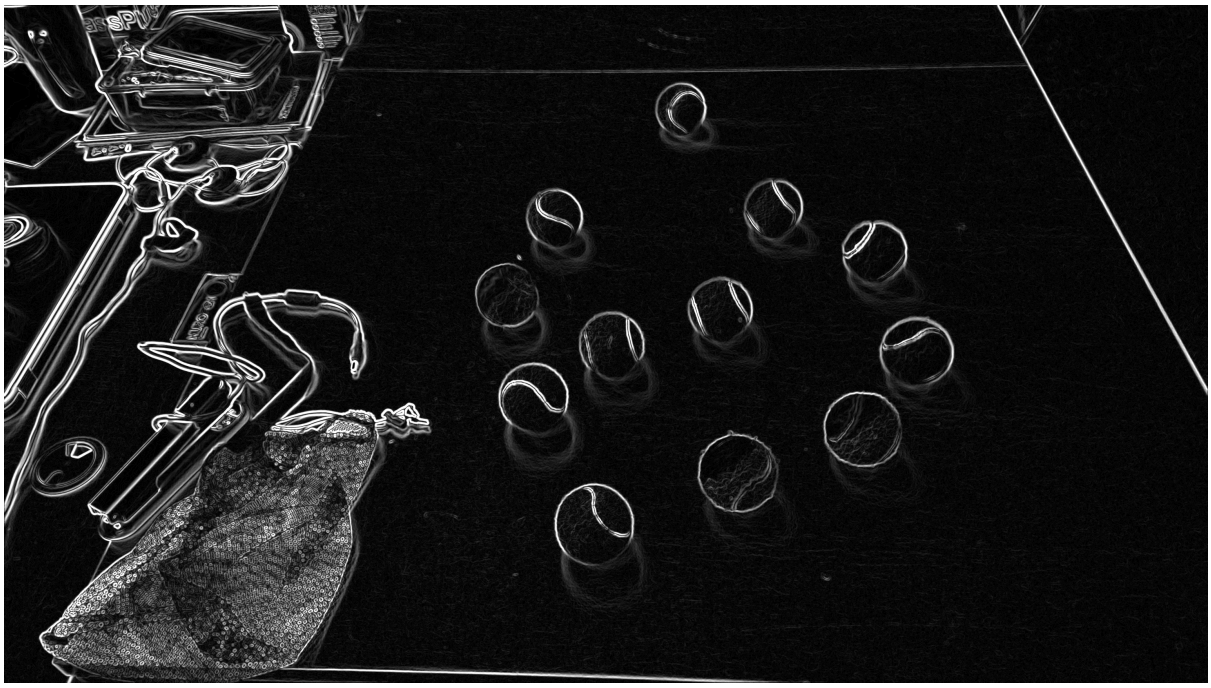


Figure 4: Magnitude transformations

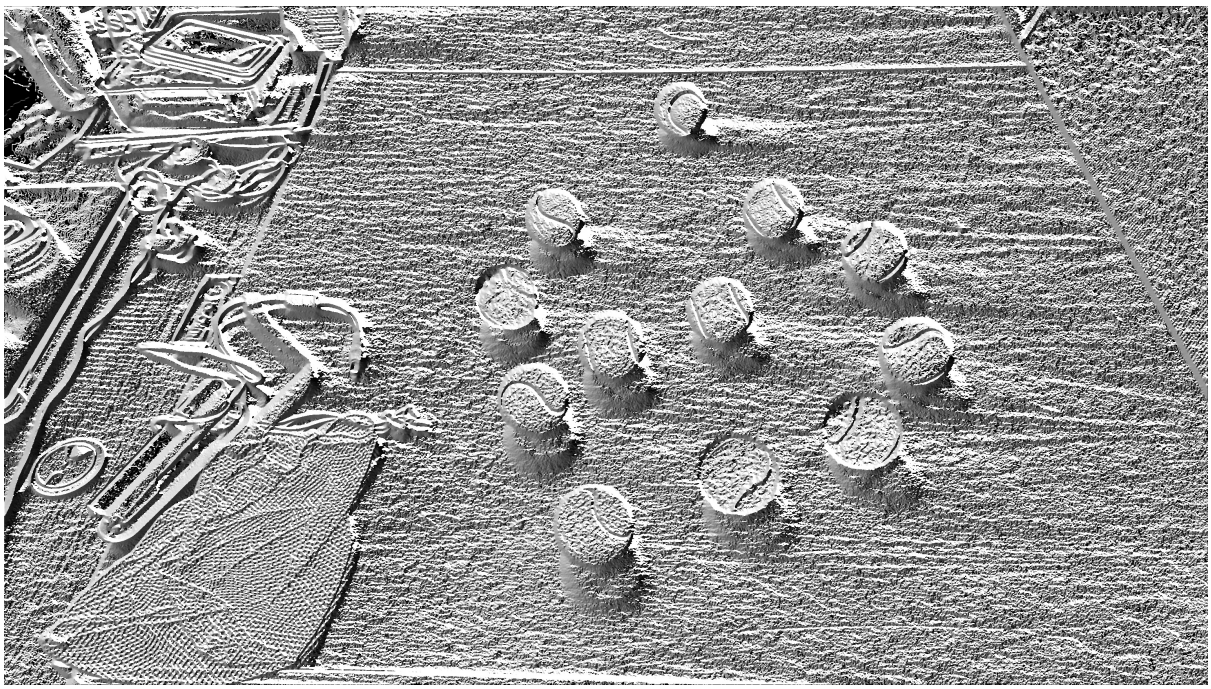


Figure 5: Angle transformations

2.5 Non Maximum Suppression

Going through all the pixels and and finding the surrounding pixels meeting the requirements at each point in respect to one another, then writing to that pixel if it what is found to be an edge white, or if found some requirements met write pixel grey, else non of the conditions were met and write the pixel black.

Below is a function called to find the Non Maximum Suppression render, thresholding also takes place simultaneously to this stage.

```
//Non Max Suppression
Mat maxsuppress(Mat mag, Mat angle){
  Mat suppress;
  float clr;
  suppress = Mat(mag.rows, mag.cols, CV_32F);
  for (int y = 1; y < mag.rows - 1; y++) {
    for (int x = 1; x < mag.cols - 1; x++) {
      float anglem = angle.at<float>(y, x);
      if (((anglem > 337.5) && (anglem <= 0)) || ((anglem <= 22.5) && (
          anglem > 0)) || ((anglem > 157.5) && (anglem <= 202.5))){
        //Thresholding takes place here, code removed for reference
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 22.5) && (anglem <= 67.5)) || ((anglem > 202.5) && (
          anglem <= 247.5))){
        //Thresholding takes place here, code removed for reference
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 67.5) && (anglem <= 112.5)) || ((anglem > 245.5) && (
          anglem <= 292.5))){
        //Thresholding takes place here, code removed for reference
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 112.5) && (anglem <= 157.5)) || ((anglem > 292.5) && (
          anglem <= 337.5))){
        //Thresholding takes place here, code removed for reference
        suppress.at<float>(y, x) = clr;
      }
    }
  }
  return suppress;
}
```

2.6 Thresholding

Along side the Non Maximum Suppression, once a value has been found the threshold will check to see if the value is greater or lower then the set point, and in turn write the pixel white or grey; if these conditions are not met then the pixel will be made black.

```
//Thresholding
if ((mag.at<float>(y, x - 1) < mag.at<float>(y, x)) && (mag.at<
    float>(y, x) > mag.at<float>(y, x + 1))){
  if (mag.at<float>(y, x) > 70) {
    clr = 255.0;
  }
  if ((mag.at<float>(y, x) < 70) && (mag.at<float>(y, x) < 1))
    clr = 100.0;
}
else {
  clr = 0.0;
}
```

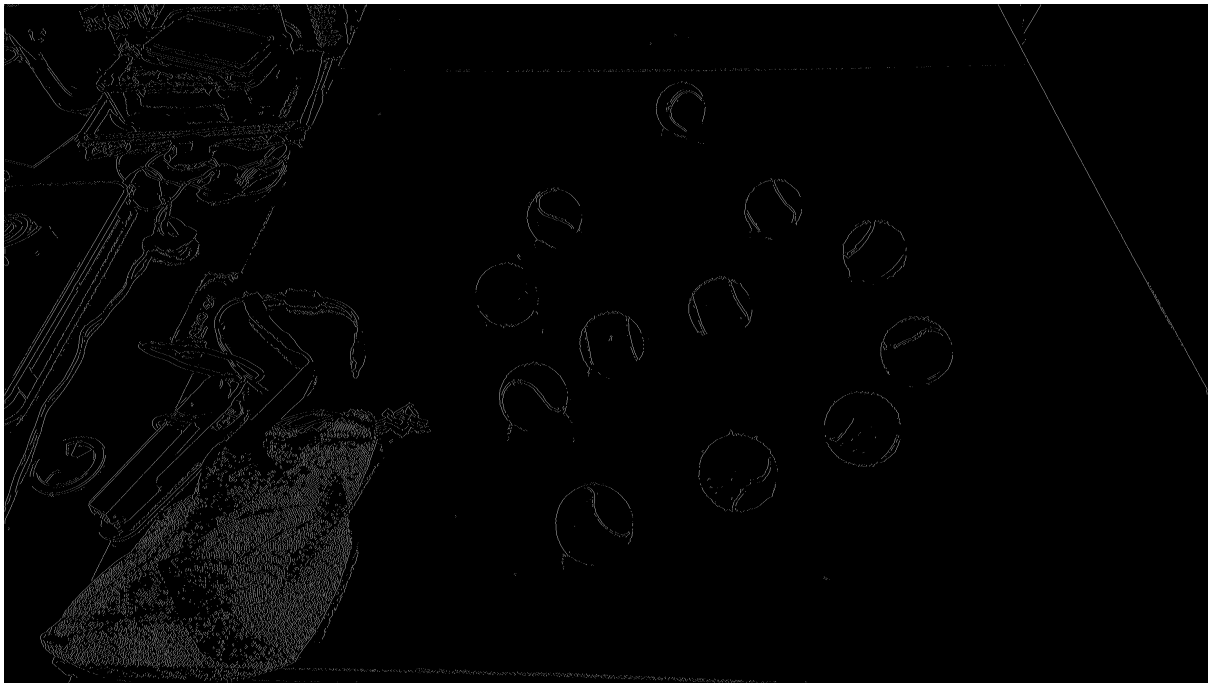


Figure 6: Non Maximum Suppression with Thresholding

2.7 Hysteresis

The Hysteresis recursive function is the called to edit the non-maximum suppression image via a pointer, and will enhance all the lines connecting the whites where possible.

```
void hysterysis(int y, int x, Mat *suppress) {
  if (suppress->at<float>(y, x + 1) == 100) {
    suppress->at<float>(y, x + 1) = 255.0;
    hysterysis(y, x + 1, suppress);
  }
  if (suppress->at<float>(y, x - 1) == 100) {
    suppress->at<float>(y, x - 1) = 255.0;
    hysterysis(y, x - 1, suppress);
  }
  if (suppress->at<float>(y + 1, x) == 100) {
    suppress->at<float>(y + 1, x) = 255.0;
    hysterysis(y + 1, x, suppress);
  }
  if (suppress->at<float>(y - 1, x) == 100) {
    suppress->at<float>(y - 1, x) = 255.0;
    hysterysis(y - 1, x, suppress);
  }
  if (suppress->at<float>(y - 1, x + 1) == 100) {
    suppress->at<float>(y - 1, x + 1) = 255.0;
    hysterysis(y - 1, x + 1, suppress);
  }
  if (suppress->at<float>(y - 1, x - 1) == 100) {
    suppress->at<float>(y - 1, x - 1) = 255.0;
    hysterysis(y - 1, x - 1, suppress);
  }
  if (suppress->at<float>(y + 1, x + 1) == 100) {
    suppress->at<float>(y + 1, x + 1) = 255.0;
    hysterysis(y + 1, x + 1, suppress);
  }
  if (suppress->at<float>(y + 1, x - 1) == 100) {
    suppress->at<float>(y + 1, x - 1) = 255.0;
    hysterysis(y + 1, x - 1, suppress);
  }
  else {
    suppress->at<float>(y,x) = 0.0;
  }
}
```

3 RESULTS

The final result from original image, to the final image of rendered edges only.

Figure 7: The final image is given of the edges only

3.1 Fine Tuning

While the process is straight forward and consists of step by step operations, the procedure in which required fine tuning. This involves checking the matrix at different points and the threshold values.

3.2 Testing

Each step has an output image saved to check the processes respectively. The images are checked to make sure each stage is completing the task correctly, if not the code is then referred to and further revisions are made.

3.3 Finalising

Making the program stable and concise, and as robust as possible builds for a good design. The method of testing all effects of the convolution matrix that might effect the system is required to make this achievable in getting the desired edge values.

4 CONCLUSIONS

The final image detection and edge render has been produced from several steps; this method while not exact is a basic idea of how the Canny operator works. The steps taken were to use the OpenCV functions to greyscale the image, then take that converted colour and run it through Gaussian blurring to reduce noise. After the transformation and noise reduction the Sobel operator is called to determine the gradients, and then cartToPolar function is used to convert these gradients to magnitudes and angles. The Non Suppression filter is then implemented with a threshold to produce more stable lines for the edges; and finally the Hysteresis stage is called to reduce the noise and threshold function to get rid of any grey static left over.

The result of the described method has been found similar to that of the canny detection method and the edges, while not perfectly shown are in the correct areas.

References

[1] Festo, "Automating with fst," *Federal Republic of Germany*, vol. 682 300, no. en 0402NH, 2004.

APPENDIX

```
#include <opencv2\opencv.hpp>
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>
#include <string>

using namespace cv;
using namespace std;

Mat maxsuppress(Mat mag, Mat angle);
void hysterysis(int y, int x, Mat *suppress);

int main(int, char** argv){

  Mat image, greyscale, blurred, maxsuppression, finalimage;

  //Read in the image
  image = imread("Image 1.jpg", IMREAD_COLOR);

  //Convert to greyscale
  cvtColor(image, greyscale, CV_BGR2GRAY);
  imwrite("Grey Scale.jpg", greyscale); //Save greyscale

  //Noise reduction
  GaussianBlur(greyscale, blurred, Size(5, 5), 1.4);
  imwrite("Blurred.jpg", greyscale);     //Save blurred

  //Gradient
  Mat gradx, grady, mag, angle;
  int scale = 2, delta = 0, ddepth = CV_32F;

  //Grad X
  Sobel(blurred, gradx, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);

  //Grad Y
  Sobel(blurred, grady, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);

  //Magnitude, Angle
  cartToPolar(gradx, grady, mag, angle, true);

  //Save images
  imwrite("Mag.jpg", mag);
  imwrite("Angle.jpg", angle);

  maxsuppression = maxsuppress(mag, angle);
  imwrite("MaxSuppression.jpg", maxsuppression);

  for (int y = 1; y < mag.rows - 1; y++) {
    for (int x = 1; x < mag.cols - 1; x++) {
      if (mag.at<float>(y, x) == 255.0) {
        hysterysis(mag.rows, mag.cols, &maxsuppression);
      }
    }
  }
```

```
  threshold(maxsuppression, maxsuppression, 245, 255, 0);

  imwrite("Final.jpg", maxsuppression);

  waitKey(27);
  destroyAllWindows();
  return 0;
}

//Non Max Suppression
Mat maxsuppress(Mat mag, Mat angle){
  Mat suppress;
  float clr;
  suppress = Mat(mag.rows, mag.cols, CV_32F);
  for (int y = 1; y < mag.rows - 1; y++) {
    for (int x = 1; x < mag.cols - 1; x++) {
      float anglem = angle.at<float>(y, x);
      if (((anglem > 337.5) && (anglem <= 0)) || ((anglem <= 22.5) && (
          anglem > 0)) || ((anglem > 157.5) && (anglem <= 202.5))){
        //Thresholding
        if ((mag.at<float>(y, x - 1) < mag.at<float>(y, x)) && (mag.at<
            float>(y, x) > mag.at<float>(y, x + 1))){
          if (mag.at<float>(y, x) > 70) {
            clr = 255.0;
          }
          if ((mag.at<float>(y, x) < 70) && (mag.at<float>(y, x) < 1))
            clr = 100.0;
        }
        else {
          clr = 0.0;
        }
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 22.5) && (anglem <= 67.5)) || ((anglem > 202.5) && (
          anglem <= 247.5))){
        //Thresholding
        if ((mag.at<float>(y, x - 1) < mag.at<float>(y, x)) && (mag.at<
            float>(y, x) > mag.at<float>(y, x + 1))){
          if (mag.at<float>(y, x) > 70) {
            clr = 255.0;
          }
          if ((mag.at<float>(y, x) < 70) && (mag.at<float>(y, x) < 1))
            clr = 100.0;
        }
        else {
          clr = 0.0;
        }
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 67.5) && (anglem <= 112.5)) || ((anglem > 245.5) && (
          anglem <= 292.5))){
        //Thresholding
        if ((mag.at<float>(y, x - 1) < mag.at<float>(y, x)) && (mag.at<
            float>(y, x) > mag.at<float>(y, x + 1))){
          if (mag.at<float>(y, x) > 70) {
            clr = 255.0;
          }
```

```
          if ((mag.at<float>(y, x) < 70) && (mag.at<float>(y, x) < 1))
            clr = 100.0;
        }
        else {
          clr = 0.0;
        }
        suppress.at<float>(y, x) = clr;
      }
      if (((anglem > 112.5) && (anglem <= 157.5)) || ((anglem > 292.5) && (
          anglem <= 337.5))){
        //Thresholding
        if ((mag.at<float>(y, x - 1) < mag.at<float>(y, x)) && (mag.at<
            float>(y, x) > mag.at<float>(y, x + 1))){
          if (mag.at<float>(y, x) > 70) {
            clr = 255.0;
          }
          if ((mag.at<float>(y, x) < 70) && (mag.at<float>(y, x) < 1))
            clr = 100.0;
        }
        else {
          clr = 0.0;
        }
        suppress.at<float>(y, x) = clr;
      }
    }
  }
  return suppress;
}

void hysterysis(int y, int x, Mat *suppress) {
  if (suppress->at<float>(y, x + 1) == 100) {
    suppress->at<float>(y, x + 1) = 255.0;
    hysterysis(y, x + 1, suppress);
  }
  if (suppress->at<float>(y, x - 1) == 100) {
    suppress->at<float>(y, x - 1) = 255.0;
    hysterysis(y, x - 1, suppress);
  }
  if (suppress->at<float>(y + 1, x) == 100) {
    suppress->at<float>(y + 1, x) = 255.0;
    hysterysis(y + 1, x, suppress);
  }
  if (suppress->at<float>(y - 1, x) == 100) {
    suppress->at<float>(y - 1, x) = 255.0;
    hysterysis(y - 1, x, suppress);
  }
  if (suppress->at<float>(y - 1, x + 1) == 100) {
    suppress->at<float>(y - 1, x + 1) = 255.0;
    hysterysis(y - 1, x + 1, suppress);
  }
  if (suppress->at<float>(y - 1, x - 1) == 100) {
    suppress->at<float>(y - 1, x - 1) = 255.0;
    hysterysis(y - 1, x - 1, suppress);
  }
  if (suppress->at<float>(y + 1, x + 1) == 100) {
    suppress->at<float>(y + 1, x + 1) = 255.0;
    hysterysis(y + 1, x + 1, suppress);
  }
```

```
    if (suppress->at<float>(y + 1, x - 1) == 100) {
      suppress->at<float>(y + 1, x - 1) = 255.0;
      hysterysis(y + 1, x - 1, suppress);
    }
    else {
      suppress->at<float>(y,x) = 0.0;
    }
}
```