Tarea 10

Angel Manrique Pozos Flores; N.C M07211505
Instituto Tecnológico Nacional de México,
Blvd. Industrial, Mesa de Otay, 22430 Tijuana, B.C., México.

10 de abril de 2016

En el presente trabajo se hace una investigación para saber en que consiste la supresión de no maximos, a su vez se hace la implementacion del detector de Canny y se sustituye el kernel Gaussiano y el kernel de Sobel por un kernel que obtenga la derivada de la Gaussiana.

1. Introducción

El procesamiento de imágenes es de gran utilidad en la mayoría de las áreas de investigación ya que todo lo que captamos en el mundo la gran mayoría de los datos provienen de nuestros sentidos en especial la vista.

Por ello el estudio de la visión computacional es de gran importancia, el presente trabajo se hace un acercamiento a esta área, utilizando las librerías de visión open source OpenCV y el software libre CodeBlocks el cual se configuro para que pudiera ser capaz de reconocer las librerías de OpenCV.

2. Supresión de no máximos

En el área de procesamiento de imágenes, la detección de los bordes de una imagen es de suma importancia y utilidad, pues facilita muchas tareas, entre ellas, el reconocimiento de objetos, la segmentación de regiones, entre otras.

La supresión de no máximos es una técnica para el adelgazamiento de bordes, después de aplicar el calculo del gradiente, la supresion de no maximos ayuda a suprimir todos los valores del gradiente igualandolos a 0 excepto los locales

maximos los cuales nos indican el cambio del valor maximo de intensidad de esa zona.

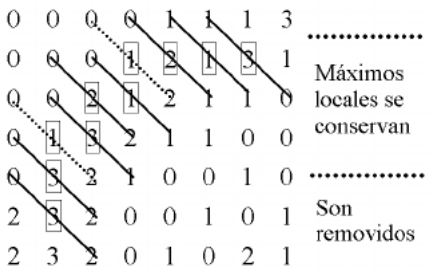


Figura 1: Supresion de no maximos.

existen 4 casos particulares:

- Si el angulo redondeado del gradiente es 0 grados (el borde esta en la dirección norte-sur), el punto debe considerarse que esta en un borde si y solo si la magnitud del gradiente es mayor que la magnitud de los pixeles en las direcciones este-oeste.

- Si el angulo redondeado del gradiente es 90 grados (el borde esta en la direccion este-oeste), el punto debe considerarse que esta en un borde si y solo si la magnitud del gradiente es mayor que la magnitud de los pixeles en las direcciones norte-sur.

- Si el angulo redondeado del gradiente es 135 grados (el borde esta en la direccion noreste-suroeste), el punto debe considerarse que esta en un borde si y solo si la magnitud del gradiente es mayor que la magnitud de los pixeles en las direcciones noroeste-sureste.

- Si el angulo redondeado del gradiente es 45 grados (el borde esta en la direccion noroeste - sureste), el punto debe considerarse que esta en un borde si y solo si la magnitud del gradiente es mayor que la magnitud de los pixeles en las direcciones noreste-suroeste.

Una forma de obsevarlo es utilizando la figura 2 donde si nos posicionamos en el pixel central (gris) solo existen 3 posibles bordes que van de norte-sur (vecinos verdes), este-oeste (vecinos azules) o los diagonales (vecinos amarillos y rojos).

Despues de la supresion de no maximos obtenemos algo que se le denomina adelgazamiento de bordes.

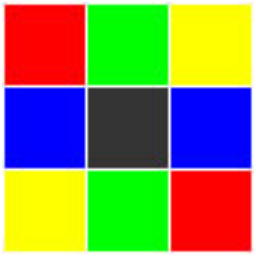


Figura 2: Posibles direcciones de un borde.

3. Detector de Canny

El algoritmo de Canny es usado para detectar todos los bordes existentes en una imagen. Este algoritmo esta considerado como uno de los mejores métodos de detección de contornos mediante el empleo de máscaras de convolución y basado en la primera derivada.

Este satisface 3 criterios principales:

- Proporcionar un error bajo, lo cual nos daría una buena detección ya que solo detectaría los bordes existentes.
- Buena localización, la distancia entre los pixeles del borde detectado y el borde real debe ser mínima.
- Respuesta mínima, solo un detector debe responder por borde.

Comúnmente se utiliza un filtro Gaussiano para la disminución del ruido un ejemplo de un kernel Gaussiano de tamaño 5 se muestra a continuacion en 1.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{1}$$

Después se aplica un filtro de Sobel para encontrar el valor de la intensidad del gradiente de la imagen, donde se aplica un par de mascaras de convolucion en la dirección de los ejes X y Y dados en la figura 3.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{2}$$

Posteriormente se calcula la magnitud y la dirección del gradiente como se observa en la figura 4

$$\begin{aligned} G &= \sqrt{G_x^2 + G_y^2} \\ \theta &= \arctan(\frac{G_y}{G_x}) \end{aligned} \tag{3}$$

La dirección es redondeada a uno de 4 posibles ángulos los cuales son 0, 45, 90, 135. Se aplica la supresión de no máximos la cual remueve los pixeles que no son considerados parte del borde, donde solamente las lineas delgadas se mantienen.

Se aplica la histeresis la cual consiste en aplicar niveles de threshold con valores máximos y valores mínimos definidos por el usuario donde:

- Si el valor del gradiente de un pixel esta por arriba del valor máximo de thresholding, el pixel se considera un borde.
- Si el valor del gradiente de un pixel esta por debajo del valor mínimo de thresholding, es rechazado.
- si el valor del gradiente de un pixel esta entre ambos valores de threshold, sera aceptado solo si tiene al menos un pixel que este conectado con el valor de thresholding máximo.

Se utilizo la imagen de la figura 3 para la realización de las pruebas.

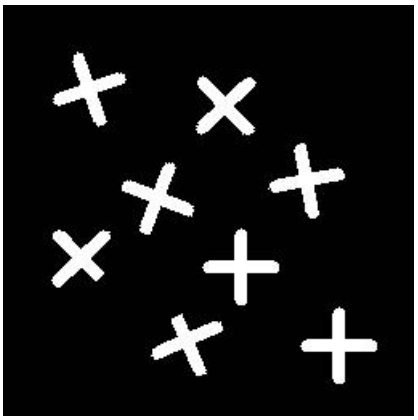


Figura 3: Imagen original para las pruebas de Canny y MyCanny.

Para ello se desarrollo el siguiente código en C++ utilizando las librerías de visión de OpenCV.

```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>


using namespace cv;
using namespace std;


Mat img, gray, dst, detected_edges;

int edgeThresh = 1;
int lowThreshold;
int const max_lowThreshold = 255;
int ratio = 3;
int kernel_size = 3;
char* window_name = "Edge Map";
char* source_window = "Original Image";

void CannyThreshold(int, void*)
{

  // Reduce noise with a kernel 3x3, Canny Edge
  blur( gray, detected_edges, Size(3,3) );
  Canny( detected_edges, detected_edges, lowThreshold,
      lowThreshold*ratio, kernel_size );

  // Using Canny's output as a mask, we display our
     result
  dst = Scalar::all(0);


  //Showing the result
  namedWindow(source_window, CV_WINDOW_AUTOSIZE );
  imshow( source_window, img );

  img.copyTo( dst, detected_edges);
  imshow( window_name, dst );
```

```
 }


int main( int argc, char** argv )
{

  img = imread("ar.jpg");

  if( !img.data )
  { return -1; }

  img.create( img.size(), img.type() );
  cvtColor( img, gray, CV_BGR2GRAY );
  namedWindow( window_name, CV_WINDOW_AUTOSIZE );

  createTrackbar( "Min Threshold:", window_name, &
     lowThreshold, max_lowThreshold, CannyThreshold );
  CannyThreshold(0, 0);

  waitKey(0);
  return 0;

  }
```

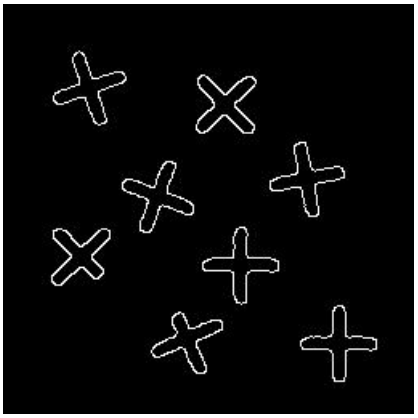


Figura 4: Imagen obtenida aplicando la funcion de Canny del algoritmo 1 con un valor de thresh de 50 en escala a 255.

Ahora vamos a modificar el algoritmo para hacer la sustitución del kernel Gaussiano y el kernel de Sobel (el cual viene predefinido dentro de la funcion

Canny desarrollada por OpenCV), y sustituirlo por un kernel que obtenga la derivada de la Gaussiana.

```
#include <opencv2/opencv.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace std;
using namespace cv;

// Global Variables
Mat img, gray, bw, dst;
Mat Kernelx, Kernely;
Mat grad_x, grad_y, abs_grad_x, abs_grad_y, grad;
Mat GaussobelX, GaussobelY;

void thinningIteration(cv::Mat& im, int iter);
void thinning(cv::Mat& im);
void Canny(int, void*);



int main(int argc, char** argv)
{

        img = imread(argv[1] = "ar.jpg");

        if (!img.data)
        {
        return -1;
        }

        imshow("Imagen Original", img);
        cvtColor(img, gray, CV_BGR2GRAY);

        // Gaussian kernel with ksize = 3 and sigma =
           1.0
        Mat kernelX = getGaussianKernel(3, 1.0, CV_32F
           );
```

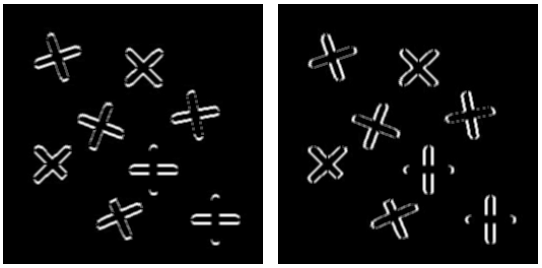
```
        Mat kernelY = getGaussianKernel(3, 1.0, CV_32F
            );
        Mat kernelXY = kernelX * kernelY.t();
        cout << "Gaussian = " << endl << " " <<
            kernelXY << endl << endl;


        Mat sobelX, sobelY;

        // x direction
        getDerivKernels(sobelX, sobelY, 1, 0, 3, false
            , CV_32F);
        Mat sobelXX = sobelX * sobelY.t();
        cout << "sobel X = " << endl << " " << sobelXX
             << endl << endl;

        // y direction
        getDerivKernels(sobelX, sobelY, 0, 1, 3, false
            , CV_32F);
        Mat sobelYY = sobelX * sobelY.t();
        cout << "sobel Y = " << endl << " " << sobelYY
             << endl << endl;


        GaussobelX = sobelXX * kernelX;
        cout << "Gauss Deriv X = " << endl << " " <<
            GaussobelX << endl << endl;

        GaussobelY = GaussobelX.t();
        cout << "Gauss Deriv Y = " << endl << " " <<
            GaussobelY << endl << endl;


        Canny(0, 0);

        waitKey(0);
        return 0;
}

void Canny(int, void*)
{
```

```
        blur(gray, gray, Size(3, 3));
        filter2D(gray, grad_x, CV_32F, GaussobelX,
           Point(-1, -1));
        filter2D(gray, grad_y, CV_32F, GaussobelY,
           Point(-1, -1));
        convertScaleAbs(grad_x, abs_grad_x);
        convertScaleAbs(grad_y, abs_grad_y);

        addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5,
           0, grad);
        imshow("Gradiente", grad);
        imshow("Gradiente X", abs_grad_x);
        imshow("Gradiente Y", abs_grad_y);


        threshold(grad, bw, 0, 255, THRESH_OTSU);
        imshow("Tresholded", bw);

        thinning(bw);
        imshow("Non Maximum Suppresion ", bw);
}

//NMS
void thinning(cv::Mat& im)
{
        im /= 255;
        cv::Mat prev = cv::Mat::zeros(im.size(),
           CV_8UC1);
        cv::Mat diff;
        do {
                thinningIteration(im, 0);
                thinningIteration(im, 1);
                cv::absdiff(im, prev, diff);
                im.copyTo(prev);
        } while (cv::countNonZero(diff) > 0);
        im *= 255;
}

// Thining Iterations Fuction
void thinningIteration(cv::Mat& im, int iter)
```

```
{
        cv::Mat marker = cv::Mat::zeros(im.size(),
            CV_8UC1);

        for (int i = 1; i < im.rows - 1; i++)
        {
                for (int j = 1; j < im.cols - 1; j++)
                {
                        uchar p2 = im.at<uchar>(i - 1,
                            j);
                        uchar p3 = im.at<uchar>(i - 1,
                            j + 1);
                        uchar p4 = im.at<uchar>(i, j +
                            1);
                        uchar p5 = im.at<uchar>(i + 1,
                            j + 1);
                        uchar p6 = im.at<uchar>(i + 1,
                            j);
                        uchar p7 = im.at<uchar>(i + 1,
                            j - 1);
                        uchar p8 = im.at<uchar>(i, j -
                            1);
                        uchar p9 = im.at<uchar>(i - 1,
                            j - 1);

                        int A = (p2 == 0 && p3 == 1) +
                            (p3 == 0 && p4 == 1) +
                                (p4 == 0 && p5 == 1) +
                                    (p5 == 0 && p6 ==
                                       1) +
                                (p6 == 0 && p7 == 1) +
                                    (p7 == 0 && p8 ==
                                       1) +
                                (p8 == 0 && p9 == 1) +
                                    (p9 == 0 && p2 ==
                                       1);
                        int B = p2 + p3 + p4 + p5 + p6
                            + p7 + p8 + p9;
                        int m1 = iter == 0 ? (p2 * p4
                           * p6) : (p2 * p4 * p8);
                        int m2 = iter == 0 ? (p4 * p6
```

```
                                        * p8) : (p2 * p6 * p8);

                                    if (A == 1 && (B >= 2 && B <=
                                        6) && m1 == 0 && m2 == 0)
                                            marker.at<uchar>(i, j)
                                                = 1;
                            }
                    }

                    im &= ~marker;
            }
```

Donde primero obtenemos los kernels Gaussianos para el eje X y Y que definimos de tamaño 3x3 con un valor de σ=1 y calculamos los valores de sus gradientes para los respectivos ejes como se muestra en la Figura 5.



(a) Gradiente de X. (b) Gradiente de Y.

Figura 5: Aplicacion de nuestro kernel sobre los ejes X y Y para la obtencion del gradiente.

Despues se hace el calculo del gradiente total de la cual obtenemos

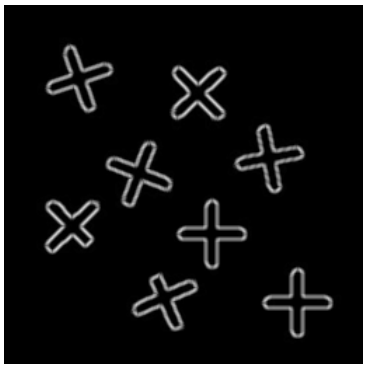
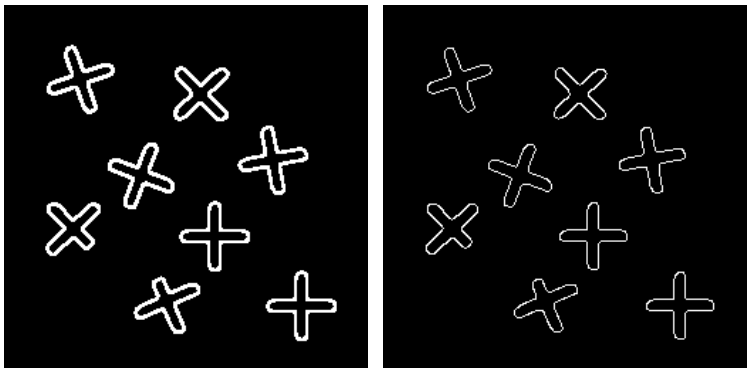


Figura 6: Gradiente total obtenido.

Se le hace un thresholding y se le aplica la supresión de no máximos para que sean visibles los valores con mayor peso.



(a) Aplicación de un thresholding a nuestra imagen del gradiente total.

(b) Aplicacion de la supresion de no maximos para la deteccion de bordes.

Figura 7: Imagenes obtenidas del desarrollo del algoritmo 2, donde a) es la imagen con un thresholding aplicado y b) es la imagen final que se obtiene de la aplicacion de supresion de no maximos.

4. Conclusiones

Como se observa utilizando la función desarrollada en OpenCV, donde intercambiamos los operadores de sobel y la gaussiana, por el calculo de la derivada de la gaussiana, obtenemos una respuesta muy buena que puede compararse fácilmente con la desarrollada en la función Canny de OpenCV, la dificultad de este ejercicio radicaba en precisamente hacer el intercambio y definir nosotros nuestro operador gaussiano y posteriormente calcular su derivada pero una vez logrando esto es solo cuestión de aplicar lo visto en ejercicios anteriores para poder llevarlo a cabo, la supresión de no máximos es un poco mas difícil de entender y llevar a cabo por ello se agrego un poco de teoría sobre el como poder visualizarla para su posterior implementan.

5. Bibliografía

- Canny Edge Detector.
  ($http://tinyurl.com/pz5rrgj$)
- The Canny Edge Detector.
  ($http://tinyurl.com/zsugeqp$)
- Canny edge detector.
  ($http://tinyurl.com/qj83woy$)
- Non-max suppression.
  ($http://tinyurl.com/jmxk8cc$)
- Thinning.
  $http://tinyurl.com/zn9ume9$