Tarea 13

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

11 de abril de 2016

En el presente trabajo se hace una investigacion y desarrollo del operador para la deteccion de esquinas propuesto por Harris y Stephens .

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. Función de autocorrelación

La funcion de autocorrelacion mide cambios locales en una señal, esta medida se obtiene por la correlacion de un elemento de la imagen con sus vecinos.

Es decir, se obtienen las diferencias de un elemento con respecto a sus vecinos por medio de corrimientos en diferentes direcciones alrededor de este, en el caso de un punto de interes y en particular de una esquina, la funcion de autocorrelacion es alta para todos los cambios de direccion donde la funcion de autocorrelacion en una imagen se define como:

$$f(x,y) = \sum_{(x_k,y_k)\in W} [I(x_k,y_k) - I(x_k+\Delta_x, y_k+\Delta_y)]^2 \tag{1}$$

Donde (x_k, y_k) pertenecen a una region W centrada en el punto (x, y) de la imagen I.

Para utilizar esta funcion como un detector de puntos de interes o como un detector de esquinas, se requiere realizar una integracion sobre todas las direcciones donde se llevaron a cabro los corrimientos, esta integracion sobre corrimientos discretos puede ser sustituida por la matriz de autocorrelacion.

La matriz de autocorrelacion se deriva de una aproximacion de primer orden de la expansion de Taylor.

$$I(x_k + \Delta_x, y_k + \Delta_y) \approx I(x_k, y_k) + (I_x(x_k, y_k) I_y(x_k, y_k)) \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \tag{2}$$

Donde $I_x(x_k, y_k)$ y $I_y(x_k, y_k)$ indican la primera derivada de la imagen I repecto a x y respecto a y, sustituyendo 2 en 1 obtenemos la siguiente expresion:

$$f(x, y) = \sum_{(x_k, y_k) \in W} \left[(I_x(x_k, y_k) I_y(x_k, y_k)) \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \right]^2 \tag{3}$$

Desarrollandola obtenemos que:

$$(\Delta_x \Delta_y) \cdot \begin{bmatrix} \sum_{(x_k, y_k) \in W} (I_x(x_k, y_k))^2 & \sum_{(x_k, y_k) \in W} (I_x(x_k, y_k) I_y(x_k, y_k)) \\ \sum_{(x_k, y_k) \in W} (I_x(x_k, y_k) I_y(x_k, y_k)) & \sum_{(x_k, y_k) \in W} (I_y(x_k, y_k))^2 \end{bmatrix} \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \tag{4}$$

Quedando:

$$(\Delta_x \Delta_y) \cdot M(x, y) \cdot \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} \tag{5}$$

De esta forma podemos aproximar la funcion de autocorrelacion por medio de la matriz $M(x, y)$ donde se le da un peso a esta por medio de un filtro gaussiano $G(\sigma)$, esto es:

$$M(x, y) = G(\sigma) \otimes \begin{bmatrix} (I_x(x_k, y_k))^2 & I_x(x_k, y_k) I_y(x_k, y_k) \\ I_x(x_k, y_k) I_y(x_k, y_k) & (I_y(x_k, y_k))^2 \end{bmatrix} \tag{6}$$

3. Detector de Harris & Stephens

En 1988 Harris & Stephens proponen un detector de esquinas basado en el detector de puntos de interes propuesto por Moravec y la funcion de autocorrelacion, donde Harris & Stephens definen la funcion de autocorrelacion $E_{x,y}$ de la siguiente forma:

$$E_{x,y} = \sum_{u,v} W_{u,v} \cdot [I_{x+u,y+u} - I_{u,v}]^2 \approx \sum_{u,v} W_{u,v} \cdot [xX + yY + O(x^2, y^2)]^2 \tag{7}$$

Donde $W_{u,v}$ es una región de la imagen I.
Y los gradientes son aproximados como:

$$X = I \otimes [-1,0,1] = \frac{\delta I}{\delta_x} \tag{8}$$

$$X = I \otimes [-1,0,1]^T = \frac{\delta I}{\delta_y} \tag{9}$$

Donde:

$$\begin{array}{rl} A = & X^2 \otimes w_{u,v} \\ B = & Y^2 \otimes w_{u,v} \,. \\ C = & XY \otimes w_{u,v} \end{array} \tag{10}$$

Tomando la gaussiana como:

$$w_{u,v} = \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right) \tag{11}$$

Finalmente los cambios en $E_{x,y}$ para pequeños corrimientos en (x,y), se puede describir como:

$$E_{x,y} = (x,y) M (x,y)^T \tag{12}$$

Donde M es una matriz simétrica de 2x2 descrita como:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{13}$$

Siendo M la matriz de autocorrelacion descrita en las ecuaciones 3 y 4, donde las curvaturas principales de la funcion de autocorrelacion estan definidas por los eigenvalores α y β de la matriz de autocorrelacion M y forman una descripcion rotacionalmente ivariante de M.

Los eigenvalores α y β de la matriz de autocorrelacion M se pueden aproximar como:

$$\begin{array}{c} T_r(M) = \alpha + \beta = A + B \\ D_{et}(M) = \alpha\beta = AB - C^2 \end{array} \tag{14}$$

Donde $T_r(M)$ indica la traza de M y $D_{et}(M)$ indica el determinante de M, de tal forma que el detector de Harris puede modelarse como:

$$Harris = D_{et}(M) - k \cdot T_r(M) \tag{15}$$

Donde k es llamada constante de Harris la cual es propuesta por el usuario pero comunmente en la practica se le asigna el valor de aproximado de 0,04.

Basandonos en las ecuaciones vistas en el presente trabajo y utilizando la imagen de la figura 1 para la realización de pruebas, podemos ser capaces de desarrollar un algoritmo en C++ utilizando las librerías de visión de OpenCV donde la funcion MyHarris involucra los pasos descritos mediante las ecuaciones.

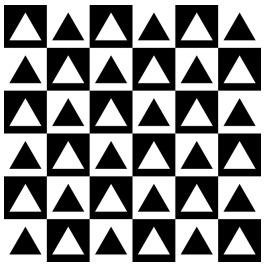

Figura 1: Imagen de prueba para el algoritmo de Harris.

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

//MyHarris
using namespace std;
using namespace cv;

// Variables Globales
Mat img, gray, bw, dst;
Mat Kernelx, Kernely;
Mat grad_x, grad_y, grad_x2, grad_y2;
Mat abs_grad_x, abs_grad_y, grad;

int thresh = 200;
int max_thresh = 255;
```

```
char* source_window = "Source image";
char* corners_window = "Corners detected";


// Function header
void MyHarris( int, void* );
void thinningIteration(cv::Mat& im, int iter);
void thinning(cv::Mat& im);



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

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

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

    cvtColor(img, gray, CV_BGR2GRAY );

    // Create a window and a trackbar
    namedWindow( source_window, CV_WINDOW_AUTOSIZE );
    createTrackbar( "Threshold: ", source_window, &
       thresh, max_thresh, MyHarris );
    imshow( source_window, img );

    MyHarris( 0, 0 );

    waitKey(0);
    return 0;
}


void MyHarris(int, void*)
{
    double k = 0.04;
```

```
Mat response_norm, response_norm_scaled;
Mat response = Mat::zeros( img.size(), CV_32FC1 );


// Kernels for x and y
Kernelx = (Mat_<double>(1, 3) <<
    -1, 0, 1);

Kernely = (Mat_<double>(3, 1) <<
    -1, 0, 1);

cout << "Kernel eje Y: " << endl << Kernely <<
   endl << endl;
cout << "Kernel eje X: " << endl << Kernelx <<
   endl << endl;


// Convolution of the Image and kernels (Gradient
   )
filter2D(gray, grad_x, CV_16S, Kernelx, Point(-1,
   -1));
filter2D(gray, grad_y, CV_16S, Kernely, 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("Gradient x", abs_grad_x);
imshow("Gradient y", abs_grad_y);
imshow("Gradient xy", grad);

/// Operaciones para el calculo de la matriz de
   autocorrelacion
Mat multix  = abs_grad_x.mul(abs_grad_x); // X*X
Mat multiy  = abs_grad_y.mul(abs_grad_y); // Y*Y
Mat multixy = abs_grad_x.mul(abs_grad_y); // X*Y
```

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

//Gaussian kernel(3x3)
Mat kernelXY = kernelY * kernelXt;

cout << "Gaussian Y = " << endl << " " << kernelY
   << endl << endl;
cout << "Gaussian X = " << endl << " " << kernelXt
    << endl << endl;
cout << "Gaussian XY = " << endl << " " <<
   kernelXY << endl << endl;


Mat convx, convy, convxy, abs_conv_x, abs_conv_y,
   abs_conv_xy;

// Convolution of the Image and kernels (Gradient
   )
filter2D(multix, convx, CV_16S, kernelXt, Point
   (-1, -1));
filter2D(multiy, convy, CV_16S, kernelY, Point(-1,
    -1));
filter2D(multixy, convxy, CV_16S, kernelXY, Point
   (-1, -1));

convertScaleAbs(convx, abs_conv_x);
convertScaleAbs(convy, abs_conv_y);
convertScaleAbs(convxy, abs_conv_xy);

imshow("Gauss convolution with x*x", abs_conv_x);
    //A = XX conv GaussX
imshow("Gauss convolution with y*y", abs_conv_y);
    //B = YY conv GaussY
imshow("Gauss convolution with x*y", abs_conv_xy);
    //C = XY conv GaussXY

// Aproximaciones
```

```
    Mat trz = (abs_conv_x) + (abs_conv_y); // A + B
    Mat det = (abs_conv_x.mul(abs_conv_y)) - (
       abs_conv_xy.mul(abs_conv_xy)); // A*B - C*C

    imshow("Traza", trz);
    imshow("Determinante", det);

    // Normalizing
    response = det - (k * trz);
    normalize( response, response_norm, 0, 255,
      NORM_MINMAX, CV_32FC1, Mat() );
    convertScaleAbs( response_norm,
       response_norm_scaled );


    // Drawing a circle around corners
    for( int j = 0; j < response_norm.rows ; j++ )
     { for( int i = 0; i < response_norm.cols; i++ )
          {
            if( (int) response_norm.at<float>(j,i) >
               thresh )
              {
               circle( response_norm_scaled, Point( i,
                   j ), 5,  Scalar(255, 0, 0), 1, 2, 0
                   );
              }
          }
     }

    // Showing the result
    thinning(response_norm_scaled);
    namedWindow( corners_window, CV_WINDOW_AUTOSIZE );
    imshow( corners_window, response_norm_scaled );

}


//Funcion para La supresion de No-Maximos
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;
}

// Funcion para realizar las iteraciones de
   adelgazamiento
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;
}
```

Obteniendo las imagenes finales mostradas a continuación.

Para el calculo de los gradientes.

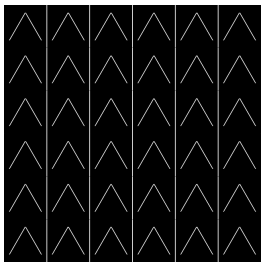

Figura 2: Gradiente X.

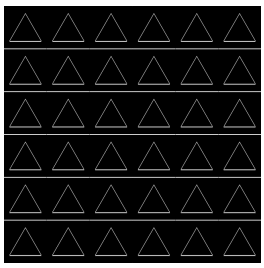

Figura 3: Gradiente Y.

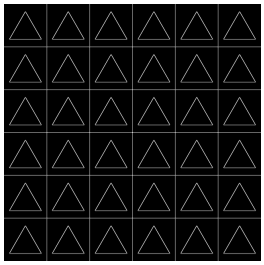

Figura 4: Gradiente XY.

Y para las operaciones de convolucion.

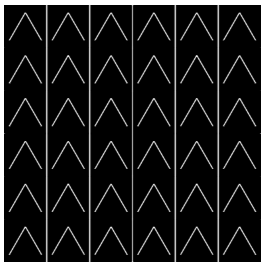

Figura 5: Convolucion de X^2 con la Gaussiana.

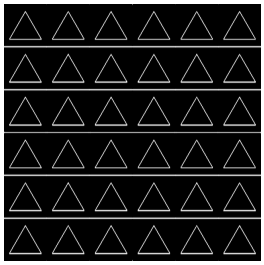

Figura 6: Convolucion de Y^2 con la Gaussiana.

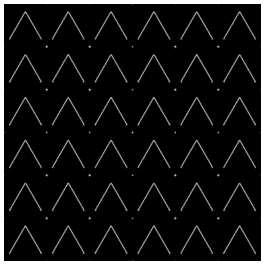

Figura 7: Convolucion de XY con la Gaussiana.

Para el determinante y la traza.

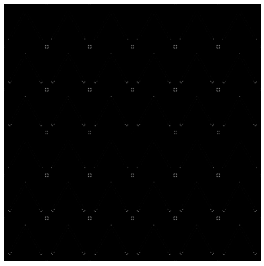

Figura 8: Determinante obtenido.

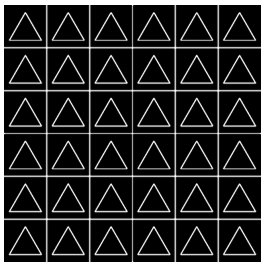

Figura 9: Traza obtenida.

Y finalmente utilizando la ecuación 15 y aplicando una supresión de no máximos obtenemos.

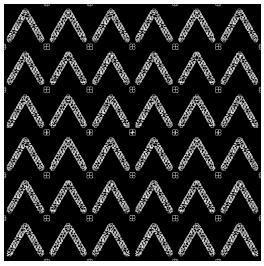

Figura 10: Imagen con un valor de threshold de 0 donde se aprecian todos los puntos de interes de la imagen localizados por MyHarris.

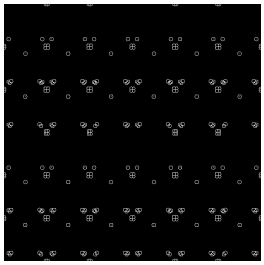

Figura 11: Imagen con un valor de threshold de 54, donde se aprecian los puntos de interes encontrados por MyHarris.

4. Conclusiones

El detector de Harris es uno de los mas detectores mas utilizados dentro de la literatura, donde la idea general es localizar los puntos de interes donde existan esquinas en las cuales mediante el calculo del gradiente y las operaciones de convolucion asociadas, somos capaces de poder determinar estas zonas de interés en una imagen, a su vez existe tambien el detector de Shi-Tomasi que es una version mejorada de este algoritmo.

5. Bibliografía

- Harris Corner Detector in Python.
  ($http://tinyurl.com/grvhff4$)
- Harris Corner Detector.
  ($http://tinyurl.com/zl93v47$)
- openCV, Harris Corner Detector.
  ($http://tinyurl.com/zy8dz8r$)
- Harris Corner Detector.
  ($http://tinyurl.com/gv42eaw$)
- Creating yor own corner detector.
  $http://tinyurl.com/zlpny4y$
- Corners y matching de regiones.
  $http://tinyurl.com/oodyjue$