Examen UIII
Deteccion de Puntos de Interes.

Angel Manrique Pozos Flores
Francisco Fernando Zuñiga Alvarez
Instituto Tecnológico Nacional de México,
Blvd. Industrial, Mesa de Otay, 22430 Tijuana, B.C., México.

12 de abril de 2016

Implementar en OpenCV el detector de puntos de interés propuestos en la ecuación 14 del articulo "Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming". Gustavo Olague, Leonardo Trujillo. Image and Vision Computing, 29, 484-498, 2011.

Donde se sugiere crear una barra de desplazamiento para ir variando el peso de W y mostrar el resultado de aplicar el operador propuesto, con diferentes valores de W en una misma imagen, notese que el ultimo paso requerido es la supresión de no máximos.

1. Operadores de puntos de interes.

Se utilizaron las ecuaciones obtenidas por Gustavo Olague y Leonardo Trujillo en [2] las cuales son.

$$\begin{array}{rcl} K^1_{MO} = & G1 * \log(G_1 * I^2) & \\ K^2_{MO} = & G2 * \mid G_1 * I - I \mid . & \\ K^3_{MO} = & \frac{G_1 * I}{I} & \end{array} \tag{1}$$

Siendo K_{MO}:

$$K_{MO} = G_2 * \mid K^1_{MO} + W \cdot K^2_{MO} \mid^2 \tag{2}$$

A su vez se utilizo también el algoritmo para la supresión de no máximos propuesto por Hilton Bristow que se basa en el trabajo de investigación de Alexander Neubeck y Luc Van Gool [1], los artículos [3, 4] nos sirvieron de marco de referencia para poder entender como se aplican este tipo de operadores donde se puede hacer una comparación con el desempeño que muestran otros no basados en Programación Genética (GP) asi como su aplicacion de estos.

Basándonos en las ecuaciones descritas en el trabajo propuesto 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 función MOP involucra las ecuaciones descritas, donde posteriormente se realiza una supresión de no máximos basada en el código desarrollado por Hilton Bristow.

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

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <stdio.h>

using namespace cv;
using namespace std;

Mat img, img2, img3, KMO1, KMO2, KMO3, KMO, dst, image
    , Mask, gris, clear;
Mat KMO_norm, KMO_abs;

int w = 50;
int W_slider = 0;
int minDistance = 10;
int QL = 50;
double qualityLevel = (QL)*0.01;
```

```
double W;
const double W_max = 20;
char* source_window = "Multi-Objetive Parameterized
    Interest Point Detector (MOP)";

/// Function headers
void nonMaximaSuppression(const Mat& src, Mat& dst1,
    const int sz, double qualityLevel, const Mat mask1)
    ;
void MOP(int, void*);


/// Multi-Objetive Parameterized Interest Point
    Detector (MOP)
int main()
{
        img = imread("lena.jpg", CV_LOAD_IMAGE_COLOR);

        gris = img.clone();
        cvtColor(gris, gris, CV_BGR2GRAY);
        clear = gris.clone();

        img.convertTo(img, CV_32F);
        cvtColor(img, image, CV_BGR2GRAY);

        namedWindow(source_window, CV_WINDOW_AUTOSIZE)
            ;
        createTrackbar("W:", source_window, &W_slider,
            W_max, MOP);
        imshow(source_window, gris);

        MOP(0, 0);

        waitKey(0);
        return 0;
}

/// MOP FUNCTION
void MOP(int, void*)
{
        gris = clear.clone();
```

```
    W = (double)W_slider / W_max;
    cout << W << endl;

    // KMO_1
    pow(image, 2, img2);
                                          // I^2
    GaussianBlur(img2, img2, Size(9, 9), 1.0, 1.0)
        ;       //G1*I ^ 2
    log(img2, img2);
                                              // log(
       G1*I^2)
    GaussianBlur(img2, KMO1, Size(9, 9), 1.0, 1.0)
        ;       // G1*log(G1*I^2)

// KMO_2
    GaussianBlur(image, img3, Size(9, 9), 1.0,
        1.0);       // G1*I
    absdiff(img3, image, img3);
                                    // |(G1*I)-I|
    GaussianBlur(img3, KMO2, Size(9, 9), 2.0, 2.0)
        ;       // G2 * |(G1*I)-I|
    KMO2 = (W * KMO2);

    // KMO_3
    //GaussianBlur(image, KMO3, Size(9, 9), 1.0,
        1.0);// G1 * I
    //divide(KMO3, image, KMO3); // (G1 * I)/I

    /// KMO
    KMO = KMO1 + KMO2;
                                              //   KMO_1
       + (W x KMO_2)
// KMO = KMO1 + KMO2 + KMO3;
                                  //KMO_1 + (W x KMO_2) +
   KMO_3
KMO = abs(KMO);
    pow(KMO, 2, KMO);
                                              //[KMO_1
       + (W x KMO_2)]^2
    GaussianBlur(KMO, KMO, Size(9, 9), 2.0, 2.0);
```

```
                // G2 * [KMO_1 + (W x KMO_2)]^2

        normalize(KMO, KMO_norm, 0, 255, NORM_MINMAX);
        convertScaleAbs(KMO_norm, KMO_abs);
        imshow("KMO", KMO_abs);

        if (Mask.empty())
        {
                Mask = Mat::zeros(KMO.size(), CV_8UC1)
                    ;
                Mask(Rect(minDistance, minDistance, (
                   KMO.cols) - (2 * minDistance), (KMO
                    .rows) - (2 * minDistance))) = 1;
        }

        //Non Maxima Suppresion application
        nonMaximaSuppression(KMO, dst, minDistance,
            qualityLevel, Mask);

        imshow("Mask", dst);

        for (int j = 0; j < dst.rows; j++)
        {
                for (int i = 0; i < dst.cols; i++)
                {
                        if ((int)dst.at<uchar>(j, i) >
                            200)
                        {
                                circle(gris, Point(i,
                                   j), 4, Scalar(255,
                                   255, 255), 2, 8, 0)
                                   ;
                        }
                }
        }

        imshow(source_window, gris);
}

/// NMS
void nonMaximaSuppression(const Mat& src, cv::Mat&
```

```
    dst1, const int sz, double qualityLevel, const cv::
    Mat mask1)
{

        double minStrength;
        double maxStrength;
        int threshold1;

        minMaxLoc(src, &minStrength, &maxStrength);
        threshold1 = qualityLevel*maxStrength;
        threshold(src, src, threshold1, 255, 3);

        const int M = src.rows;
        const int N = src.cols;
        const bool masked = !mask1.empty();
        Mat block = 255 * Mat_<uint8_t>::ones(Size(2 *
            sz + 1, 2 * sz + 1));
        dst1 = Mat_<uint8_t>::zeros(src.size());

        for (int m = 0; m < M; m += sz + 1)
        {
                for (int n = 0; n < N; n += sz + 1)
                {
                        Point ijmax;
                        double vcmax, vnmax;

                        Range ic(m, min(m + sz + 1, M)
                            );
                        Range jc(n, min(n + sz + 1, N)
                            );
                        minMaxLoc(src(ic, jc), NULL, &
                            vcmax, NULL, &ijmax, masked
                             ? mask1(ic, jc) : noArray
                            ());
                        Point cc = ijmax + Point(jc.
                            start, ic.start);
                        Range in(max(cc.y - sz, 0),
                            min(cc.y + sz + 1, M));
                        Range jn(max(cc.x - sz, 0),
                            min(cc.x + sz + 1, N));
```

```
                                Mat_<uint8_t> blockmask;
                                block(Range(0, in.size()),
                                   Range(0, jn.size())).copyTo
                                   (blockmask);
                                Range iis(ic.start - in.start,
                                    min(ic.start - in.start +
                                   sz + 1, in.size()));
                                Range jis(jc.start - jn.start,
                                    min(jc.start - jn.start +
                                   sz + 1, jn.size()));
                                blockmask(iis, jis) = Mat_<
                                   uint8_t>::zeros(Size(jis.
                                   size(), iis.size()));
                                minMaxLoc(src(in, jn), NULL, &
                                   vnmax, NULL, &ijmax, masked
                                    ? mask1(in, jn).mul(
                                   blockmask) : blockmask);

                                if (vcmax > vnmax)
                            {

                                        dst1.at<uint8_t>(cc.y,
                                            cc.x) = 255;

                            }
                            }
                    }
}
```

Obteniendo las imágenes finales mostradas a continuación.

Para el valor de W = 0 obtuvimos que.

Figura 2: Original aplicando MOP con un valor de W = 0

Figura 3: Imagen con KMO aplicado con un valor de W = 0.

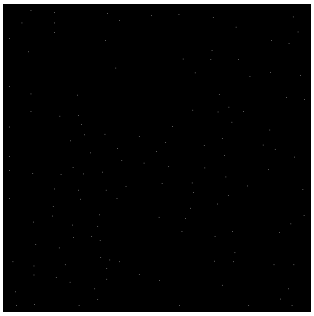

Figura 4: Mascara que contiene los puntos de interés para W = 0.

Y para un valor de W = 0,05 obtuvimos:

Figura 5: Original aplicando MOP con un valor de W = 0.05

Figura 6: Imagen con KMO aplicado con un valor de W = 0.05

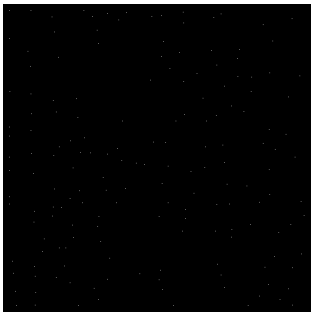

Figura 7: Mascara que contiene los puntos de interés para W = 0.05.

Y para un valor de W = 0,5 obtuvimos:

Figura 8: Original aplicando MOP con un valor de W = 0.5

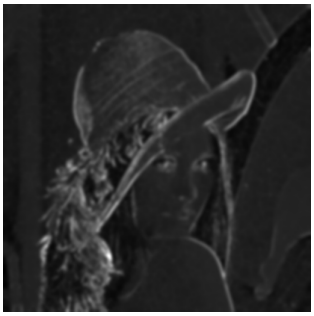

Figura 9: Imagen con KMO aplicado con un valor de W = 0.5

Figura 10: Mascara que contiene los puntos de interés para W = 0.5

Y finalmente para el valor de W = 1 obtuvimos:

Figura 11: Original aplicando MOP con un valor de W = 1

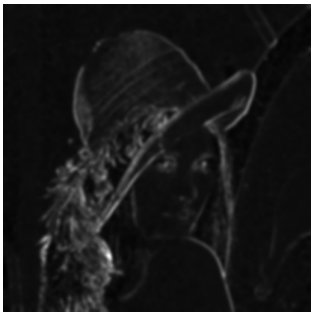

Figura 12: Imagen con KMO aplicado con un valor de W = 1

Figura 13: Mascara que contiene los puntos de interés para W = 1

El programa contiene a su vez una barra para variar el valor de W mostrando los incrementos en terminal como se muestra en la Figura 14.

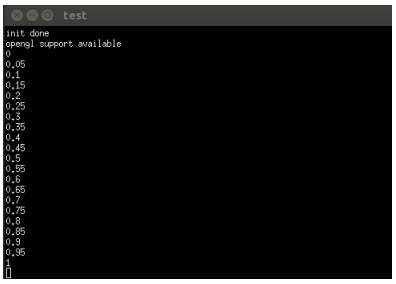

Figura 14: Salida en terminal para mostrar los incrementos de W utilizando la barra de desplazamiento.

2. Bibliografía

1 Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In Proceedings of the 18th International Conference on Pattern Recognition - Volume 03, ICPR '06, pages 850–855, Washington, DC, USA, 2006. IEEE Computer Society.

2 Gustavo Olague and Leonardo Trujillo. Interest point detection through multiobjective genetic programming. Appl. Soft Comput., 12(8):2566–2582, 2012.

3 Gustavo Olague and Leonardo Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. Image Vision Comput., 29(7):484–498, June 2011.

4 Leonardo Trujillo and Gustavo Olague. Automated design of image operators that detect interest points. Evol. Comput., 16(4):483–507, 2008.