Tarea 14

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 obtiene la transformada de Fourier de una imagen y se realiza la implementacion de un filtro Butterworth.

1. Introducción

El filtrado en el dominio de la frecuencia consiste en obtener la transformada de Fourier, aplicar al multiplica el filtro deseado y calcular la transformada inversa para regresar al dominio espacial.

2. Filtrado en el dominio de la frecuencia

Existen muchas clases de filtros que se aplican en el dominio de la frecuencia, dos de los filtros mas comunes son los llamados *filtro ideal* y el *filtro Butterworth* ambos tipos de filtros pueden ser pasa-altos y pasa-bajos, el filtro ideal pasa-bajas tiene una funcion de transferencia H(u,v) que es igual a 1 para todas las frecuencias menores a cierto valor D_0 y 0 para las demas frecuencias.

Un filtro ideal pasa-altas tiene la funcion de transferencia opuesta, es decir que es 0 para todas las frecuencias menores a cierto valor D_0 y 1 para todas las demas frecuencias.

La transformada de Fourier descompone una imagen en componentes utilizando senos y cosenos, en otras palabras transforma una imagen que esta en el dominio espacial a el dominio de la frecuencia, la idea principal es que una funcion puede ser aproximada mediante la suma de funciones de senos y cosenos, la

transformada de Fourier es una manera de poder realizar esto.

Matematicamente una transformada de Fourier para dos dimensiones se representa como:

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)e^{-i2\pi(\frac{ki}{N}+\frac{lj}{N})} \qquad e^{ix} = \cos x + i\sin x \tag{1}$$

Donde f is el valor de la imagen en el dominio espacial y F es en el dominio de la frecuencia, el resultado de la transformada son números complejos.

Para llevar a cabo la transformación es necesario seguir una serie de pasos descritos a continuación.

- Expandir la imagen a un tamaño optimo.
- Crear una variable que contenga los numeros reales y los numeros complejos.
- Calcular la transformada discreta de Fourier.
- Calcular la magnitud utilizando los valores reales y complejos

$$M = \sqrt[2]{Re(DFT(I))^2 + Im(DFT(I))^2} \tag{2}$$

- Pasar a la escala logaritmica.

$$M_1 = \log(1+M) \tag{3}$$

- Cortar y reordenar.
- Finalmente hacer una normalizacion.

3. Filtro Butterworth

El filtro de Butterworth es uno de los filtros electrónicos más básicos, diseñado para producir la respuesta más plana que sea posible hasta la frecuencia de corte.

El filtro Butterworth más básico es el típico filtro pasa bajo de primer orden, el cual puede ser modificado a un filtro pasa alto o añadir en serie otros formando un filtro pasa banda o elimina banda y filtros de mayores órdenes, el filtro de Butterworth es el único filtro que mantiene su forma para órdenes mayores (sólo con

una pendiente mayor a partir de la frecuencia de corte).

Este tipo de filtros elimina ruidos externos y ayuda a evitar la contaminación de la red por causa de algunos ruidos.

Si llamamos H a la respuesta en frecuencia, se debe cumplir que las 2N - 1 primeras derivadas de $\mid H(w) \mid^2$ sean 0 para $w = 0$ y $w = \infty$ unicamente posee polos, donde su función de transferencia es:

$$\mid H(w) \mid^2 = \frac{1}{1 + (w/w_c)^{2N}} \tag{4}$$

Donde N es el orden del filtro, w_c es la frecuencia de corte y w es la frecuencia analogica compleja (w = jw).

Basandonos en las ecuaciones podemos ser capaces de desarrollar un algoritmo en C++ utilizando las librerías de visión de OpenCV.

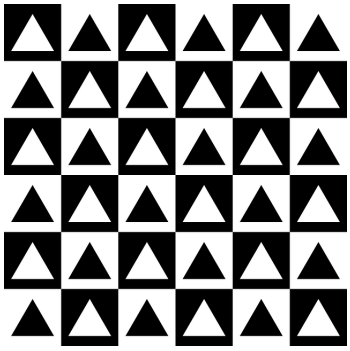

Figura 1: Imagen de prueba para el algoritmo de DFT y las pruebas de Butterworth.

Para el caso del calculo de la transformada discreta de Fourier (DFT) se tiene que:

```
///DFT
#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;
```

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

    Mat I = imread(argv[1] = "chessb.jpg",
       CV_LOAD_IMAGE_GRAYSCALE);

    if( I.empty())
        return -1;

    Mat padded;                                //expand
       input image to optimal size
    int m = getOptimalDFTSize( I.rows );
    int n = getOptimalDFTSize( I.cols ); // on the
       border add zero values
    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.
       cols, BORDER_CONSTANT, Scalar::all(0));

    Mat planes[] = {Mat_<float>(padded), Mat::zeros(
       padded.size(), CV_32F)};
    Mat complexI;
    merge(planes, 2, complexI);         // Add to the
       expanded another plane with zeros
    dft(complexI, complexI);            // this way
       the result may fit in the source matrix
    split(complexI, planes);                   //
       planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
    magnitude(planes[0], planes[1], planes[0]);//
       planes[0] = magnitude
    Mat magI = planes[0];

    magI += Scalar::all(1);                    //
       switch to logarithmic scale
    log(magI, magI);

    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows &
        -2));

    int cx = magI.cols/2;
    int cy = magI.rows/2;
```

```
    Mat q0(magI, Rect(0, 0, cx, cy));    // Top-Left -
       Create a ROI per quadrant
    Mat q1(magI, Rect(cx, 0, cx, cy));   // Top-Right
    Mat q2(magI, Rect(0, cy, cx, cy));   // Bottom-Left
    Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-
       Right

    Mat tmp;                             // swap
       quadrants (Top-Left with Bottom-Right)
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);

    q1.copyTo(tmp);                      // swap
       quadrant (Top-Right with Bottom-Left)
    q2.copyTo(q1);
    tmp.copyTo(q2);

    normalize(magI, magI, 0, 1, CV_MINMAX); //
       Transform the matrix

    imshow("Input Image"       , I   );    // Show the
        result
    imshow("spectrum magnitude", magI);

    waitKey();

    return 0;
}
```

Obteniendo la imagen mostrada a continuación.

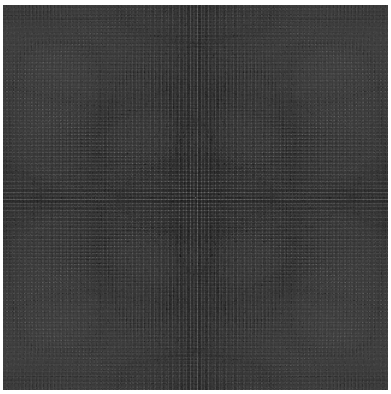

Figura 2: Imagen obtenida de la transformacion aplicando la transformada discreta de Fourier.

Para el caso del filtro de Butterworth se tiene el siguiente algoritmo.

```
//BUTTERWORTH DFT
#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;


void shiftDFT(Mat &fImage )
{
    Mat tmp, q0, q1, q2, q3;

    // first crop the image, if it has an odd number
       of rows or columns

    fImage = fImage(Rect(0, 0, fImage.cols & -2,
       fImage.rows & -2));
```

```
    int cx = fImage.cols / 2;
    int cy = fImage.rows / 2;

    // rearrange the quadrants of Fourier image
    // so that the origin is at the image center

    q0 = fImage(Rect(0, 0, cx, cy));
    q1 = fImage(Rect(cx, 0, cx, cy));
    q2 = fImage(Rect(0, cy, cx, cy));
    q3 = fImage(Rect(cx, cy, cx, cy));

    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);

    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);
}



Mat create_spectrum_magnitude_display(Mat &complexImg,
    bool rearrange)
{
    Mat planes[2];

    // compute magnitude spectrum (N.B. for display)
    // compute log(1 + sqrt(Re(DFT(img))**2 + Im(DFT(
       img))**2))

    split(complexImg, planes);
    magnitude(planes[0], planes[1], planes[0]);

    Mat mag = (planes[0]).clone();
    mag += Scalar::all(1);
    log(mag, mag);

    if (rearrange)
    {
        // re-arrange the quaderants
```

```
        shiftDFT(mag);
    }

    normalize(mag, mag, 0, 1, CV_MINMAX);

    return mag;

}

void create_butterworth_lowpass_filter(Mat &dft_Filter
    , int D, int n, int W)
{
    Mat tmp = Mat(dft_Filter.rows, dft_Filter.cols,
        CV_32F);

    Point centre = Point(dft_Filter.rows / 2,
        dft_Filter.cols / 2);
    double radius;

    // based on the forumla in the IP notes (p. 130 of
        2009/10 version)
    // see also HIPR2 on-line

    for (int i = 0; i < dft_Filter.rows; i++)
    {
        for (int j = 0; j < dft_Filter.cols; j++)
        {
            radius = (double) sqrt(pow((i - centre.x),
                2.0) + pow((double) (j - centre.y),
                2.0));

            // Butterworth low pass:
            // tmp.at<float>(i,j) = (float)
            //                      ( 1 / (1 + pow((
                double) (radius / D), (double) (2 * n)
                ))));

            // Butterworth band reject, page 244,
                paragraph 5.4.1, Gonzalez Woods, "
                Digital Image Processing 2nd Edition"
            // D(u,v) -> radius
```

```
                // D_0 -> D
                tmp.at<float>(i, j) = (float)
                                      ( 1 / (1 + pow((
                                         double) (radius *
                                          W) / ( pow((
                                         double)radius, 2)
                                          - D * D ), (
                                         double) (2 * n)))
                                         );
            }
        }

        Mat toMerge[] = {tmp, tmp};
        merge(toMerge, 2, dft_Filter);
}


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

        Mat img, imgGray, imgOutput;  // image object(s)
        img = imread(argv[1] = "chessb.jpg",
           CV_LOAD_IMAGE_COLOR);

        Mat padded;         // fourier image objects and
           arrays
        Mat complexImg, filter, filterOutput;
        Mat planes[2], mag;

        int N, M; // fourier image sizes

        int radius = 20;                // low pass filter
           parameter
        int order = 2;                  // low pass filter
           parameter
        int width = 3;

        const string originalName = "Input Image (
           grayscale)"; // window name
        const string spectrumMagName = "Magnitude Image (
           log transformed)"; // window name
```

```
const string lowPassName = "Butterworth Low Pass
    Filtered (grayscale)"; // window name
const string filterName = "Filter Image"; //
    window nam

bool keepProcessing = true;    // loop control flag
int  key;                      // user input
int  EVENT_LOOP_DELAY = 40;    // delay for GUI
    window


    namedWindow(originalName, 0);
    namedWindow(spectrumMagName, 0);
    namedWindow(lowPassName, 0);
    namedWindow(filterName, 0);


    // setup the DFT image sizes
    M = getOptimalDFTSize( img.rows );
    N = getOptimalDFTSize( img.cols );

    // add adjustable trackbar for low pass filter
        threshold parameter
    createTrackbar("Radius", lowPassName, &radius,
        (min(M, N) / 2));
    createTrackbar("Order", lowPassName, &order,
        10);
    createTrackbar("Width", lowPassName, &width, (
        min(M, N) / 2));

    // start main loop
    while (keepProcessing)
    {

        // convert input to grayscale
        cvtColor(img, imgGray, CV_BGR2GRAY);

        // setup the DFT images
        copyMakeBorder(imgGray, padded, 0, M -
            imgGray.rows, 0,
                       N - imgGray.cols,
```

```
                    BORDER_CONSTANT, Scalar
                    ::all(0));
planes[0] = Mat_<float>(padded);
planes[1] = Mat::zeros(padded.size(),
   CV_32F);

merge(planes, 2, complexImg);

// do the DFT
dft(complexImg, complexImg);

// construct the filter (same size as
   complex image)
filter = complexImg.clone();
create_butterworth_lowpass_filter(filter,
   radius, order, width);

// apply filter
shiftDFT(complexImg);
mulSpectrums(complexImg, filter,
   complexImg, 0);
shiftDFT(complexImg);

// create magnitude spectrum for display
mag = create_spectrum_magnitude_display(
   complexImg, true);

// do inverse DFT on filtered image
idft(complexImg, complexImg);

// split into planes and extract plane 0
   as output image
cv::Mat myplanes[2];
split(complexImg, myplanes);
double minimum = -1;
double maximum = -1;
cv::Point minloc(-1, -1), maxloc(-1, -1);
minMaxLoc(myplanes[0], &minimum, &maximum,
    &minloc, &maxloc);
std::cout << "min=" << minimum << "@" <<
   minloc << "\tmax=" << maximum << "@" <<
```

```
                maxloc << "\n";
            //normalize(myplanes[0], imgOutput, 0, 1,
               CV_MINMAX);
            imgOutput = myplanes[0];

            // do the same with the filter image
            split(filter, planes);
            normalize(planes[0], filterOutput, 0, 1,
               CV_MINMAX);

            // display image in window
            imshow(originalName, imgGray);
            imshow(spectrumMagName, mag);
            imshow(lowPassName, imgOutput);
            imshow(filterName, filterOutput);

            // start event processing loop (very
               important,in fact essential for GUI)
            // 40 ms roughly equates to 1000ms/25fps =
                4ms per frame
            key = waitKey(EVENT_LOOP_DELAY);

            if (key == 'x')
            {

                // if user presses "x" then exit to
                   exit from infinite while statement
                std::cout << "Keyboard exit requested"
                          << std::endl;
                keepProcessing = false;
            }
        }

        return 0;
}

double deg2rad( double a )
{
    return (M_PI*a)/180;
}
```

```
void create_a_model()
{
    cv::Mat img(360, 360, cv::DataType<unsigned char
        >::type);
    for ( size_t i = 0; i < img.cols; i+=1 )
    {
        for ( size_t j = 0; j < img.rows; j+=1 )
        {
            img.at<unsigned char>(i,j) = 127 + 127 *
                sin(deg2rad(20*j));
        }

    }
    imwrite("sin.bmp", img);
}
```

Obteniendo la siguiente serie de imagenes las cuales pueden ser modificadas para variar el orden, el radio y el ancho del filtro, mientras que las ventanas nos muestran los valores del max y el min, la magnitud calculada de la imagen, la imagen filtrada y la imagen original que esta siendo utilizada.

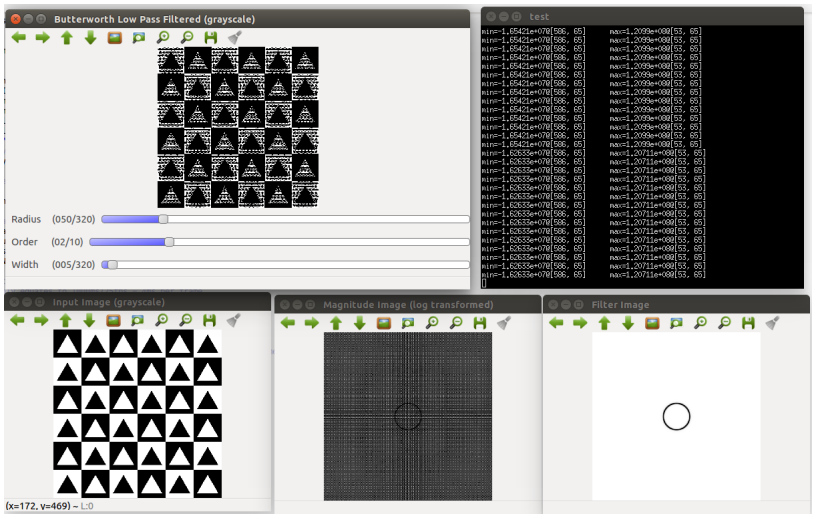

Figura 3: Radio = 50, orden = 2, Ancho = 005

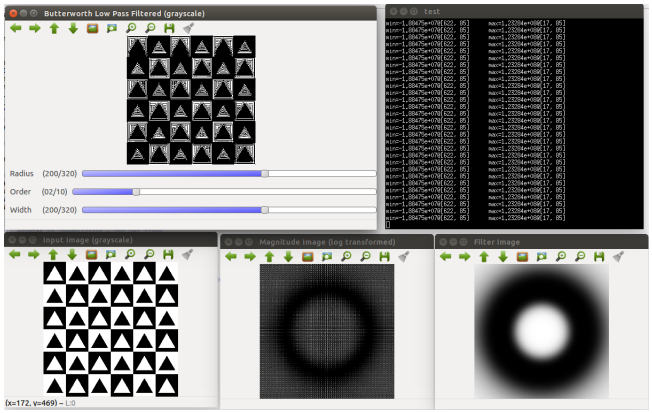

Figura 4: Radio = 200, orden = 2, Ancho = 200

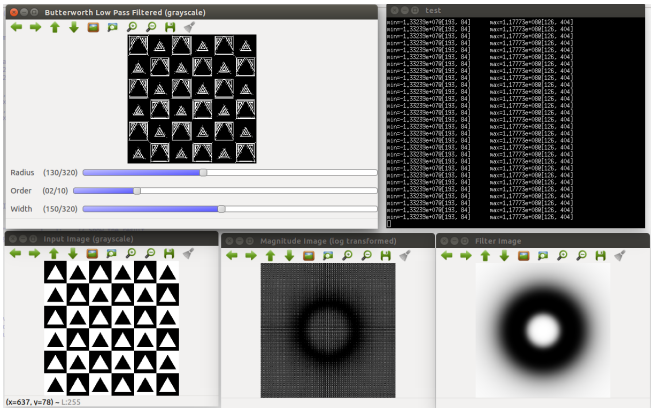

Figura 5: Radio = 130, orden = 2, Ancho = 150

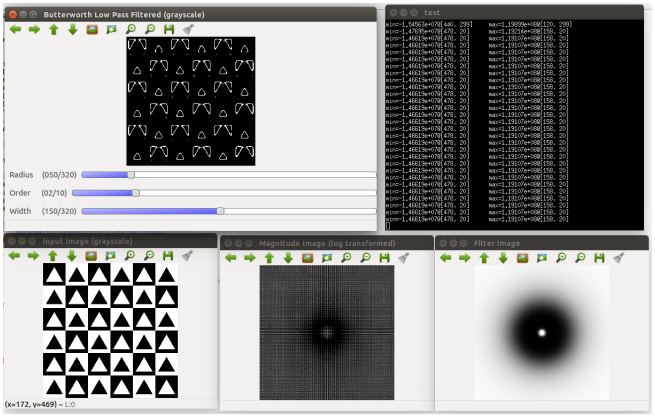

Figura 6: Radio = 50, orden = 2, Ancho = 150

4. Conclusiones

Es importante saber que existen otros metodos para el tratamiento de la señales (imagenes) donde podemos pasar de un dominio espacial a uno basado en la frecuencia, esto nos permite representar una señal o un sistema LTI de uno a otro dominio sin que haya perdida de informacion.

Siendo el filtro de Butterworth una de las herramientas mas utilizadas para el tratamiento de señales este filtro presenta muchas caracteristicas que lo hacen muy util ya que puede ser utilizado como un filtro pasa-bajas y un filtro pasa-altas que sirve para el filtrado de ruidos principalmente.

Por ello tratar nuestra señal en el dominio de la frecuencia nos da la oprtunidad de poder hacer un analisis mas detallado de la señal que se esta tratando.

5. Bibliografía

- Discrete Fourier Transform.
  ($http://tinyurl.com/qgmrfg3$)
- Alessando Gentilini, 2013.
  ($http://tinyurl.com/pgjseym$)