Instituto Tecnológico de Buenos Aires

Ingeniería Informática

Métodos Numéricos Avanzados

==========

Trabajo Práctico Número 2

Filtrado de imágenes utilizando transformada de Fourier discreta

Profesora Noni, Laura

Castiglione, Gonzalo

Castro Peña, Gonzalo

Wassington, Axel

Procesamiento de imágenes I++

Abstract

Este informe tiene como objetivo analizar espectralmente una imágen en escala de grises y utilizar la transformada discreta de Fourier para realizar filtrados espaciales. Se presentarán diferentes tipos de filtros y el resultado de aplicarlos sobre una imágen.

1 Introducción

El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad, facilitar la búsqueda de información o comprimirlas.
El proceso de filtrado es el conjunto de técnicas englobadas dentro del preprocesamiento de imágenes cuyo objetivo fundamental es obtener, a partir de una imágen origen, otra final cuyo resultado sea más adecuado para una aplicación específica. Cambiando así ciertas características de la misma para facilitar operaciones de procesado sobre esta.
Los principales objetivos que se persiguen con la aplicación de filtros son:

- Suavizar la imágen: reducir la cantidad de variaciones de intensidad de color entre píxeles vecinos.
- Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imágen como en el de transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imágen.
- Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la intensidad luminosa.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imágen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

El proceso de filtrado puede llevarse a cabo sobre los dominios de frecuencia y/o espacio.

2 Desarrollo

2.1 Implementación de un programa que computa la TDF 2D.

A fin de simplificar las explicaciónes sobre la implementación del programa, primero se presentarán las diferentes transformadas de Fourier en las que este informe esta basado.

A continuación se presenta la ecuación de la Transformada Discreta de Fourier de una secuencia bidimensional $x_{n,m}$ de tamaño $N \times N$ (imágen), la cual se define como:

$$X_{l,k} = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_{n,m} e^{-i\frac{2\pi}{N}(nl+mk)} \tag{1}$$

Transformada Inversa Discreta de Fourier de una secuencia bidimensional:

$$x_{n,m} = \frac{1}{N^2} \sum_{l=0}^{N-1} \sum_{k=0}^{N-1} X_{l,k} e^{i\frac{2\pi}{N}(nl+mk)} \tag{2}$$

Basandose en la fórmula (1) es posible reescribir *la Transformada Discreta de Fourier de una secuencia bidimensional* en función de *Transformadas Discretas de secuencias unidireccionales* de la siguiente manera:

$$X_{l,k} = \sum_{l=0}^{N-1} e^{-i\frac{2\pi}{N}nl} (\sum_{k=0}^{N-1} X_{l,k} e^{-i\frac{2\pi}{N}mk}), \tag{3}$$

de manera análoga se puede proceder con la transformada inversa. [6]

Debido a la cantidad de procesamiento que el cálculo de la *Transformada Discreta de Fourier para secuencias bidimensional* implica, el cómupto de la tranformada para una imágen de $400x400$ pixeles podría demorarse hasta dos horas [1]. Volviendo su aplicación impráctica.
Frente a este problema, se optó por la realización de una implementación de la *Transformada Rápida de Fourier (FFT)* recursiva para secuencias *unidireccionales* [2].

Debido a que implementacíon realizada de la FFT requeire que las dimensiones de la imágen original sean potencias de 2 (requerimiento bastante restrictivo) se optó por la complementar a esta con una que compute la transformada "original" sobre la parte de la imágen que la FFT recursiva no puede.

Una segunda opción tenida en cuenta, consistía en realizar un *paddeo* con ceros hasta completar a la minima potencia de 2 mayor a lo que se va a transformar (es decir, para una imágen de $400x400$ pixeles, se la llevaría a $512x512$ pixeles completando con el color negro). Esta alternativa fue desestimada ya que requiere retoques sobre la imágen a procesar.

La implementación completa del algoritmo último puede encontrarse en la sección *Anexo*, el en punto (1).

Puede encontrarse otras implementaciónes de este algoritmo por ejemplo en *Matlab* u *Octave*[3] con el nombre de $fft2$. Esta implementación utiliza una técnica de "*divide and conquer*" y esta además desarrollada en un lenguaje compilado, consiguiende de esa manera, aún mejor performance con respecto a la implementación presentada en el informe.

2.2 Visualización de la imágen a tratar

El archivo *saturno* contiene una matriz de $400x400$ píxeles y corresponde a niveles de intensidad luminosa comprendidos entre 0 y 255.

Para visualizar esta imágen en escala de grises, es necesario establecer un mapa de color de 255 niveles.

En la Figura 1 se muestra una imágen del planeta Saturno, capturada durante la misión Voyager.

[1] Tiempo medido en una compuladora con procesador Intel *i*3 y *Ubuntu* 13
[2] Puede encontrarse una implementación de la misma en la refencia [5]
[3] En el presente informe se utilizará la versión 3.6.4 de GNU Octave

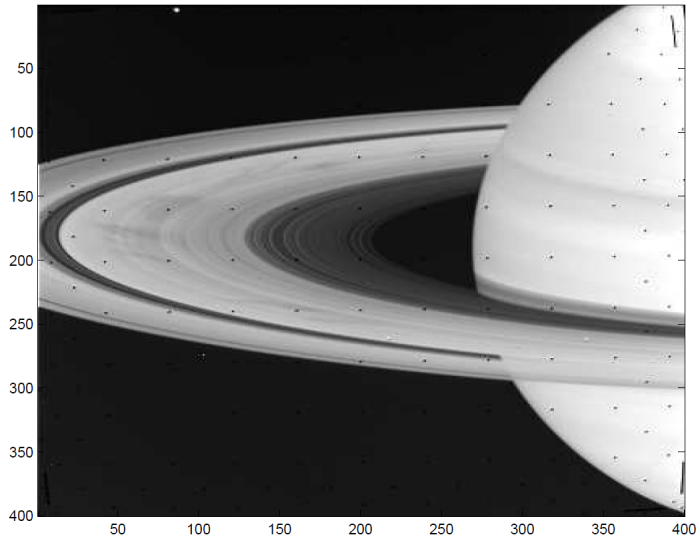

Figure 1: Imágen de saturno obtenida a partir de las mediciones

2.3 Visualización de la amplitud y fase de la transformada sobre la imágen

En esta sección se va a trabajar con la transformada de la imágen de *saturno* y se pretende dar una visualización de lo obtenido a modo de entender gráficamente con lo que se esta tratando.

2.3.1 Amplitud

Para el cálculo de la amplitud, dada X (transformada de fourier de x), se utilizó el módulo del número complejo obtenido sobre cada valor de X. Para normalizarlo se utilizaron dos técnicas.

La primer técnica consiste en realizar un mapeo lineal entre el intervalo de valores que toma la matriz transformada de Fourier y el intervalo $[0 - 255]$ utilizando la siguiente fórmula:

$$Output_i = floor((\frac{absol_i}{maxAbsol}) * 255), \tag{4}$$

donde $maxAbsol$ es el máximo de las amplitudes, $absol_i$ es cada una de las amplitudes y $Output_i$ es el píxel de salida.

Como se puede ver en la fórmula (4), para lograr dicho mapeo, primero se divide el intervalo original por el máximo valor absoluto (se lo lleva a $[0, 1]$), luego se multiplica por 255 para dejarlo en el intervalo deseado. El problema con esto es que todavía se tienen valores reales (aunque dentro del intervalo), mientras que la escala exige valores enteros. Es por esto último que se toma la parte entera inferior de lo obtenido anteriormente.

Una vez realizado el procedimiento, se obtuvo la imágen mostrada en la figura 2.

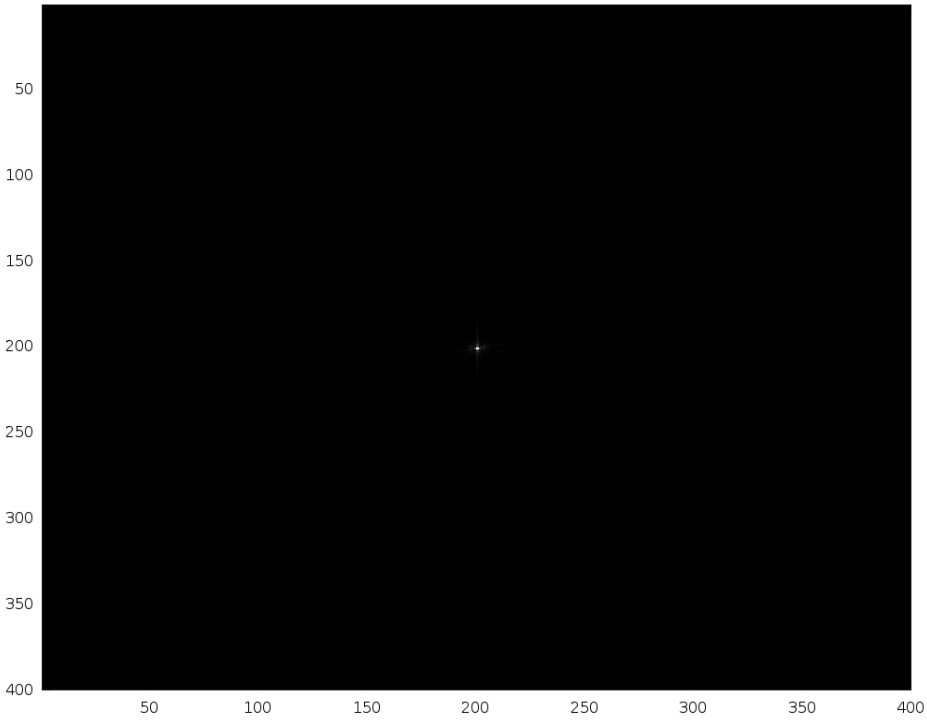

Figure 2: Imágen de la amplitud de cada píxel de la transformada de Fourier de la figura 1

Como se puede observar en la figura 2, hay un grupo de frecuencias (centro de la figura) muy superiores que el resto. Pero debido a la técnica de mapeo utilizada, no es posible decir mucho más al respecto.

Si en cambio, se utiliza una escala logarítmica como la siguiente:

$$Output_i = \frac{255}{log(1 + Absol_i)}, \tag{5}$$

donde $absol_i$ es cada una de las amplitudes y $Output_i$ es el píxel de salida.

Con este nuevo procedimiento, la imágen que se obtiene es la siguiente:

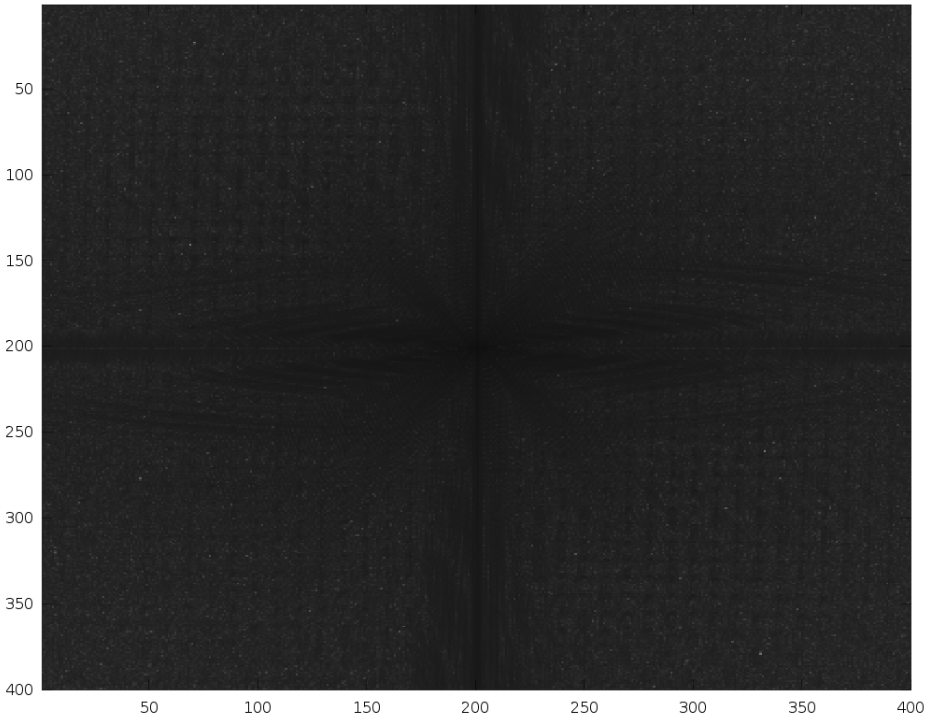

Figure 3: Imágen en escala logarítmica de la amplitud de cada píxel de la transformada de Fourier de la figura 1.

Aplicando dicha fórmula, se obtiene una visualización más interesante ya que se puede distinguir (además del pico de frecuencia central) una trama en el resto de las frecuencias.

Nota: Ambas imágenes de la amplitud de la frecuencia están "shifteadas" para que la frecuencia 0 *quede en el centro. Es decir,* se parte la imágen en dos columnas y se cambia el orden. Luego se parte la imágen en dos filas y nuevamente se cambia el orden. El código para este cambio se presenta en la sección *Anexo*, en el punto 2.

2.3.2 Fase

Para el cálculo de la fase, se utilizó el argumento del número complejo obtenido sobre cada valor de X. Para normalizarlo se utilizó la primera técnica de mapeo, la de mapeo lineal:

$$Output_i = floor(((\frac{args_i}{2 * maxArg}) + 1/2) * 255), \tag{6}$$

donde $maxArg$ es el argumento de mayor valor absoluto, $args_i$ es cada uno de los argumentos y $Output_i$ es el píxel de salida.

Como se puede ver en la fórmula (6), para lograr dicho mapeo, primero se divide el intervalo original por dos veces el máximo valor absoluto (se lo lleva a $[-0.5, 0.5]$), luego se suma 0.5 y multiplica por 255 para dejarlo en el intervalo deseado. El problema con esto es que todavía se tienen valores reales (aunque dentro del intervalo), mientras que la escala exige valores enteros. Es por esto último que se toma la parte entera inferior de lo obtenido anteriormente.

Se pueden obvservar los resulatdos en la figura 4.

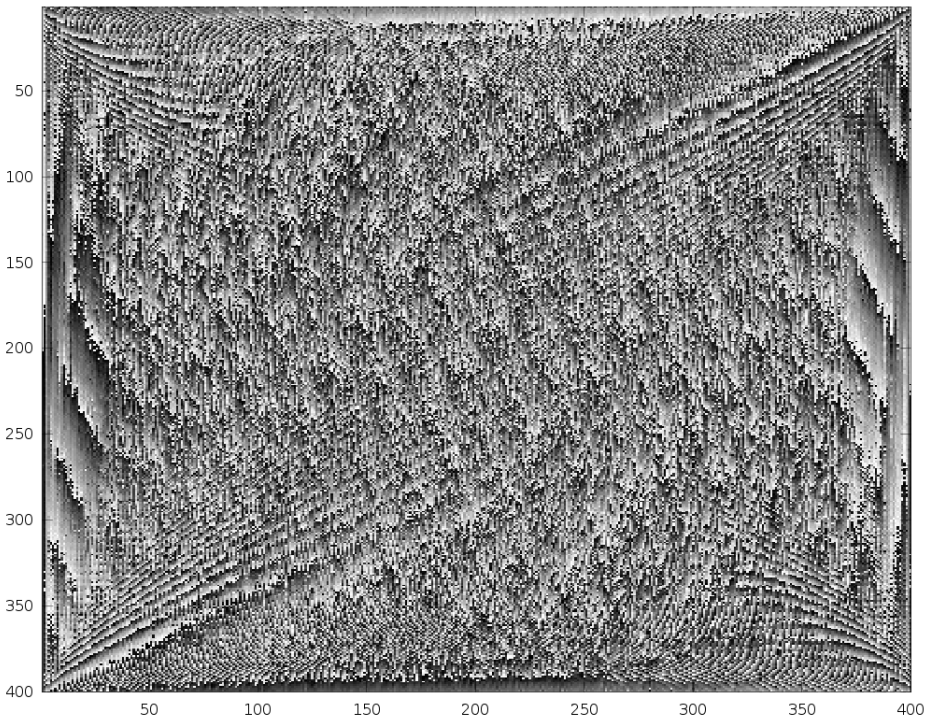

Figure 4: Imágen de la fase de cada píxel de la transformada de fourier de la figura 1.

La matriz con los valores obtenidos para la amplitud y fase, pueden encontrarse en los archivos *absolutos.mat* y *argumentos.mat* respectivamente.

2.4 Aplicación de filtros

El papel de un filtro en frecuencia es modificar la fase y amplitud de las componentes obtenidas luego de realizar una transformación. Para la aplicación de un filtro se requiere de los siguientes pasos:

1. Transformar los valores a una escala de frecuencias
2. Punto a punto, multiplicar el valor del punto por el valor del filtro para el punto
3. Anti-transformar la señal filtrada a escala de tiempo.
4. Tomar la parte real de la señal filtrada

Este proceso completo se puede apreciar gráficamente en la la figura 5 [4]:

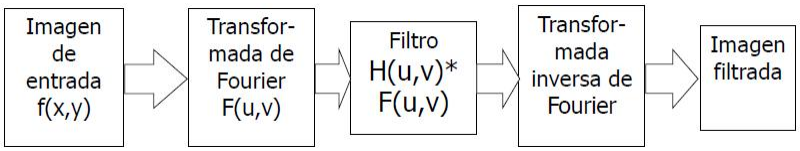

Figure 5: Proceso de filtrado de una imágen utilizando filtros en dominio de frecuencia

El programa utilizado para ejecutar este proceso se encuentra en la sección *Anexo*, en el punto 3.

Debido a la simplicidad de las condiciones de cada filtro, el código utilizado no se presenta en este informe pero puede ser encontrado en los archivos $filtro1.m$, $filtroGauss.m$ y $filtroDamero.m$

2.4.1 Filtro 1

$$H_{k,l} = \begin{cases} 0 & 0 \leq k \leq 400,\ 190 \leq\ l \leq 210 \\ 0 & 0 \leq l \leq 400, 190 \leq k \leq 210 \\ 1 & ,\ \text{en todo otro caso} \end{cases}$$

Resultado:

[4] Imágen extraída de un artículo de wikipedia: `http://commons.wikimedia.org/wiki/File:Etapas_filtrado_dominio_frecuencia.jpg`

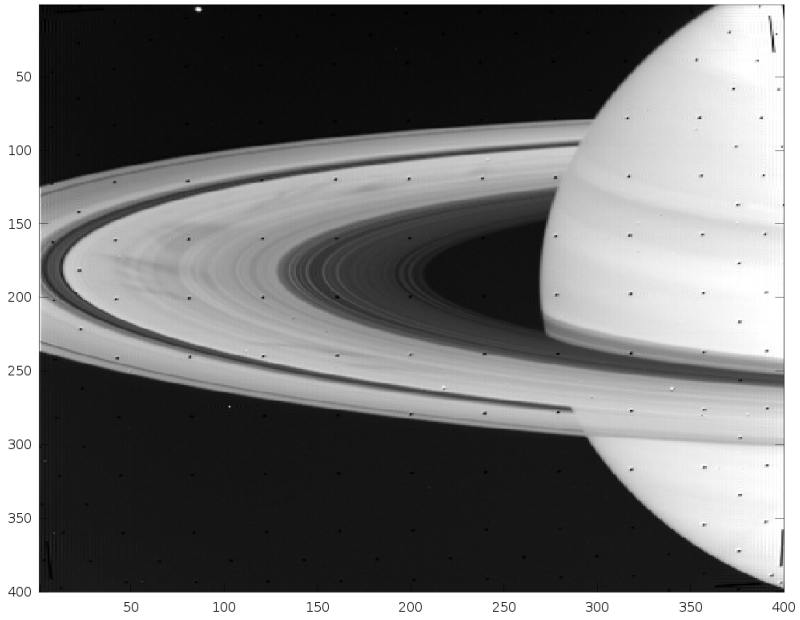

Figure 6: Imágen obtenida luego aplicado el filtro 1 sobre la matriz transformada

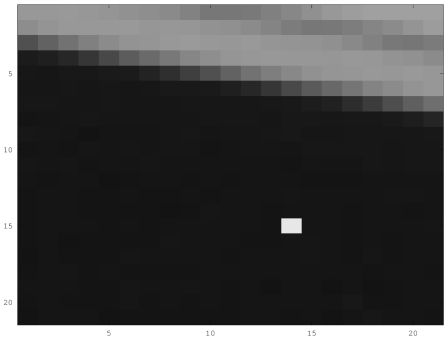

Figure 7: Zoom a un punto blanco en la imágen original.

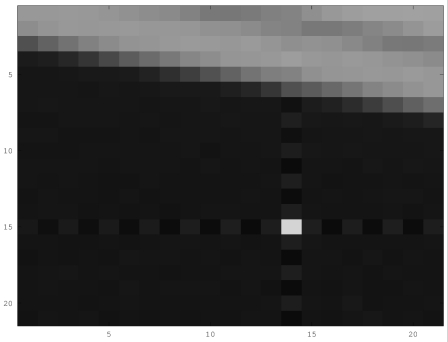

Figure 8: Zoom al mismo punto de la figura 7 luego aplicado el filtro1.

También conocido como "filtro notch". Este filtro no permite el paso de señales cuyas frecuencias se encuentran comprendidas entre las frecuencias de corte superior e inferior. Se utiliza para eliminar interferencia destructiva, encontrar o eliminar patrones en una imágen. En el caso presentado se puede ver (sobre todo en los puntos negros y blancos -ver figuras 7 y 8) que repite un patrón cruz a lo largo y ancho de la imágen. [1] [3]

2.4.2 Filtro Gaussiano

$$H_{k,l} = exp(-p * (k^2 + l^2))$$

Resultado:

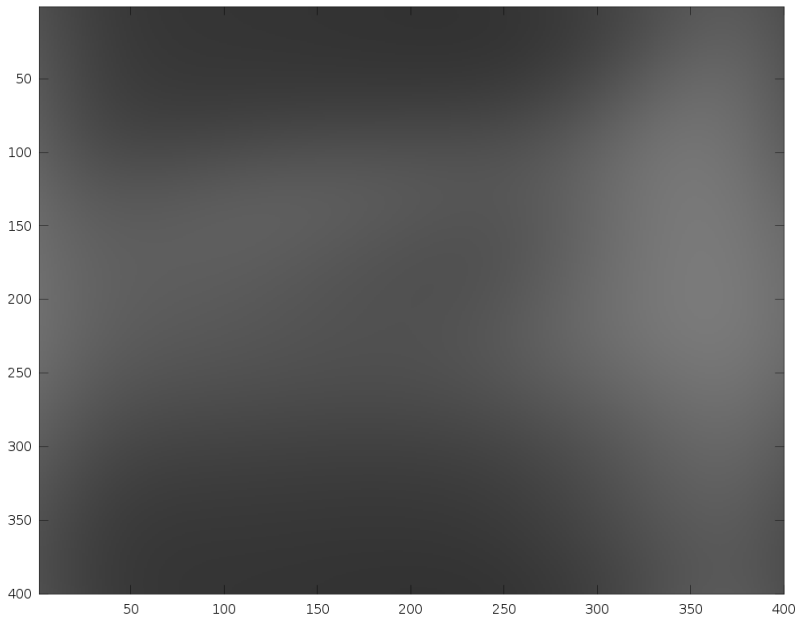

Figure 9: Imágen obtenida luego aplicado el filtro Gaussiano con $p = 0.1$ sobre la matriz transformada

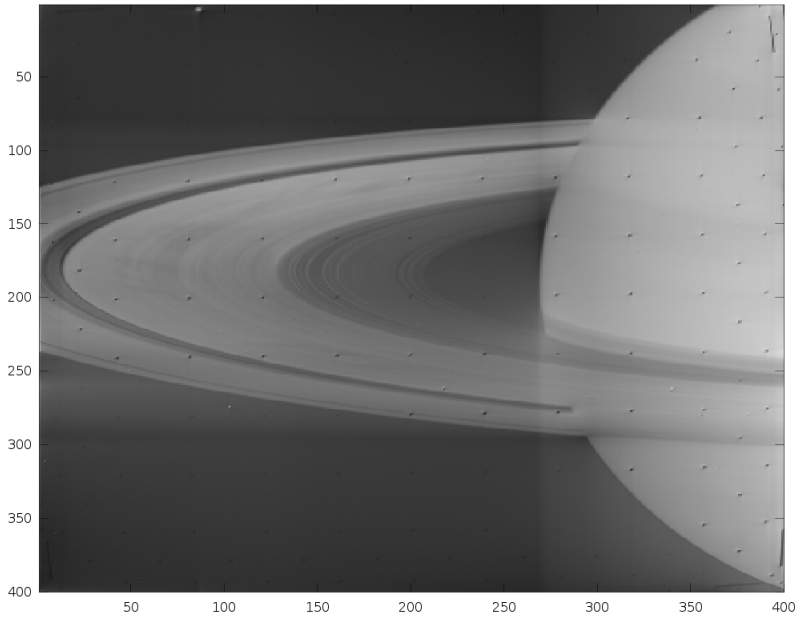

Figure 10: Imágen obtenida luego aplicado el filtro Gaussiano con $p = 0.00001$ sobre la matriz transformada

También conocido como "desenfoque gausiano". Este filtro mezcla ligeramente las intensidades de los píxeles que estén vecinos el uno al otro, en un mapa de bits (imágen), haciendo que la imágen pierda detalles, y de esta forma se suavizen bordes.

La figura 7 muestra los efectos de un desenfoque exagerado, y la figura 8 los de un pequeño desenfoque. Mientras que en la figuara 7 apenas se distingue la zona más clara de la imágen de la más oscura, en la figura 8 se distingue claramente la imágen original levemente desenfocada.

En ambos caos, puede notarse la desaparición de muchos de los puntos negros que se encontraban en la imagen, estas "marcas" que no son parte del objeto real, comunmente se lo conoce como ruido y complican la detección de bordes ya que terminan siendo detectados como objetos en la imagen, cuando en realidad no lo son.

El encontrar el valor adecuado de desenfoque para el tratamiento de la imagen en cuestión dependerá del objetivo que se busque y no es tarea sencilla de calcular. Borrar mucho ruido resulta en perdidas de detalles en la imagen, lo cual podría complicar en vez de mejorar la detección de bordes. Mientras que, borrar poco ruido, podría llevar a detección incorrecta de objetos. [4]

2.4.3 Filtro Damero

$$H_{k,l} = \begin{cases} 0 & l+k \text{ es par} \\ 1 & l+k \text{ es impar} \end{cases}$$

El filtro damero filtra los pixeles alternados, como en un tablero de ajedrez. Esto produce que se filtren (se apaguen) algunas frecuencias de manera salteada, causando un comportamiento altamente no lineal en el dominio

del tiempo.

Resultado:

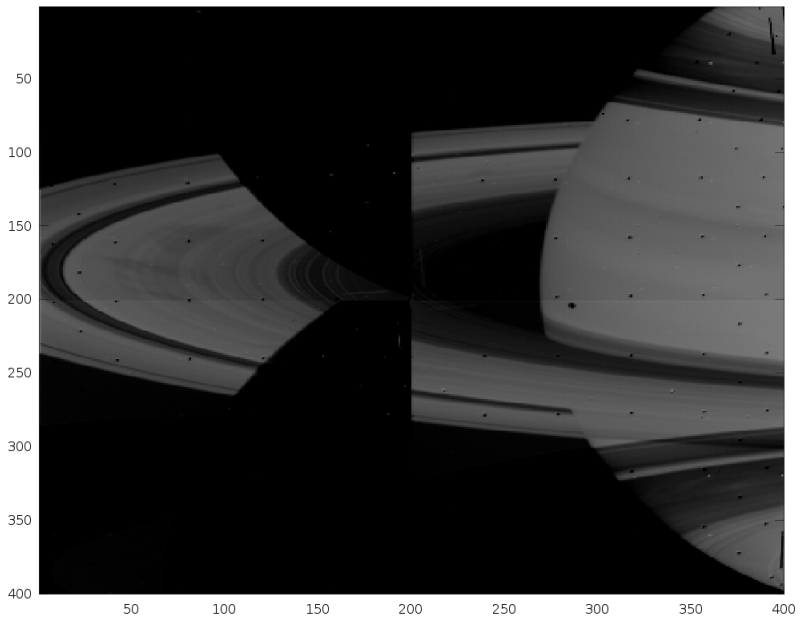

Figure 11: Imágen obtenida luego aplicado el filtro Damero sobre la matriz transformada

Este filtro parece haber intercambiado pedazos de la imágen, y haber puesto en esos mismos pedazos, los colores opuestos a los de la imágen original.

3 Resultados y Conclusiones

Luego de analizados los resultados obtenidos de la aplicación de filtros sobre imágenes en el espacio de frecuencias. Resulta muy notable la cantidad de áreas en donde esta técnica podría tener aplicaciones ya que permite simplificar notablemente el procesamiento de imágenes.

Se presetaron filtros para destruccion de interferencias por rango, reducción de ruido y suavizamiento de bordes, los cuales no requieren mas que una implementación de una funcion muy simple para la modificación de los valores de las frecuencias. Los cuales, si tuviesen que ser pensados para aplicarse sobre la imagen original, requerirían de un esfuerzo computacional muy superior.

Los filtros pueden tener variadas utilidades. Se pueden usar para hacer desde efectos sencillos a efectos complejos. Muchos son conocidos y se conocen muchas de sus utilidades, pero otros son nuevos y aún tienen utilidades desconocidas o incomprensibles.

El filtrado de imágenes resulta un tema complejo e interesante.

En cuanto a la implemenación del algoritmo de transformada de Fourier. Un problema que se encontró, es la notable cantidad de procesamiento requerido para la aplicación de los filtros para una imágen relativamente pequeña (encima en blanco y negro!). Es por eso que se recomienda el uso de la FFT (trasformada rápida de Fourier).

4 Bibliografía

[1] Frequency Filter - http://homepages.inf.ed.ac.uk/rbf/HIPR2/freqfilt.htm accedido en la fecha 26.06.2013

[2] Digital Filter - http://en.wikipedia.org/wiki/Digital_filter accedido en la fecha 26.06.2013

[3] Band-stop Filter - http://en.wikipedia.org/wiki/Band-stop_filter accedido en la fecha 26.06.2013

[4] Gaussian Blur - http://en.wikipedia.org/wiki/Gaussian_blur accedido en la fecha 26.06.2013

[5] A Recursive Implementation of the Fast Fourier Transform - http://beige.ucs.indiana.edu/B673/node13.html accedido en la fecha 27.06.2013

[6] Algorithm (FFT2) - http://www.originlab.com/www/helponline/Origin/en/UserGuide/Algorithm_(FFT2).html accedido en la fecha 27.06.2013

5 Anexo

5.1 Cálculo de transformada de Fourier discreta

```
function z = new_fft2(x,inverse)

    //Se calcula la transformada sobre las columnas de x.
    n = size(x,2);
    for u = 1:n
        y(:,u) = new_fft(x(:,u)',inverse)';
    endfor

    //Si es la transformada inversa hay que dividir por la cantidad de columnas.
    if(inverse)
        y = y ./n;
    endif

    //Se calcula la transformada sobre las filas de y.
    y = y';
    n = size(y,2);
    for v = 1:n
        z(:,v) = new_fft(y(:,v)',inverse)';
    endfor
    z = z';

    //Si es la transformada inversa hay que dividir por la cantidad de filas.
    if(inverse)
        z = z ./n;
    endif
endfunction


//Implementacion recursiva de la transformada rápida de fourier.
function y = new_fft(x,inverse)
    n = length(x);
    if(mod(n,2) != 0)
        //Caso base:Si no es divisible por 2 calculo la transformada normal.
        y = new_ft(x,inverse);
    else
        //Paso inductivo.
        m = n/2;
        //Llamada recursiva, dividiendo entre pares e impares.
        y_odd = new_fft(x(1:2:n),inverse);
        y_even = new_fft(x(2:2:n),inverse);

        if(inverse)
            d = exp(2 * pi * i / n) .^ (0:m-1);
```

```
        else
            d = exp(-2 * pi * i / n) .^ (0:m-1);
        endif
        z = d .* y_even;
        y = [ y_odd + z , y_odd - z ];
    endif
endfunction

//Implementación de la transformada de fourier
function y = new_ft(x,inverse)
    n = length(x);

    if(inverse)
        d = exp(2 * pi * i / n) .^ (0:n-1);
    else
        d = exp(-2 * pi * i / n) .^ (0:n-1);
    endif

    for k = (1:n)
        y(k) = sum(x .* (d.^(k-1)));
    endfor
endfunction
```

5.2 Shifteo de la *frecuencia* de una matriz, a fines de centrar la frecuencia 0 en el centro

```
//shifteado de las columnas.
Y = [Y(:,201:400) Y(:,1:200)]
//shifteado de las filas.
Y = [Y(201:400,:) ; Y(1:200,:)]
```

5.3 Código para aplicación de un filtro en dominio de frecuencia

```
//Se transformó la señal a la frecuencia.
Y = new_fft2(X,false);
//Se multiplicó la señal transformada por el filtro punto a punto.
Y2 = Y .* filtroxxx(400);
//Se destransformó la señal filtrada.
X2 = new_fft2(Y2,true);
//Se tomó la parte real de la señal filtrada.
Output = real(X2);
```