Procesamiento de Imágenes

Julián Gutiérrez (51141) — Pablo Pauli (51185) — Ivan Itzcovich (53891) — Gustavo Del Giudice (51289)

Resumen

Este trabajo tiene como objetivo aplicar las técnicas de la transformada discreta de Fourier para realizar filtrados espaciales sobre una imágen asignada. El filtrado espacial es una técnica comprendida en el campo del procesamiento digital, que se aplica sobre una imágen para resaltar o atenuar detalles espaciales con el objetivo de mejorar su interpretación visual.

Palabras clave. Procesamiento de imágenes, Fourier, FFT, Filtros.

1. Introducción

El procesamiento digital de imágenes es un conjunto de técnicas que tienen como objetivo alterar la naturaleza de una imágen para lograr una mejor interpretación visual de la misma o facilitar la busqueda de informacion. Estas técnicas buscan obtener una nueva imágen, cuyas características sean más adecuadas para una determinada aplicación que la imágen original. Dichos procedimientos pueden tener como fin suavizar la imágen, eliminar ruido, realizar bordes, detectar bordes, etc. A continuación, en la Figura 1, se puede apreciar la técnica para suavizar una imágen, es decir, reducir el cambio brusco de color en los pixel vecinos.

Figura 1: Filtrado promedio

Se entiende como filtro a las operaciones que se ejecutan sobre los pixels de la imágen digital para conseguir determinado efecto visual.

Los filtros se pueden realizar en el dominio de las frecuencias y/o en el dominio de las fases. En este trabajo, estos filtros se aplican al dominio de las frecuencias de la transformada discreta de Fourier de la imágen digital original. A continuaciónn, en la Figura 2, se pueden apreciar los pasos que constituyen el procesamiento de la imágen digital aplicando un filtro sobre el dominio de las frecuencias.

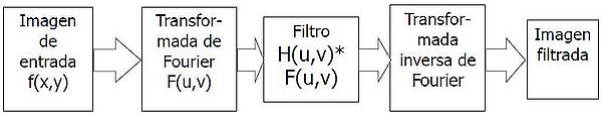

Figura 2: Etapas filtrado

El trabajo consta de dos partes básicamente: en una primera instancia se detalla la metodología implementada, es decir, los pasos que se llevaron a cabo para realizar las experiencias, con su respectivo respaldo teórico. Luego, se exponen los resultados y las conclusiones.

2. Desarrollo

2.1. Transformada Discreta de Fourier de una Secuencia Bidimensional

"*El trabajo se realiza en base a una imágen del planeta Saturno, capturada por la mision Voyager, sobre la cual se van a implementar las técnicas de la TDF para aplicar los filtros que correspondan. Esta imágen es la representación digital, en una escala de grises, de una matriz de 400x400 con valores enteros que van de 0 a 255.*"

Extracto del enunciado - Pablo fierens

Para computar la Transformada Discreta de Fourier de una secuencia bidimensional de NxN que caracteriza a la imágen, usualmente se implementa la ecuacion 1:

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

Esta operación es de $O(n^4)$, motivo por el cual resulta muy lento su cómputo en la práctica (notar que tiene que recorrer toda la matriz, tarea de orden n^2, y sobre cada uno otra tarea de orden n^2, lo que resulta en el orden n^4). Por esta razón, se optó por realizar la Transformada discreta de una manera alternativa, que consta de componer la Transformada Rápida de Fourier (FFT, según sus siglas en inglés), cuyos algoritmos reducen el tiempo de c omputo de la TDF. Esta estrategia es de tipo "divide and conquerza que fracciona las columnas en partes iguales y se invoca a si mismo en forma recursiva. Debido a esto, surge como restriccion que el tamanio de las columnas sea potencia de 2. Como la imágen empleada es de 400, se soluciono tomando un caso base de N=25, y sobre esos elementos

aplicar la FFT convencional. Finalmente el nuevo tiempo tiempo obtenido es de $O((nlog(n))^2)$, el cuadrado surge componer dos veces la funcion, para lograr la fft2 que era el objetivo.

2.2. Fidelidad de la implementación

Para asegurarnos de que la implementación que realizamos era fiel decidimos compararla con la nativa de Octave, Ejecutando por fuera de la simulación la rutina `checkCTFFT` donde se calcula el error cuadrático medio entre ambas, sobre la imágen a trabajar. El error obtenido fue de 4.68×10^{-20}, para los fines de la experiencia a realizar las consideramos "identicas".

2.3. Imágenes en amplitud y fase

Al aplicar la Transformada Discreta de Fourier sobre la imágen original, se obtiene una matriz, sobre la cual se pueden mapear los valores con el intervalo [0-255] y de esta manera encontrar su representación visual. En este ejercicio, primero se implementó la TDF en fase, y luego en amplitud, que consiste simplemente en aplicar el módulo a la transformada.

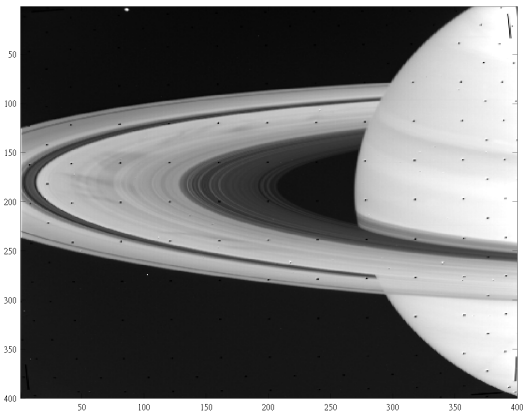

Figura 3: imágen original.

2.4. Aplicando Filtros

El filtro es una función que se aplica sobre el dominio de las frecuencias en la TDF de la imágen. Existen diversos tipos de filtros, según las características que se quieran imponer. Los filtros se pueden clasificar según la parte del espectro que dejan "pasar" y la que atenúan, existiendo así filtros de paso bajo (se atenúan solo los componentes de alta frecuencia, dejando los de baja frecuencia sin variaciones), de paso alto (se atenúan los componentes de baja frecuencia, pero no los de alta frecuencia) o intermedios.

Aplicar un filtro es simplemente capturar la respuesta de dicho filtro y multiplicarlo por los valores que retorna la TDF de una arreglo.

$$\begin{aligned} X &= F(x) \\ xfil &= F^{-1}(H * X) \end{aligned}$$

En el trabajo, se implementan 3 tipos de filtro sobre la TDF de la imágen original. Las ecuaciones 2 3 y 4 describen los fitros implementados:

- Filtro de unos

$$H_{k,l} = \begin{cases} 0 & \text{si } 0 \leq k \leq 400, 190 \leq l \leq 210 \\ 0 & \text{si } 190 \leq k \leq 210, 0 \leq l \leq 400 \\ 1 & \text{en otro caso} \end{cases} \tag{2}$$

- Filtro Gaussiano

$$H_{k,l} = e^{-0,01(k^2+l^2)} \tag{3}$$

- El damero

$$H_{k,l} = \begin{cases} 0 & \text{si } l+k \text{ es par} \\ 1 & \text{si } l+k \text{ es par} \end{cases} \tag{4}$$

3. Anti Transformando

A partir de los valores alterados por los filtros, se puede recuperar la señal aplicando la Transformada Inversa Discreta de Fourier, para poder obtener la imágen filtrada. La ecuación 5 que responde a este proceso es la siguiente:

$$x_{n,m} = \frac{1}{N^2} \sum_{l=0}^{N-1} \sum_{k=0}^{N-1} X_{l,k} e^{+i\frac{2\pi}{N}(nl+mk)} \tag{5}$$

En esta experiencia, primero se verifica la funcionalidad de la Anti Transformada sobre la TDF de la imágen original sin modificar por los filtros. En teoría, la imágen recuperada debe ser idéntica a la original. Luego, se aplica la misma operación, pero sobre los valores filtrados, para recuperar las imágenes filtradas y, observar los cambios y los efectos producidos por estos sobre la imágen original.

4. Resultados

Las rutinas están detalladas al final del artículo, en la sección anexo. Al correr el programa "simulation.m", se realizan todos los procesos que requiere la experiencia. Esta rutina invoca tanto las funciones de implementación propia que computan las transformadas y las antitransformadas, como las funciones que

alteran los arreglos matriciales con los filtros.

1) TDF de la imágen original

Se obtuvieron las siguientes imágenes al transformar la imágen original.

a) en fase (Figura 4)

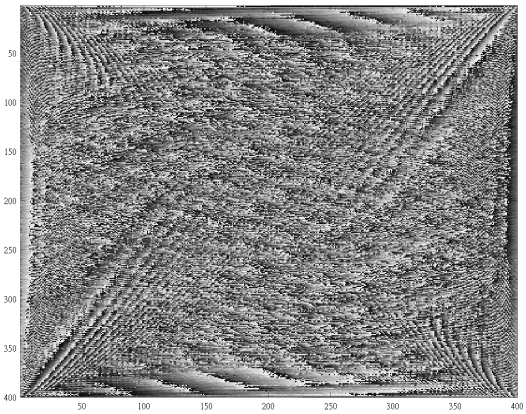

Figura 4: imágen de saturno en fase

b) en amplitud (Figura 5)

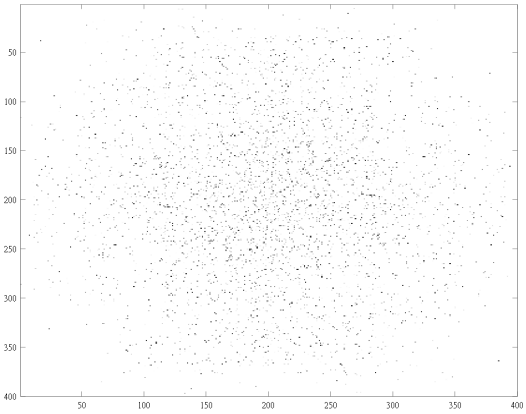

Figura 5: imágen de saturno en amplitud

2) Imágen reconstruida con la Transformada Inversa (Figura 6)

Al aplicar la transformada inversa sobre la matriz correspondiente a la imágen en fase del ítem anterior, se obtuvo la siguiente imágen. Notar que es idéntica a la original.

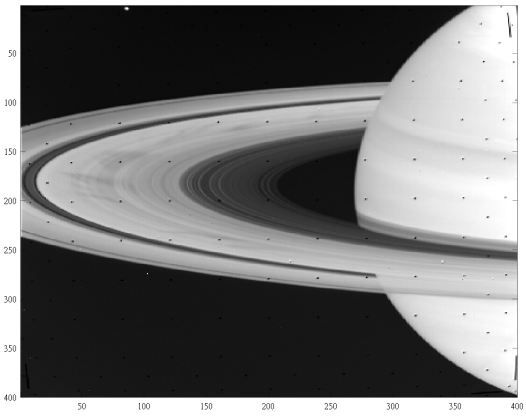

Figura 6: La imágen de saturno recuperada.

3) Imágen reconstruida con efecto de filtros
A continuación, se muestran las imágenes filtradas.
a) primer filtro (Figura 7)

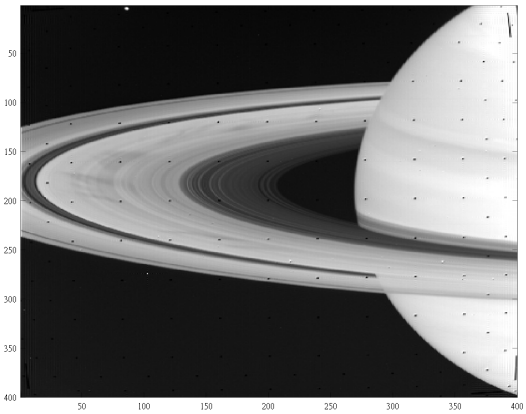

Figura 7: La imágen de saturno con el filtro 1 aplicado.

b) filtro Gaussiano (Figura 8)

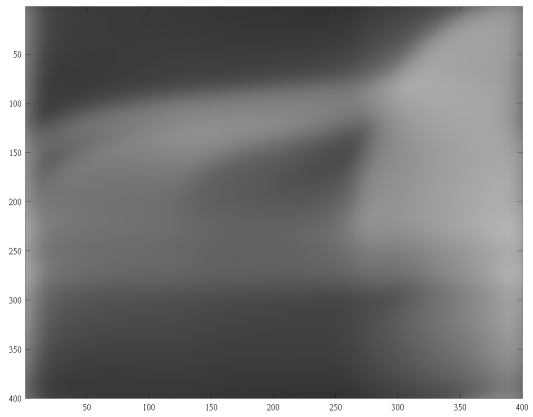

Figura 8: La imágen de saturno con el filtro Gaussiano aplicado.

c) Damero (Figura 9)

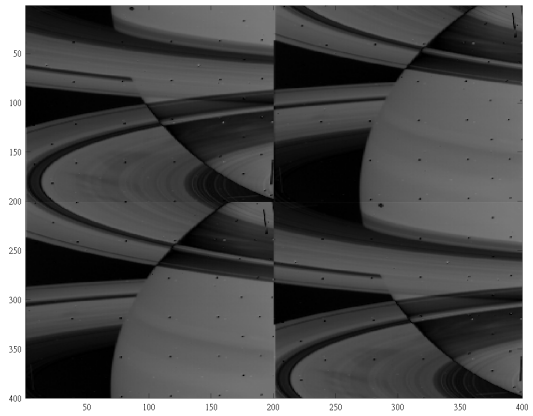

Figura 9: La imágen de saturno con el filtro Damero aplicado.

5. Conclusión

El primer filtro utilizado no alteró en demasía la imágen original, a diferencia de los otros. El filtro Gaussiano se caracteriza por perturbar la imágen, otorgándole un efecto "borroso" a la original, motivo por el cual los bordes son poco distinguibles y la imágen pierde precisión. Por último, el filtro damero provoca que la imágen se solape con si misma, resaltando una sensación de simetría, donde los bordes son notables y los colores más oscuros.

En conclusión, en este artículo se observa y verifica que los procesos que involucran la Transformada Discreta de Fourier y su Anti Transformadas son óptimos para el procesamiento digital que corresponde al estudio de filtros aplicados a imágenes. Esta disciplina es útil y eficaz cuando se quiere resaltar ciertas características visuales de una determinada imágen digital, para poder facilitar su interpretación y la búsqueda de información sobre la misma.

6. Bibliografía

- Fierens, Pablo. *Guía 2*, Métodos Numéricos Avanzados. ITBA, 1er Cuatrimestre 2014
- Wikipedia *Procesamiento digital de imágenes* URL: `http://es.wikipedia.org/wiki/Procesamiento_digital_de_im\%C3\%A1genes`
- R. Fisher, S. Perkins, A. Walker and E. Wolfart. *Fourier Transform* URL: `http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm`
- DEPARTAMENTO DE ELECTRÓNICA. U.A.H. *PROCESADO DE IMÁGENES EN EL DOMINIO DE LA FRECUENCIA.* URL: `http://www.depeca.uah.es/depeca/repositorio/asignaturas/1/t5-procesado.pdf`

7. Anexo

Aquí se pueden ver las funciones de *GNU Octave* utilizadas para este análisis.

El *script* `ctfft.m` describe la Transformada Rapida de Fourier(TRF), en el caso de una matriz se aplica sobre cada columna de la misma. Se utiliza el algoritmo de Cooley Tuckey ya mencionado.

```
ctfft.m


function X = ctfft(x)
X = core(x,-1);
endfunction
```

El *script* `ictfft.m` describe la Inversa de la TRF.

```
ictfft.m


function X = ictfft(x)
X = core(x,1);
endfunction
```

El *script* `ctfft2.m` ctfft2 describe la Transformada Discreta de Fourier(TDF) para una secuencia binomial.

```
ctfft2.m


%
%cooley tukey fft2
%known fact:
%fft2(x)-fft(fft(x).').'== 0;
%
function X = ctfft2(x)
    X = ctfft(ctfft(x).').';
endfunction
```

El *script* `ictfft2.m` describe la Inversa de la TDF, tambien para una secuencia binomial.

```
ictfft2.m


function X = ictfft2(x)
    X = ictfft(ictfft(x).').';
endfunction
```

El *script* `core.m` representa el "núcleo"que calcula tanto las transformadas como las antitransformadas mediante el algoritmo descrito. Se separo de esta manera para poder reutilizar la rutina sin repetir instrucciones.

core.m

```
%cooley tukey fast fourier transformed/inverse
%in the case of a matrix performs fft over each column
%if dimentions are not power of two a proper amount of zeroes will be used
%to fill up to the next power and trim on the output, so the user won't notice.
%
%inverse is 1 when calculating the inverse. -1 otherwise
%
function X = core(x , inv_)
    [n m] = size(x);
    X = zeros(n,m);
    for k = 1:m
        X(:,k)= ctfftCore(x(:,k),n,inv_);
    endfor
    if(inv_ == 1)
        X = X/n;
    endif
endfunction

function X =  ctfftCore(x,N,inv_)
    if(N == 25)
        %X(1)=x(1);
        col = exp(inv_*2*pi*i* (0:N-1)/N).';
        V = vander(col,N);
        X = x.'*V;
        X = fliplr(X);
    else
        X(1:N/2) = ctfftCore(x(1:2:N),N/2,inv_);
        X(N/2+1:N) = ctfftCore(x(2:2:N),N/2,inv_);
        k = 1: N/2;
        t = X(k);
        X(k) = t + exp(inv_*2*pi*i* (k-1)/N).*X(k+N/2);
        X(k+N/2) = t - exp(inv_*2*pi*i* (k-1)/N).*X(k+N/2);
    endif
endfunction
```

El *script* `checkerboardFilter.m` implementa retorna la matriz que representa el filtro "damero"para ser aplicado sobre una imágen.

```
checkerboardFilter.m


%
% H will have the shape of a checkerboard, black and white 'squares'.
%
function H = checkerboardFilter()
    H = zeros(400, 400);
    for i=1:400
        for j=1:400
            H(i,j) = mod(i+j, 2);
        endfor
    endfor

endfunction
```

El *script* `gaussianFilter.m` implementa retorna la matriz que representa el filtro "gaussiano"para ser aplicado sobre una imágen.

```
gaussianFilter.m


function H = gaussianFilter()
    H = zeros(400, 400);
    for i=1:400
        for j=1:400
            H(i,j) = exp(-0.01*(i^2 + j^2));
        endfor
    endfor
endfunction
```

El *script* `onesFilter.m` implementa retorna la matriz que representa el filtro ünos"para ser aplicado sobre una imágen.

```
onesFilter.m


function H = onesFilter()
    H = ones(400, 400);
    for i=1:400
        for j=190:210
            H(i,j) = 0;
        endfor
    endfor

    for i=190:210
        for j=1:400
            H(i,j) = 0;
        endfor
    endfor

endfunction
```

El *script* `loadData.m` implementa la carga inicial de la imágen desde el archivo provisto.

loadData.m

```
function [image map] = loadData(file)
map = colormap(gray(255));
image = load(file);
endfunction
```

El *script* `checkCTFFT.m` compara nuestra implementación de la TDF contra la de octave, tomando la imágen a utilizar y devuelve el error cuadrático medio entre ambas.

checkCTFFT.m

```
%Compares ctfft2 with fft2 from octave
%and returns error
function e = checkCTFFT()
x = loadData('../data/saturn');
A = fft2(x);
B = ctfft2(x);
e = meanError(A,B);
endfunction
```

El *script* `meanError.m` caclula el error cuadratico medio entre la imágen original y la recuperada.

meanError.m

```
%Calculates mean square error between two grayscale
%images (in the form of matrixes) of the same size
function e = meanError(A,B)
[n m] = size(A);
e = abs(sum(sum((double(A) - double(B)) .^ 2))/(n*m));
endfunction
```

El *script* `simulation.m` junto a sus subrutinas describen la tarea realizada mostrando las imágenes que se obtienen durante el proceso, ademas de guardarlas. Adicionalmente muestra por consola el error cometido al recuperar la imágen.

simulation.m

```
%Compute the whole simulation saving and showing images
%also display error on recovery.
function simulation()
addpath('./filters');
x = loadData('../data/saturn');
colormap(gray(255));

%Display original image
doi(x);

%Display image transformed into phase
tic
X = ctfft2(x);
toc
ditip(X);

%Display image transformed into amplitude
ditia(X);

%Display recovered original image
originalImage = x';
recoveredImage = droi(X);
printf('Error while recovering image: %.22f\n',
meanError(originalImage,recoveredImage));

%Apply first filter
aff(X);

%Apply second filter (Gaussian)
asf(X);

%Apply third filter (Checkerboard)
atf(X);
end

function doi(x)
image(x');
print("-dpng", "../img/saturn.png");
endfunction

function ditip(X)
printf('Transforming image into phase... ');
fflush(stdout);
phase = angle(X);
mx = max(max(phase));
mn = min(min(phase));
m = 255/(mx-mn);
b = -m*mn;
```

```
nphase = floor(m*phase + b);
image(nphase');
print("-dpng", "../img/saturn_phase.png");
printf('done.\n');
fflush(stdout);
endfunction

function ditia(X)
printf('Transforming image into amplitude... ');
fflush(stdout);
image(abs(X)');
print("-dpng", "../img/saturn_amplitude.png");
printf('done.\n');
fflush(stdout);
endfunction

function recoveredImage = droi(X)
printf('Recovering original image... ');
fflush(stdout);
xprima = ictfft2(X);
recoveredImage = round((real(xprima)).');
image(recoveredImage);
print("-dpng", "../img/saturn_recovered.png");
printf('done.\n');
fflush(stdout);
endfunction

function aff(X)
printf('Applying first filter... ');
fflush(stdout);
xf1_function = onesFilter().*X;
xf1_result = abs(ictfft2(xf1_function))';
image(xf1_result);
print("-dpng", "../img/saturn_filter1.png");
printf('done.\n');
fflush(stdout);
endfunction

function asf(X)
printf('Applying gaussian filter... ');
fflush(stdout);
xf2_function = gaussianFilter().*X;
xf2_result = abs(ictfft2(xf2_function))';
image(xf2_result);
print("-dpng", "../img/saturn_filter2.png");
printf('done.\n');
fflush(stdout);
endfunction

function atf(X)
printf('Applying checkeredboard filter... ');
fflush(stdout);
xf3_function = checkerboardFilter().*X;
xf3_result = abs(ictfft2(xf3_function))';
image(xf3_result);
print("-dpng", "../img/saturn_filter3.png");
printf('done.\n');
fflush(stdout);
endfunction
```