FICH, UNL - Departamento de Informática - Ingeniería Informática

**Procesamiento Digital**
**de Imágenes**

# Procesamiento y filtrado frecuencial con CImg

## Resumen de funciones y sugerencias para la resolución de la guía de trabajos prácticos

2012

## 1. Introducción

Este resumen tiene como objetivo sintetizar las funciones provistas por la librería CImg[1] para realizar el cálculo de la transformada discreta de Fourier (TDF) directa e inversa. Al mismo tiempo se provee al alumno con sugerencias y ejemplos de código que le ayuden a realizar una correcta y adecuada implementación de los ejercicios de procesamiento y filtrado frecuencial planteados en la guía de trabajos prácticos.

## 2. Cómo configurar CImg para realizar el cálculo de la TDF

La librería CImg usa por defecto una implementación de rutinas propias para el cálculo de la TDF directa e inversa. Sin embargo, esta opción predeterminada tiene la limitación de funcionar exclusivamente para imágenes cuyo tamaño sea potencia de 2.

Para transformar imágenes de cualquier tamaño CImg invoca las funciones definidas en la librería `FFTW`[2], para lo cual el sistema operativo que está siendo utilizado debe tener instalados los paquetes `fftw3` y `fftw3-dev`. Versiones de estas librerías se encuentran disponibles para distintos sistemas operativos.

A continuación es necesario configurar CImg para que utilice por defecto `fftw3` en el cálculo de la TDF. Para ello, en la parte de definiciones del archivo `CImg.h` debe especificarse la siguiente línea:

```
#define cimg_use_fftw3 1
```

antes de la secuencia:

```
#ifdef cimg_use_fftw3
  extern ''C''{
```

```
    #include ‘‘fftw3.h’’
  }
#endif
```

Por último, al compilar se debe incluir el parámetro `-lfftw3` para realizar el *linkeo* correspondiente.

## 3. Funciones para el cálculo de la TDF

Los métodos para el cálculo de la TDF son los siguientes:

- `CImgList&get_FFT(const char eje, const bool inversa=false)`

y su versión *in-place* (sobreescribe el valor original del objeto CImgList):

- `CImgList&FFT(const char eje, const bool inversa=false)`

Los parámetros anteriores permiten setear el eje a lo largo del cual calcular la TDF (vertical, horizontal o ambos-*default*-) y si se trata de la transformada directa (`inversa=false`) o inversa (`inversa=true`). El valor por defecto es `false`.

El método permite obtener un objeto de tipo `CImgList` formado por dos elementos. El primer elemento de la lista corresponde a la **parte real** de la TDF y el segundo a la **parte imaginaria**. El ejemplo 1 muestra cómo pueden calcularse las TDF directa e inversa para una imagen genérica de cualquier tamaño.

### Ejemplo 1

```
CImg<double>img(‘‘imagen.jpg’’);
//Transformada directa y obtención de las partes real e imaginaria
CImgList<>TDF_img=img.get_FFT();
CImg<double>TDF_real=TDF_img[0];
CImg<double>TDF_imaginaria=TDF_img[1];
...
...
CImgList<>TDF_nueva(TDF_img);
TDF_nueva[0]=TDF_real;
TDF_nueva[1]=TDF_imaginaria;
//Transformada inversa y obtención de la parte real
CImg<double>img_nueva=TDF_nueva.get_FFT(true)[0];
```

## 4. Cálculo de magnitud y fase y notación en coordenadas polares

Debido a que los métodos `CImgList&get_FFT()` y `CImgList&FFT()` devuelven las partes real e imaginaria de la TDF, el cálculo de la magnitud y fase se puede implementar mediante la obtención del módulo y argumento del número complejo formado por la combinación de la parte real e imaginaria para cada píxel, como se muestra en el ejemplo 2.

## Ejemplo 2

```
#include <complex>
...
//Definición de la constante imaginaria I=sqrt(-1):
complex<double>I(0.,1.);
CImg<double>magnitud(img.width(),img.height(),1,1);
CImg<double>fase(img.width(),img.height(),1,1);
for (int i=0;i<img.width();i++){
     for (int j=0;j<img.height();j++){
       magnitud(i,j)=sqrt(pow(TDF_real(i,j),2)+pow(TDF_imaginaria(i,j),2));
       //Expresión de cada valor complejo de la TDF a partir de
       //su parte real e imaginaria:
       complex<double>complejo=TDF_real(i,j)+I*TDF_imaginaria(i,j);
       //La función ''arg'' devuelve
       //el argumento (ángulo) del número complejo:
       fase(i,j)=arg(complejo);
     }
}
...
```

La expresión de la TDF en coordenadas polares (en función de la magnitud y fase calculadas anteriormente):

$$F(u,v) = |F(u,v)|e^{j\theta(u,v)}$$

y luego la obtención de la transformada inversa, se puede realizar como muestra el ejemplo 3.

## Ejemplo 3

```
#include <complex>
...
//Definición de la constante imaginaria I=sqrt(-1):
complex<double>I(0.,1.);
CImg<double>real(img.width(),img.height(),1,1);
CImg<double>imaginaria(img.width(),img.height(),1,1);
for (int i=0;i<img.width();i++){
     for (int j=0;j<img.height();j++){
       //Obtención de las partes real e imaginaria
       //de la TDF expresada en coordenadas polares.
       //Las funciones ''real'' e ''imag'' se encuentran
       //definidas en la librería estándar ''complex.h'':
       real(i,j)=real(magnitud(i,j)*exp(I*fase(i,j)));
       imaginaria(i,j)=imag(magnitud(i,j)*exp(I*fase(i,j)));
     }
}
...
```

Las partes real e imaginaria calculadas en el ejemplo anterior se pueden utilizar ahora para obtener la TDF inversa y volver así al dominio espacial.

## 5. Funciones de visualización

En el *header* de funciones ‘‘`PDI_functions.h`’’ se encuentran definidas dos funciones que pueden ser de utilidad a la hora de visualizar la apariencia de la magnitud de la TDF y de las respuestas en frecuencia de filtros.

Estas funciones son las siguientes:

- `void magn_tdf(CImg<double>img, bool centrada, const char* palname)`: Calcula y devuelve en el objeto `magnitud_tdf` el módulo de la TDF de la imagen `img` transformada logarítmicamente para comprimir el rango dinámico de intensidades. Si la opción `centrada` es `true`, la magnitud se encuentra centrada, de lo contrario se encuentra descentrada.
  **Advertencia!** Esta función fue implementada exclusivamente a los fines de facilitar la visualización de la magnitud de la TDF como imagen 2D. Nunca se debe utilizar cuando sea necesario manipular o modificar componentes de la TDF, como por ejemplo en el caso de filtrado.

- `void draw_3D_image(CImg<unsigned char>mag_filtro)`: Si bien la respuesta en frecuencia de los filtros se puede observar como imagen 2D, esta función permite visualizarla en un entorno interactivo 3D. Muestra en consola las opciones de teclado y mouse disponibles para rotar el punto de vista, cambiar la textura de visualización, etc. El código es una adaptación del provisto por CImg.
  **Advertencia!** En el sistema de coordenadas visible en el ángulo inferior izquierdo, el eje $z$ aparece invertido.

## 6. Centrado y descentrado de transformadas

Para centrar o descentrar el módulo de la TDF se pueden utilizar los siguientes métodos, utilizando el parámetro `condicion_borde=2` (repite el patrón del borde opuesto, adecuado debido a la periodicidad de la TDF):

- `CImg&get_shift(const int deltax, const int deltay = 0, const int deltaz = 0, const int deltav = 0, const int condicion_borde = 0)`

y su versión *in-place*:

- `CImg&shift(const int deltax, const int deltay = 0, const int deltaz = 0, const int deltav = 0, const int condicion_borde = 0)`

donde los parámetros `deltax` y `deltay` deben adoptar los valores `(int)CImg&width()/2` y `(int)CImg&height()/2`, respectivamente, mientras que los parámetros `deltaz` y `deltav` deben ser `0`.

## 7. Relleno con ceros

La necesidad de rellenar con ceros (*padding*) al filtrar en frecuencia se encuentra detallada en la página 199 del libro de Gonzalez y Woods [3]. El ejercicio 6 de la guía de trabajos prácticos a la cual corresponde esta síntesis propone demostrar de manera práctica esta necesidad.

El relleno con ceros de una imagen de tamaño $M \times N$ a un tamaño especificado $P \times Q$ se puede realizar en CImg mediante los métodos:

- `CImg&get_resize(const int pdx = -100, const int pdy = -100, const int pdz = -100, const int pdv = -100, const int interp = 1)`

y su versión *in-place*:

- `CImg&resize(const int pdx = -100, const int pdy = -100, const int pdz = -100, const int pdv = -100, const int interp = 1)`

En estos métodos, los valores de `pdx` y `pdy` (nuevo tamaño de la imagen) deben setearse con los valores de $P$ y $Q$ respectivamente, los valores de `pdz` y `pdv` en `-100` (no se alteran estas dimensiones), y el valor de `interp` en `0` (a la imagen la deja del mismo tamaño y completa con ceros hasta llegar al tamaño $P \times Q$).

## 8. Definición del espaciamiento de frecuencias normalizado para generar respuestas en frecuencia de los filtros

Es de utilidad definir frecuencias equiespaciadas sobre el cual definir la magnitud de la respuesta en frecuencia de los diferentes filtros. El ejemplo 4 muestra cómo se puede definir un espacio de frecuencias normalizado en los rangos $[-1, 1]$ y $[-1, 1]$ para cada una de las variables frecuenciales $u$ y $v$, siendo $M \times N$ el tamaño de la imagen a filtrar.

**Ejemplo 4**

```
int M=img.width();
int N=img.height();
CImg<double>U(M,N,1,1),V(M,N,1,1);
CImg<double>D(M,N,1,1);
   for (int i=0;i<M;i++){
      for (int j=0;j<N;j++{
         //Variables de frecuencia U y V:
         U(i,j)=-1.+i*2./(M-1);
         V(i,j)=-1.+j*2./(N-1);
         D(i,j)=sqrt(pow(U(i,j),2)+pow(V(i,j),2));
         //Definicion de la magn. de rta. en frec. de H en función de D:
         H(i,j)=Filtro Butterworth, Gaussiano, etc...
         ...
      }
}
```

## 9. Medición de tiempos de ejecución para comparación de métodos

Algunos de los ejercicios planteados en la guía de trabajos prácticos requieren el cálculo del tiempo de ejecución de algunos métodos o algoritmos. Una forma simple de realizar esto es utilizando la librería estándar ‘‘time.h’’ tal como se muestra en el ejemplo 5:

### Ejemplo 5

```
#include <time.h>
...
//inicio y fin son del tipo time_t definido en ‘‘time.h’’
time_t inicio, fin;
struct tm *tpoInicio, *tpoFin;
//Medir el tiempo desde aquí...:
inicio=time(NULL);
...
//... hasta aquí:
fin=time(NULL);
cout<<‘‘Tiempo transcurrido (seg.):’’<<difftime(fin,inicio)<<endl;
...
```

## Referencias


[1] Tschumperlé, D.“The CImg Library. C++ Template Image Processing Toolkit”. Web: http:cimg.sourceforge.net

[2] Frigo M. and Johnson S. G. “The Design and Implementation of FFTW3”. In Proceedings of the IEEE 93 (2), 216-231 (2005). Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation. Web: http:www.fftw.org

[3] Gonzalez, R. C. and Woods, R. E. “Digital Image Processing”, 2nd. Edition, Prentice Hall, 2002.