| Práctica 6: Operadores globales | | Grupo | |
|---|---|---|---|
| | | Puesto | |
| **Apellidos, nombre** | **Sánchez Garzón, Laura** | **Fecha** | |
| **Apellidos, nombre** | **Remuiñán Cid, Sara** | | |

El objetivo de esta práctica es presentar al alumno los fundamentos de los operadores globales y parte de sus aplicaciones.

Desarrolle cada ejercicio en un fichero de comandos 'ejercicio_X.m' separado. Para conocer el funcionamiento preciso de los comandos que se introducen en este guion, utilice la ayuda de MATLAB. Para evitar posibles interferencias con otras variables o ventanas recuerde incluir siempre las instrucciones `clear all` y `close all` al principio de cada fichero de comandos.

Al finalizar la práctica, comprima el documento de observaciones y los ficheros '.m' generados en un único fichero con el nombre '**FTDI_P6_ApellidosNombre1_ApellidosNombre2.zip**', conéctese al sistema de entrega de prácticas de *Moodle* y entréguelo.

**NOTA IMPORTANTE**: En el desarrollo de esta práctica y posteriores se pide realizar operaciones repetidamente, por ejemplo, asignando individualmente píxeles concretos de una imagen. En estos casos se recomienda acudir al uso de estructuras de control (bucles, condiciones, etc.). Adicionalmente, debido a la creciente complejidad de los programas que se le va a pedir desarrollar, se recomienda encarecidamente el uso del depurador o *debugger* de MatLab, cuya funcionalidad está descrita en la ayuda del programa (*User's Guide - Desktop Tools and Development Environment - Editing and Debugging MATLAB Code*).

# 1 Diseño frecuencial de máscaras

Como han visto en teoría, el diseño de filtros en frecuencia puede servir como banco de pruebas para el diseño de filtros espaciales. Asimismo, en teoría se desarrolló una metodología para este proceso que aquí incluimos y desarrollamos por medio de un ejemplo.

**Metodología:**

I. *Generación del filtro frecuencial con los parámetros deseados.*

II. *Obtención de su respuesta impulsiva.*

III. *Selección de rango –anchura de la máscara- y obtención de valores enteros en el rango deseado cuyo error de redondeo de lugar a mínimo error cuadrático medio respecto a los valores racionales originales.*

IV. *Ajuste de la respuesta de la máscara a una señal constante.*

***Ejemplo: Diseño a partir de filtros ideales cuadrados centrados***

I. *Generación del filtro frecuencial con los parámetros deseados.*

Genere un filtro paso bajo ideal en la región $0 \leq x \leq 1,\ 0 \leq y \leq 1$ con las siguientes características:

a. Frecuencia de muestreo en el retículo ortogonal: *fs* = 400 .

```
fs=400;
```

b. Retículo ortogonal de vectores: $\vec{v}_1 = (1/fs\,, 0), \vec{v}_2 = (0{,}1/fs)$

**Divida el retículo en** *fs* +1 = 401 **muestras, es decir, conserve la última muestra de manera que el número de muestras sea impar en ambas dimensiones.**

```
v1=1/fs; v2=1/fs;
x=[0:v1:1];
y=[0:v2:1];
[X,Y]=meshgrid(x,y);
```

c. Posición central del filtro (en el centro de la imagen): $f_{0x} = 0.5, f_{0y} = 0.5$

```
f0x=0.5;
f0y=0.5;
Xc = (X - f0x);Yc = (Y - f0y);
```

d. Frecuencia de corte (horizontal y vertical): $D_0 = fs/4$

```
D0=fs/4;
% filtro paso bajo ideal con frecuencia de corte D0
f_filter =double(abs(Xc) <= D0/fs & abs(Yc) <= D0/fs );
```

II. *Obtención de su respuesta impulsiva*.

Calcule la respuesta al impulso del filtro paso bajo ideal definido Para ello recuerde mover primero el centro del filtro (frecuencia cero) al origen de la matriz y deshacer esta operación en la transformada inversa de Fourier obtenida mediante la función `ifft2`. Puede utilizar para ello las funciones `ifftshift` y `fftshift` de Matlab, es decir, si denominamos al filtro en frecuencia `f_filter`:

```
f_filter_d = ifftshift(f_filter);
h_d = real(ifft2(f_filter_d));
h = fftshift(h_d);
```

Compruebe que la respuesta al impulso obtenida *h*[*x*, *y*] es real en todo su rango de definición.

III. *Selección de rango –anchura de la máscara- y obtención de valores enteros en el rango deseado cuyo error de redondeo de lugar a mínimo error cuadrático medio respecto a los valores racionales originales.*

Primero localice la posición del centro (*cx*, *cy*) del filtro espacial *h*[*x*, *y*] . Note que, dada la definición del filtro en frecuencia, este valor puede obtenerse como:

$$c_x = 1 + \frac{fs}{2} \qquad c_y = 1 + \frac{fs}{2}$$

A continuación, defina la anchura del filtro `w`. Por ejemplo, para obtener una máscara de filtrado *3x3*, fije `w = 1` y obtenga la máscara de filtrado recortando la respuesta a partir de la expresión:

```
filter_mask=h(cy-w:cy+w,cx-w:cx+w);
```

A continuación, obtenga la versión en número enteros de la máscara obtenida.

Para ello:

a. Obtenga primero el mínimo valor en `filter_mask` y divida la máscara por ese valor, así el menor valor será uno.

```
      [m,j] = min(filter_mask(:));
filter_mask = filter_mask./m;
```

b. Defina el máximo factor de ponderación utilizable. Para ello, note que el rango de la máscara tras este proceso deberá estará comprendido entre los 256 niveles: $[-128,128)$ permitiendo que la máscara contenga valores negativos.

   Así, el máximo factor de ponderación utilizable, `R`, podrá ser aproximado mediante la expresión:

```
R = ceil(127./max(abs(filter_mask(:))));
```

c. Recorra mediante un bucle los enteros comprendidos entre 1 y `R`. Calcule la versión en números enteros de `filter_mask` resultante de aplicar la ponderación correspondiente en cada iteración del bucle y mida el error cuadrático medio respecto a la máscara original:

```
% búsqueda de la versión entera que minimiza el MSE
mse = Inf.*ones(1,R);
for j =1:R,
    dif = (filter_mask - double(round(filter_mask.*j)./j)).^2;
    mse(j) = mean(dif(:));
end
[~,j]=find(mse==min(mse),1);
filter_mask = round(filter_mask.*j);
```

IV. *Ajuste de la respuesta de la máscara a una señal constante.*

La obtención del factor de ponderación: *C* será determinante en el uso de la máscara diseñada. El resultado de la operación de filtrado habrá de ponderarse por este factor. Consulte con el temario de teoría el método para calcular *C* una vez que `filter_mask` ha sido obtenida.

```
filter_mask=filter_mask/C;
```

## *1.1 Ejercicio 1: Diseño a partir de filtros ideales centrados*

Siga las instrucciones para el diseño de máscaras de filtrado en frecuencia: "***Ejemplo: Diseño a partir de filtros ideales cuadrados centrados***"

## *1.2 Ejercicio 2: Diseño a partir de filtros ideales circulares centrados*

Siga las instrucciones para el diseño de máscaras de filtrado en frecuencia.

Construya un **filtro paso bajo circular ideal** en la región $0 \leq x \leq 1,\ 0 \leq y \leq 1$ con las siguientes características:

a. Frecuencia de muestreo en el retículo ortogonal: fs = 400 .
b. Retículo ortogonal de vectores: $\vec{v}_1 = (1/fs\,, 0), \vec{v}_2 = (0, 1/fs)$
c. Posición central del filtro (en el centro de la imagen): $f_{0x} = 0.5, f_{0y} = 0.5$
d. Frecuencia de corte (horizontal y vertical): $D_0 = fs/4$

Para la definición del filtro, utilice la ecuación del círculo, $(x^2 + y^2) \leq r^2$ , y $D_0/fs$ como radio.

**Medida del error del diseño**

Obtenga el error del filtro espacial obtenido respecto del filtro frecuencial deseado. Para ello parta de la respuesta al impulso obtenida en el paso III y sustituya los valores correspondientes al área del filtro espacial por los obtenidos tras el proceso de redondeo a entero. Anule el resto de los valores fuera de una ventana de tamaño `w` situada en el centro del filtro definido por `cx` y `cy`:

```
C=sum(sum(filter_mask));
h_f = zeros(size(h));
h_f(cy-w:cy+w,cx-w:cx+w) = filter_mask./C;
```

Obtenga la transformada inversa de Fourier de esta respuesta al impulso truncado, para ello recuerde realizar las operaciones de desplazado opuestas a las del paso II.

```
h_f_d = fftshift(h_f);
f_filter_t_d = fft2(h_f_d);
f_filter_t = abs(ifftshift(f_filter_t_d));% sólo el módulo
```

Realice un estiramiento (operador puntual de ampliación de contraste) de `f_filter_t` al rango del filtro `f_filter [0,1].` Puede seguir el procedimiento de las prácticas anteriores.

Represente en un gráfico de 3 columnas: el filtro diseñado en frecuencia (`f_filter`), el módulo del filtro en frecuencia aproximado (`f_filter_t`) y la diferencia al cuadrado entre ambos. Calcule también el error cuadrático medio (MSE) entre `f_filter` y `f_filter_t`.

Para ello utilice los comandos `subplot, imagesc` y `colormap(gray).`

Repita el ejercicio para varios valores de la anchura del filtro `w`, ¿qué observa? ¿Qué problemas puede acarrear el incremento de `w`?

## *1.3 Ejercicio 3: Diseño a partir de filtros paso bajo Gaussiano*

Repita el **ejercicio 2** analizando el efecto de `w` en el filtro espacial estimado, pero modificando el tipo de filtro en frecuencia.

Construya un filtro **paso bajo Gaussiano** en la región $0 \leq x \leq 1,\ 0 \leq y \leq 1$ como sigue:

a. Frecuencia de muestreo en el retículo ortogonal: fs = 400 .
b. Retículo ortogonal de vectores: $\vec{v}_1 = (1/fs, 0), \vec{v}_2 = (0, 1/fs)$
c. Posición central del filtro (en el centro de la imagen): $f_{0x} = 0.5, f_{0y} = 0.5$
d. Frecuencia de corte (horizontal y vertical): $D_0 = fs/8$

Defina la ecuación del filtro paso bajo gaussiano (fpb) como:

```
d   =sqrt((Xc).^2 + (Yc).^2);  % Xc = (X - f0x),Yc = (X - f0y)
fpb=exp(-(d.*d)./(2*D0/fs*D0/fs));
```

Para definir el truncamiento del apartado III de la metodología utilice un número de sigmas (ancho de banda de la Gaussiana en el dominio espacial) como se describe a continuación:

```
ws =round(w*(1/(2*pi*D0/fs))); % w sigmas

filter_mask=h(cy-ws:cy+ws,cx-ws:cx+ws); % w sigmas
```

## *1.4 Ejercicio 4: Diseño a partir de filtros paso alto Laplaciano*

Obtenga, siguiendo el mismo proceso de los **ejercicios 2 y 3**, la máscara de filtrado espacial 5x5 correspondiente a una **Laplaciana**, como sigue:

a. Frecuencia de muestreo en el retículo ortogonal: fs = 400 .
b. Retículo ortogonal de vectores: $\vec{v}_1 = (1/fs, 0), \vec{v}_2 = (0, 1/fs)$
c. Posición central del filtro (en el centro de la imagen): $f_{0x} = 0.5, f_{0y} = 0.5$

Defina la ecuación del filtro paso alto laplaciano (fpa) como:

```
fpa = -((Xc.*Xc) + (Yc.*Yc)); % Xc = (X - f0x),Yc = (X - f0y)
```

Utilice el mismo método utilizado en el ejercicio 2 para realizar el truncamiento del apartado III de la metodología.

**Nota, en este caso para realizar la medida del error cometido el rango del estiramiento es el del filtro fpa [-0.5,0].**

Visualice y reflexione sobre la máscara de filtrado obtenida (utilice el comando `surf`). Comente y reflexione sobre los resultados obtenidos para los ejercicios 2,3 y 4.

**Le recomendamos que, una vez finalizada la práctica, observe el efecto que tiene la aplicación de las máscaras de filtrado creadas en los ejercicios 1-4. Para ello, utilícelas según lo explicado en la primera sesión de la práctica 5.**

# 2 Restauración lineal

Como han visto en teoría, el objetivo de la restauración lineal es recuperar una imagen que ha sido degradada de una manera conocida. El método consiste en modelar la degradación y aplicar el proceso inverso.

*a. Restauración lineal en ausencia de ruido*

Asuma que una imagen original $\psi(x, y)$ ha sido filtrada utilizando una máscara de filtrado $h(x, y)$ de manera que la imagen resultante $\theta(x, y)$ es el resultado de la operación:

$$\theta(x, y) = \psi(x, y) * h(x, y), \quad \eta(x, y) = 0$$

, donde $\eta(x, y)$ representa el ruido de la señal.

En este caso, la señal original puede recuperarse prácticamente sin alteraciones si conocemos el filtro aplicado. Para ello, lo más cómodo es trabajar en el domino frecuencial:

$$\Psi(u, v) = \frac{\Theta(u, v)}{H(u, v)}, \Psi(u, v) \xrightarrow{IFT} \psi(x, y)$$

*b. Restauración de una imagen afectada por ruido de cuantificación.*

Dada una imagen filtrada, `Ifilt`, la existencia de ruido de cuantificación puede simularse mediante la expresión:

```
Ifilt_Q = double(uint8(255.*Ifilt./max(Ifilt(:)))./255;
```

, note que este ruido es inherente a casi todos los procedimientos que hemos realizado durante las prácticas de lo que se deriva que es fácil encontrar imágenes afectadas por este tipo de ruido.

En este caso, el ruido de la imagen no es nulo, se produce al cuantificar los **más de 256** niveles de `Ifilt` en los **256** niveles de `Ifilt_Q`.

$$\theta(x,y) = \psi(x,y) * h(x,y) + \eta(x,y), \qquad \eta(x,y) \neq 0$$

Para eliminar la influencia de este ruido en la restauración, pueden utilizarse estimadores del ruido y compensarlo en el dominio frecuencial.

$$\tilde{\Psi}(u,v) = \frac{\Theta(u,v)}{H(u,v)} = \Psi(u,v) + \frac{N(u,v)}{H(u,v)}$$

Uno de los más usados es el filtrado por Wiener o MMSE, que en la práctica se aplica en el dominio frecuencial mediante la expresión:

$$\tilde{\Psi}(u,v) = \left[ \frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + \kappa} \right] \Theta(u,v)$$

, donde $\kappa$ es el factor de estimador de ruido.

En MatLab, este filtro puede definirse mediante la expresión:

```
Hc=H.*conj(H);
```

```
w_filter=(1./H).*(Hc./(Hc+k));
```

, donde `H` es el filtro en frecuencia utilizado para la transformación local lineal.

Para aplicarlo en frecuencia se debe multiplicar **`w_filter`** por la TF de la señal filtrada.

c. *Restauración de una imagen afectada por ruido blanco Gaussiano.*

El ruido blanco Gaussiano es el más usado para la verificación de algoritmos de estimación de ruido, dada su naturaleza aditiva, se entiende que, si no es muy intenso, puede detectarse y compensarse.

Dada una imagen filtrada, `Ifilt`, la existencia de ruido de blanco Gaussiano puede simularse mediante la expresión

```
Ifilt_G=imnoise(Ifilt,'gaussian',0,0.001);
```

, documéntese sobre la función `imnoise` mediante la ayuda de MatLab para conocer los parámetros que modelan el ruido.

El filtrado por Wiener descrito en el apartado anterior tiene como principal limitación que requiere conocer los espectros de potencia de señal y ruido (por separado) como datos para la estimación del parámetro $\kappa$. Esto en la práctica no suele suceder.

El filtrado CLS ofrece una alternativa cuyo parámetro modelador del ruido $\gamma$ (gamma) también es un escalar que requiere exclusivamente del conocimiento del ruido. En este caso de la media y la desviación típica del ruido gaussiano.

Su aplicación en frecuencia viene dada por la ecuación:

$$\tilde{\Psi}(u,v) = \left[ \frac{H^*(u,v)}{|H(u,v)|^2 + \gamma |L(u,v)|^2} \right] \Theta(u,v)$$

, donde $L(u,v)$ es un filtro Laplaciano.

En MatLab, esta operación puede realizarse mediante la expresión:

```
f_lapc= laplacian_filter.*conj(laplacian_filter);
cls_filter=conj(H)./(Hc+gamma*f_lapc);
```

, donde `laplacian_filter` es un filtro Laplaciano definido sobre el mismo retículo utilizado para la definición de `H`, sin frecuencia de corte.

Defina la ecuación del filtro paso alto laplaciano como:

```
laplacian_filter = -((X.*X) + (Y.*Y));
```

Para aplicarlo en frecuencia se debe multiplicar `cls_filter` por la TF de la señal filtrada.

## 2.1 Ejercicio 5: Restauración de una imagen en ausencia de ruido

Cargue el fichero `filteredI.mat`[1]. Para ello puede invocar al comando `load` de MatLab sin asignar la salida de `load` a ninguna variable. Se cargará la variable `Ifilt`, que contiene una imagen filtrada $\theta(x,y)$ con un filtro $H(u,v)$.

Sea $H(u,v)$ el filtro en frecuencia que se ha utilizado para conseguir `Ifilt`.

El filtro ha sido definido mediante la siguiente sucesión de comandos MatLab:

```
[rows,cols] = size(Ifilt);
x = [0:1/(cols-1):1] - 1/2;
y = [0:1/(rows-1):1] - 1/2;
k = pi;
ax =30;
ay =10;
[X,Y] = meshgrid(x,y);
H = k.*(ax.*(X))+(ay.*(Y)) + 1e-10;
H = (1./(H)).*sin(H).*exp(-j*H);
```

Obtenga y represente: el módulo del filtro $H(u,v)$ en frecuencia mediante la instrucción `imagesc` , la señal filtrada $\theta(x,y)$ y la parte real de la señal recuperada $\psi(x,y)$ asumiendo ausencia de ruido, ambas mediante `imshow`.

## 2.2 Ejercicio 6: Restauración de imágenes asumiendo ruido de cuantificación

Cargue el fichero `filteredI.mat`[2]. Para ello puede invocar al comando `load` de MatLab.

---

[1] Descárguese esta imagen del Moodle de la asignatura

[2] Descárguese esta imagen del Moodle de la asignatura

Simule la existencia de ruido de cuantificación (ver sección 1), obteniendo la imagen `Ifilt_Q`.

Restaure la imagen `Ifilt_Q` utilizando el filtro $H(u,v)$ definido en el ejercicio anterior asumiendo ausencia de ruido. Adicionalmente restaure la imagen `Ifilt_Q` mediante la aplicación del filtrado por Wiener con $\kappa = 0.00170$.

Compare y reflexione sobre los resultados obtenidos con las dos restauraciones, para ello deberá representar las señales involucradas de forma análoga a la realizada en el ejercicio anterior.

## *2.3 Ejercicio 7: Restauración de una imagen afectada por ruido blanco Gaussiano.*

Cargue el fichero `filteredI.mat`[3], para ello puede invocar al comando `load` de MatLab. Simule la existencia de ruido blanco Gaussiano de media 0 y varianza 0.001, obteniendo la imagen `Ifilt_G`.

> Nota: Para generar siempre el mismo ruido, utilice las siguientes líneas de código antes de llamar a **imnoise**:
>
> ```
> s = RandStream('mt19937ar','Seed',semilla);
> RandStream.setGlobalStream(s);
> ```
>
> Las soluciones orientativas has sido generadas con la semilla fijada a 0.

Restaure la imagen `Ifilt_G` directamente mediante la aproximación del ejercicio 1, mediante la aplicación del filtrado por Wiener del ejercicio 2 con $\kappa = 0.03083$ y mediante la aplicación del filtrado CLS con $\gamma = 235.0$. Para todos los casos utilice el filtro $H(u,v)$ del ejercicio 5.

Compare e reflexione sobre los resultados obtenidos, para ello deberá representar las señales involucradas de forma análoga a la realizada en los ejercicios anteriores.

[3] Descárguese esta imagen del Moodle de la asignatura