# PRACTICA 4: PIRÁMIDE GAUSSIANA Y LAPLACIANA

## ÍNDICE




Realizado por: Juan Pablo Cano López, jupacanolopez@correo.ugr.es

# 1-INTRODUCCIÓN

El objetivo de la práctica es aprender los conceptos y la utilización práctica de las pirámides Gaussiana y Laplaciana, en concreto, se va a fusionar una imagen con otra y también mejorar la eficiencia de la función de alineación que se realizó en la práctica 1. Para ello se aplica la pirámide gaussiana a la máscara(que consiste en una matriz de 1 a una mitad y 0 en la otra) y la pirámide laplaciana a las dos imágenes objetivo de fusión. Después se combinan las pirámides laplacianas de ambas imágenes en cada nivel y se les aplica la pirámide gaussiana de la máscara para ese nivel, finalmente se reconstruye la imagen resultante a partir de la función reconstruye, capaz de obtener la imagen que sería la original a partir de la pirámide laplaciana de una imagen.

# 2-RESPUESTA A LAS PREGUNTAS

**1-Nota que aplicar reduce a una imagen y después expand al resultado nos proporciona una imagen del mismo tamaño que la original pero no con los mismos valores. ¿Qué se ha perdido en el proceso?**

Si se aplica primero un reduce y luego un expand, al reducir el tamaño de la imagen, se reduce la calidad de la imagen, ya que al aplicar el reduce, se deben recodificar los valores de color anteriores para un número menor de píxeles, cuando se aplica el expand no se recuperan los valores perdidos, esta perdida es irreversible.

**2- La reconstrucción de la imagen a partir de pirámide laplaciana se realiza siguiendo el mismo proceso que hemos usado para la creación de la pirámide (completa esta función):**

```
function im = reconstruye(pirL)
% Reconstruye la imagen a partir de su pirámide laplaciana
%   IM = RECONSTRUYE(PIRL) reconstruye la imagen IM a partir de su pirámide
%   laplaciana PIRL.

niveles = size(pirL,2);
%%%% COMPLETAR ESTA PARTE
im = pirL{niveles};
for i=niveles:-1:2
 % expandimos
 aux = impyramid(im,'expand');
 % ajustamos el tamaño
 a= size(aux, 2);
 b= size(pirL{i-1}, 2);
 aux = padarray(aux, abs(size(pirL{i-1})-size(aux)), 'replicate', 'post');
 c= size(aux, 2);
 %aux = padarray(aux, size(pirL{i})-size(aux), 'replicate', 'post');
 %aux = padarray(aux, size(aux)-size(im), 'replicate', 'post');
 % sumamos
    im = aux + pirL{i-1}; %gaussiana anterior(aux) + diferencias
end
```

Fig1-Función de reconstrucción de la imagen

**3-En el script testmezcla.m está implementada la unión simple, la unión emborronando la máscara y, parcialmente, la fusión usando pirámide laplaciana. Completa el script para hacer fusión.**

```
%% Pirámide Gaussiana y Laplaciana
%se repite la matriz 1 vez en la primera y segunda y en la tercera
%dimensión tres veces -> RGB
%con el filtrogaussiano quitas altas frecuencias asi que las pierdes
%con tamaños impares de pixeles la imagen expandida y de nuevo reducida no
%tienen el mismo tamaño
%cell arrays porque las imagenes no tienen las mismas dimensiones asi que
%no se pueden guardar n+1= nivel final +1  nivel 1 primero.
%6 niveles

%Laplaciana Nivel más alto piramide laplaciana = nivel más alto gaussiana
%diferencias entre la expansión del nivel anterior y  la expandida de la
%gaussiana de más alto nivel.

A = im2double(imread('orange.jpeg'));
B = im2double(imread('apple.jpeg'));
M = im2double(imread('mascara.png'));

if size(M,3)==1 % Si la máscara no es RGB, convertir a RGB replicando
    M = repmat(M,[1,1,3]); %1 vez en cada dimensiñon y 3 veces (R G B)
end

% Unir las imágenes una junto a otra
union = A .* (1-M) + B .* M;

% Unirlas emborronando la máscara
Mb = imfilter(M, fspecial('gaussian', 45, 11),'replicate');
unionborrosa = A .* (1-Mb) + B .* Mb;
```

```
%% Fusión de pirámides laplacianas

% Calcular el número de niveles
niveles = floor(log2(min(size(A(:,:,1)))));
%El logaritmo de 2 es algo es el número de veces que puedes dividir por 2
%%
    % Crear las pirámides laplacianas de las imágenes y la pirámide
    % gaussiana de la máscara

    %%
    %% COMPLETAR ESTA PARTE
    ApirL = piramide_laplaciana(A, niveles);
    BpirL = piramide_laplaciana(B, niveles);
    MpirG = piramide_gaussiana(M, niveles);
    figure(1)
    imshow(montaje_piramide(ApirL,true))
    figure(2)
    imshow(montaje_piramide(BpirL,true))
    figure(3)
    imshow(montaje_piramide(MpirG,true))
    %%%% HASTA AQUI
    %%
     CpirL = cell(1, numel(ApirL));
     %fundir las pirámides laplacianas
    for i = 1 : length(ApirL)
        %%%% COMPLETAR ESTA PARTE
        CpirL{i} = ApirL{i} .* (1-MpirG{i}) + BpirL{i} .* MpirG{i};
        %%%% HASTA AQUI
    end

     % Create final blended image
     fusionlaplaciana = reconstruye(CpirL);
```

```
%% Mostrar las imágenes
figure(4);
subplot(2,3,1); imshow(A); title('Primera imagen');
subplot(2,3,2); imshow(B); title('Segunda imagen');
subplot(2,3,3); imshow(M); title('Máscara');
subplot(2,3,4); imshow(union); title('Unión simple');
subplot(2,3,5); imshow(unionborrosa); title('Unión borrosa');
subplot(2,3,6); imshow(fusionlaplaciana); title('Fusión de pirámides laplacianas');
truesize
```

Fig2-Script que realiza la fusión

**4-Fíjate que el rango de búsqueda es muy pequeño ¿por qué?(Alinear multiescala)**

Porque se reduce la imagen a 64x64 píxeles, no tendría sentido utilizar un rango de búsqueda grande. Además, cuanto menor sea el rango menor es el tiempo tardado en realizar la búsqueda.

**5- Completa los huecos en la función alinear_multiescala:**

```
function [Salida, dy, dx] = alinear_multiescala(imagen, referencia, rango)

% Calcular el número de niveles de la pirámide para que como máximo
% tengamos 64 píxeles de tamaño
niveles = floor(log2(min(ceil(size(imagen)/64))));

% Calcular las pirámides gaussianas de la imagen y la referencia
%%%% COMPLETAR ESTO
imgpirG =piramide_gaussiana(imagen, niveles); ;
refpirG = piramide_gaussiana(referencia, niveles);;
%%%% HASTA AQUI

% Calcular la alineación en el nivel más alto con el rango inicial
[Salida, dy, dx] = alinear(imgpirG{niveles+1}, refpirG{niveles+1}, rango);

% para cada nivel adicional refinar la alineación
rango = [1 1];
for nivel = niveles:-1:2
    % Al subir de nivel, el desplazamiento calculado se duplica
%%%% COMPLETAR ESTO
    dy = dy*2;
    dx = dx*2;
    % seleccionamos las imágenes a alinear
    %refescalada = impyramid( refpirG{nivel+1} ,'expand');
    %imgescalada = impyramid( imgpirG{nivel+1} ,'expand');
    refescalada = impyramid( refpirG{nivel} ,'expand');
    imgescalada = impyramid( imgpirG{nivel} ,'expand');

%%%% HASTA AQUI
    % colocamos la imagen en la posición ya calculada
    imgescalada = circshift(imgescalada, [dy dx]);
    % refinamos el desplazamiento
    [Salida, dyn, dxn] = alinear(imgescalada, refescalada, rango);
    %Salida = padarray(Salida, abs(size(Salida)-size(referencia)), 'replicate', 'post');
    % Calculamos el nuevo desplazamiento
%%%% COMPLETAR ESTO
    dy =dy+dyn;
    dx =dx+dxn;
%%%% HASTA AQUI
    fprintf('nivel=%d, dy=%d, dx=%d\n', nivel, dy, dx);
end
end
```

Fig3-Alinear_Multiescala.m

**6-Calcula los tiempos necesarios para alinear las imágenes con la estrategia multiescala y sin ella**

Se ha utilizado un rango [2, 2] y el tiempo en realizar la alineación ha sido de 0.811451s, de hecho no he conseguido alinear la imagen correctamente, en un tiempo menor a 3 minutos con el otro método.


Fig4-Tiempo transcurrido

## 3-IMÁGENES DE RESULTADOS Y COMENTARIOS

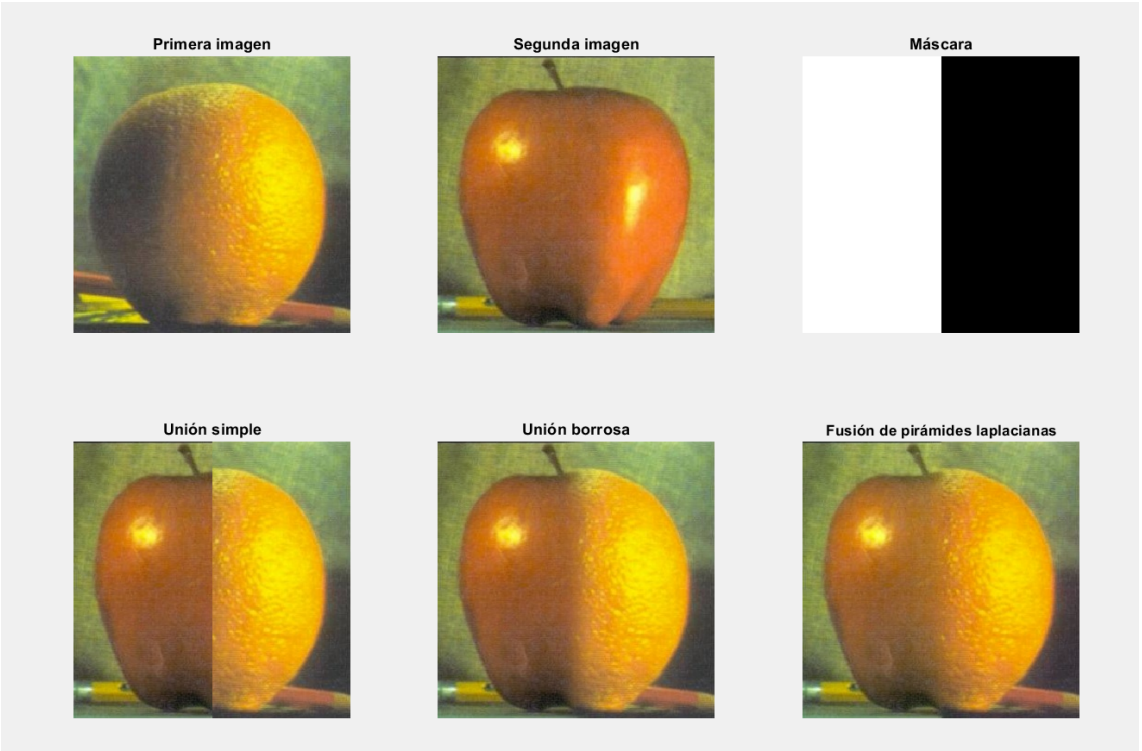
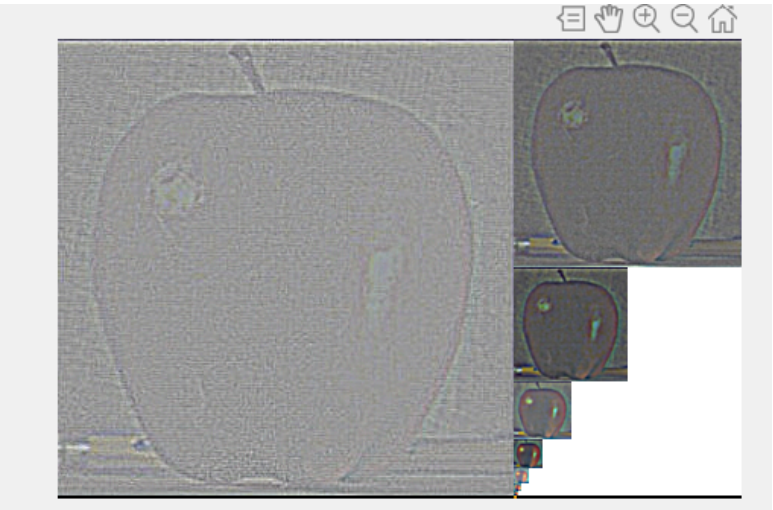



Fig5-Resultados de la fusión



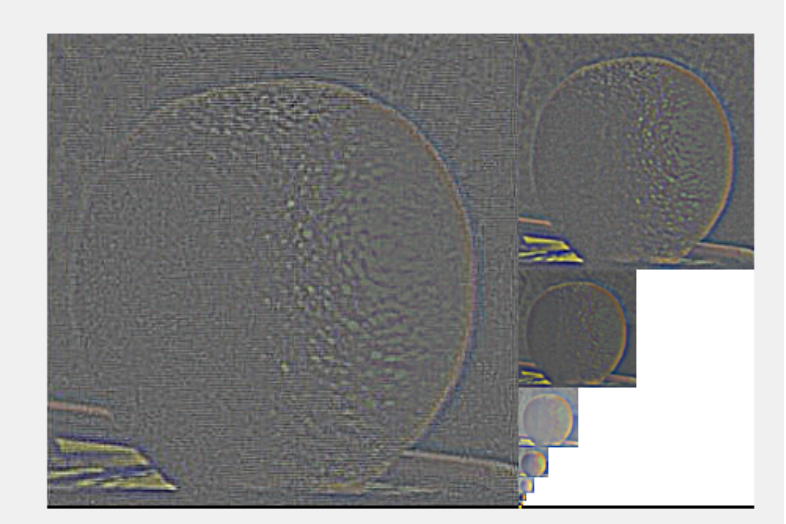


Fig6-Pirámides laplacianas de la naranja y la manzana


Fig7-Imagen Tiff aceptablemente alineada

La alineación no es perfecta dyR=47 pero se ha utilizado un rango [2, 2] y el tiempo en realizar la alineación ha sido de 0.811451s, un tiempo récord frente a los minutos que se tarda de la otra forma, para imágenes jpg, los resultados no son tan distantes, puede utilizarse el otro método sin demasiada inconveniencia.