Proyecto 1. Procesado de imágenes con MATLAB

Máster Universitario en Ingeniería de Telecomunicación

Procesado de señales acústicas e imágenes

Enrique Fernández Sánchez
Link al código: Github: Raniita/image-processing-matlab

21 de noviembre de 2021

Índice

1 Anonimizado **2**

2 Contraste **4**

3 Iluminación **5**

4 Suavizado **6**

5 Realzado **7**

6 Ruido **10**

7 Patrones **12**

8 Pseudocoloración **17**

Listado de códigos

1 Implementación anonimizado con MATLAB 2
2 Implementación de filtros para el realzado en MATLAB 7
3 Implementación filtro mediana para reducir ruido en MATLAB 10
4 Implementación detección de patrones en MATLAB 12
5 Implementación de pseucoloración en MATLAB 17

1 Anonimizado

En este ejercicio tenemos que realizar técnicas de anonimizado para recortar la información a procesar de la imagen 1, para ello podemos aplicar una técnica de máscara binaria o bien seleccionar manualmente la parte de la imagen a procesar.

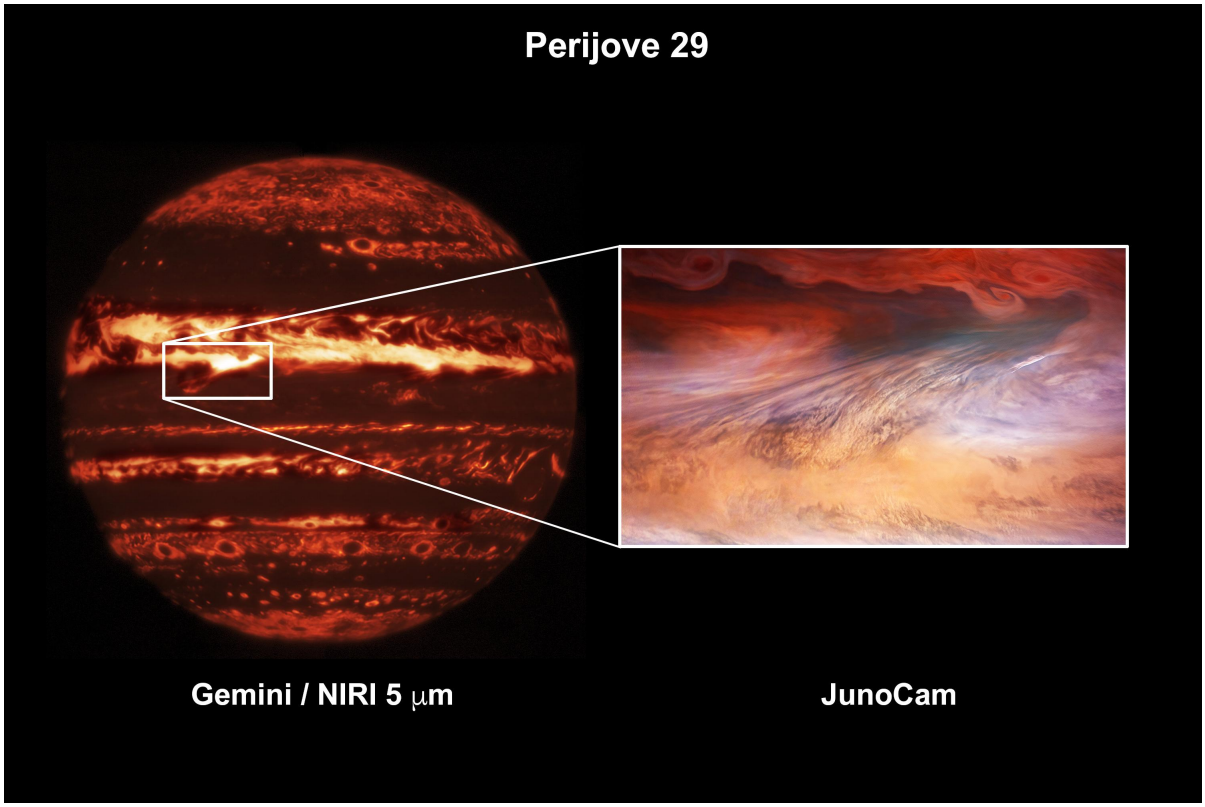

Figura 1: Imagen inicial anonimizado

En mi implementación, he optado por realizar una selección manual de la zona a recortar. Sin embargo, si tuviéramos que procesar muchas imágenes diferentes, la mejor opción sería realizar una máscara binaria, ya que así podríamos aplicarla de manera automática (sin intervención manual del usuario).

El código implementado sería el siguiente:

Código 1: Implementación anonimizado con MATLAB

```
% 1 - Anonimizado
% Enrique
% Ref: https://es.mathworks.com/help/images/ref/imcrop.html
clear;

% Cargamos la imagen
img = imread('anonimizado.jpg');


```

```
% Representamos la imagen original
figure
imshow(img);
title('Imagen inicial (sin recortar)')

% Recortamos la imagen con un rectangulo
[crop_img, rect_crop] = imcrop(img);

% Manera automatica
%rect_crop = [120 300 2750 1400];     % [xmin ymin width height]
%crop_img = imcrop(img, rect_crop);

% Comparativa imagen original vs imagen recortada
figure
%suptitle('Imagen Anonimizada')

subplot(1,2,1)
imshow(img)
title('Imagen inicial')

subplot(1,2,2)
imshow(crop_img)
title('Imagen recortada utilizando un rectangulo')

```

Una vez ejecutado el código, tendremos como salida una comparativa entre la imagen inicial (ver figura 1) y la imagen recortada:

Figura 2: Figura de salida al ejecutar script de MATLAB `anonimizado.m`

2 Contraste

3 Iluminación

4 Suavizado

5 Realzado

En este ejercicio tenemos que aplicar técnicas de realzado para obtener más detalles en la imagen X. Para ello, aplicaremos diferentes filtros, con el objetivo de resaltar los cambios entre las diferentes zonas de color de la imagen.

Figura 3: Imagen inicial realzado

Para realzar la imagen podemos aplicar filtros directamente sobre la imagen original. En este caso, hemos aplicado un preajuste del histograma, para hacer que el margen dinámico ocupe todo el histograma. Una vez tenemos la imagen preajustada, la filtramos con `prewitt` o `sobel`.

Código 2: Implementación de filtros para el realzado en `MATLAB`

```
% 5 - Realzado
% Enrique
clear;

img_rgb = imread('realzado.jpg');

% Preajuste expansion margen dinamico
img_adj = imadjust(img_rgb, stretchlim(img_rgb), [ ]);

% Caso uno, prewitt
filter = fspecial('prewitt');
img_filtered = imfilter(img_adj, filter, 'replicate');
%img_rgb_1 = imadd(img_adj, img_filtered);
img_rgb_1 = img_adj - img_filtered;

```

```
% Caso dos, sobel
filter = fspecial('sobel');
img_filtered = imfilter(img_adj, filter, 'replicate');
%img_rgb_2 = imadd(img_adj, img_filtered);
img_rgb_2 = img_adj - img_filtered;

figure
subplot(2,2,1)
imshow(img_rgb, [])
title('Imagen original')

subplot(2,2,2)
imshow(img_adj, [])
title('Imagen preadjustada')

subplot(2,2,3)
imshow(img_rgb_1, [])
title('Imagen realzada (prewitt)')

subplot(2,2,4)
imshow(img_rgb_2, [])
title('Imagen realzada (sobel)')


```

Una vez ejecutado el código, obtenemos como salida una figura comparativa entre la imagen original, la imagen preajustada y las opciones de filtrado (`prewitt` y `sobel`).

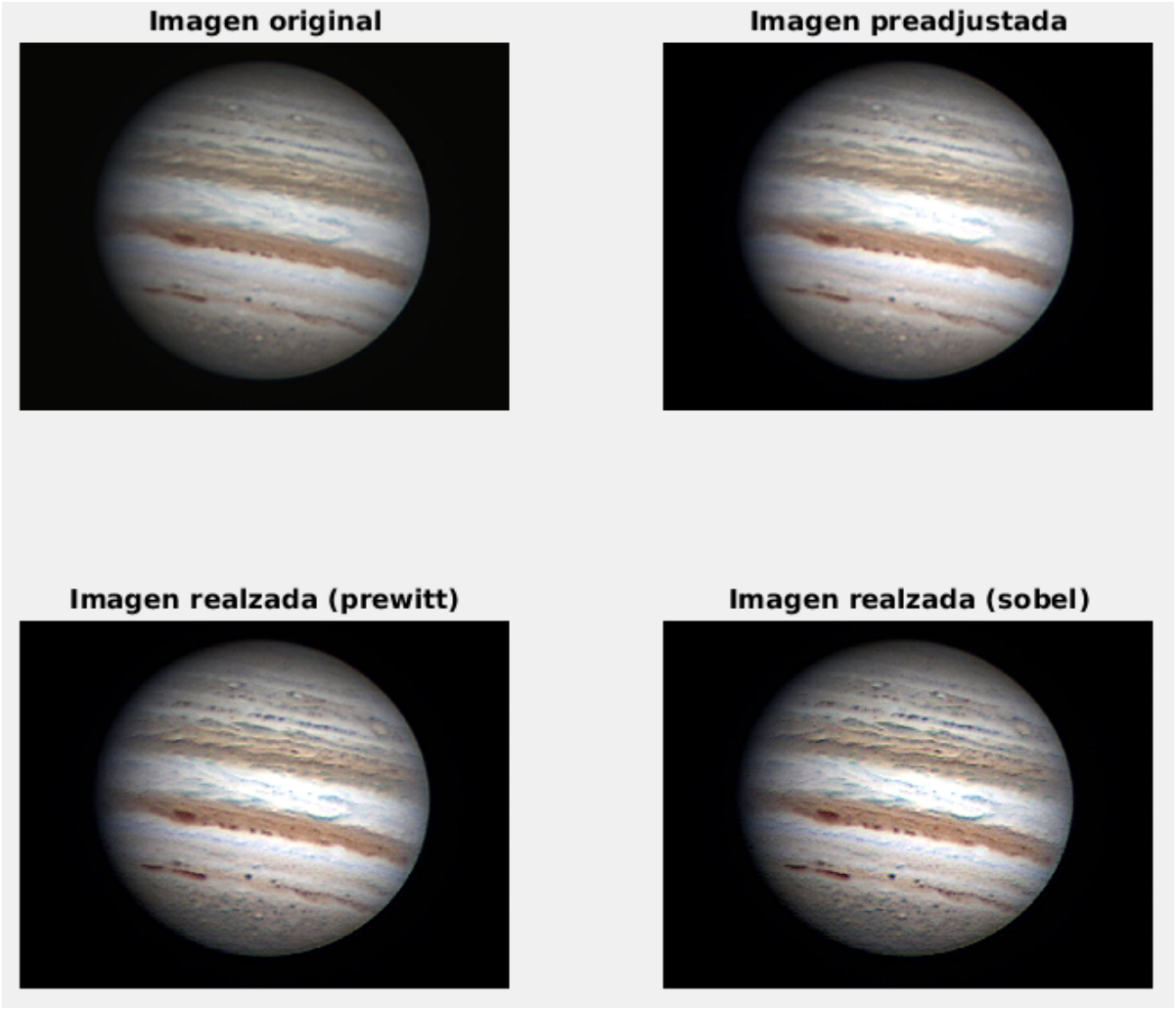

Figura 4: Imagen inicial ruido

Si tuvieramos que quedarnos con uno de los filtrados, el filtro `sobel` parece que realza mucho más las transiciones de color, cosa que en el caso de `prewitt` también distinguimos, pero en esta imagen en concreto el realzado es menor que respecto a `sobel`.

6 Ruido

En este ejercicio tenemos que aplicar técnicas de reducción de ruido a la imagen 5. Lo primero que tendríamos que intentar hacer es ver que tipo de ruido es, en este caso creemos que es ruido tipo "sal y pimienta".

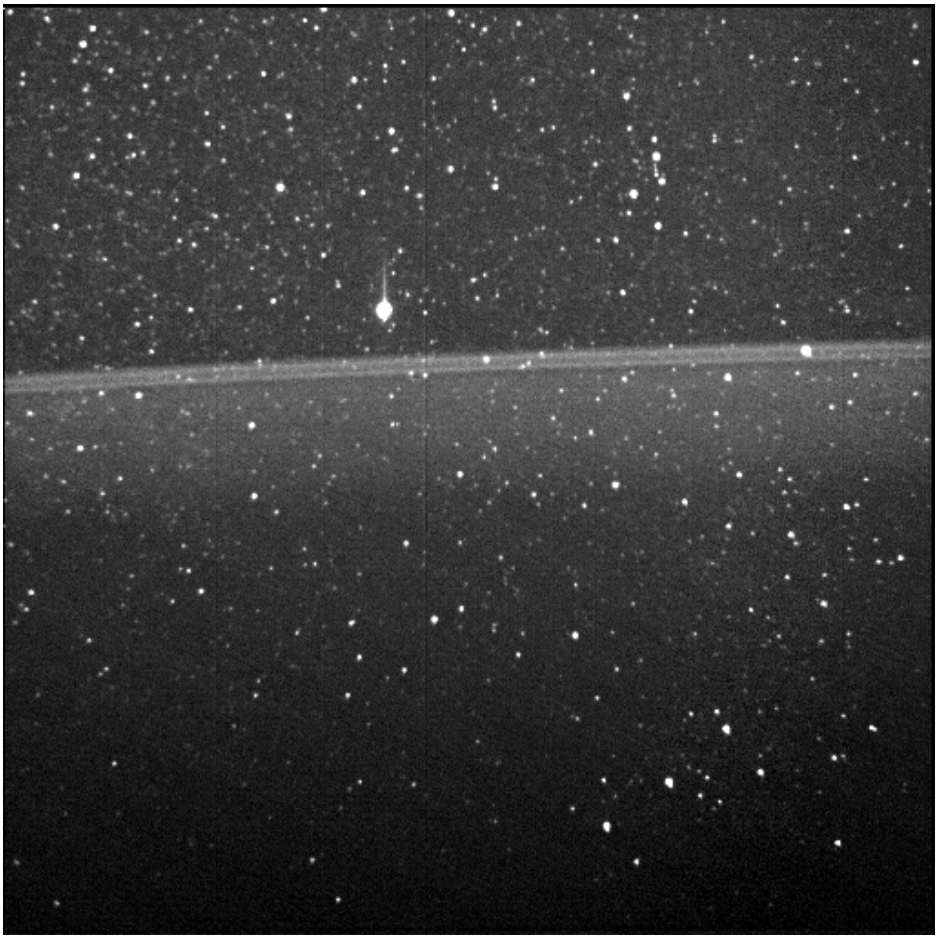

Figura 5: Imagen inicial ruido

Puesto que el ruido de la imagen se puede entender como un ruido de tipo "sal y pimienta", una de las técnicas que podemos utilizar es ejecutar de manera recursiva un filtro de mediana sobre la imagen.

Código 3: Implementación filtro mediana para reducir ruido en MATLAB

```
% 6 - Ruido Sal y Pimienta
% Enrique
clear;

img_rgb = 'ruido.jpg';
neighborn = 4;
times = 5;

```

```
img = imread(img_rgb);
img = mat2gray(img,[0 255]);
img = rgb2gray(img);

figure
subplot(1,2,1)
imshow(img, [])
title('Imagen original [RGB]')

% Aplicamos filtro mediana
img_median = img;
for n=1:times,
img_median = medfilt2(img_median, [1 1]*neighborn);
end

% Representamos resultado
subplot(1,2,2)
imshow(img_median, []),
axis off image,
title(['Imagen eliminado el ruido usando filtro mediana con N=' num2str(neighborn)
    ' T=' num2str(times) ])

```

Una vez ejecutado el código, tendremos como salida una comparativa entre la imagen inicial (ver figura 5) y imagen resultante tras eliminar el ruido "sal y pimienta":

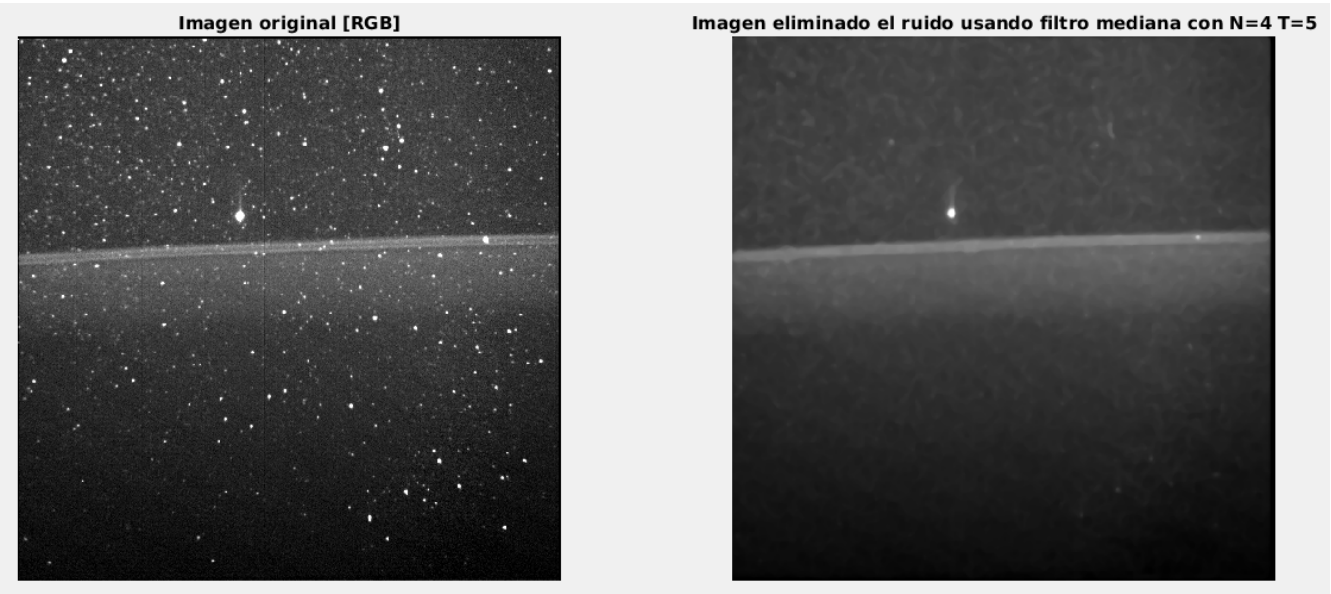

Figura 6: Figura de salida al ejecutar script de MATLAB `ruido.m`

7 Patrones

En este ejercicio vamos a practicar la técnica de detección de patrones utilizando el coeficiente de correlación de Pearson. Para ello, trabajamos sobre la imagen 7.

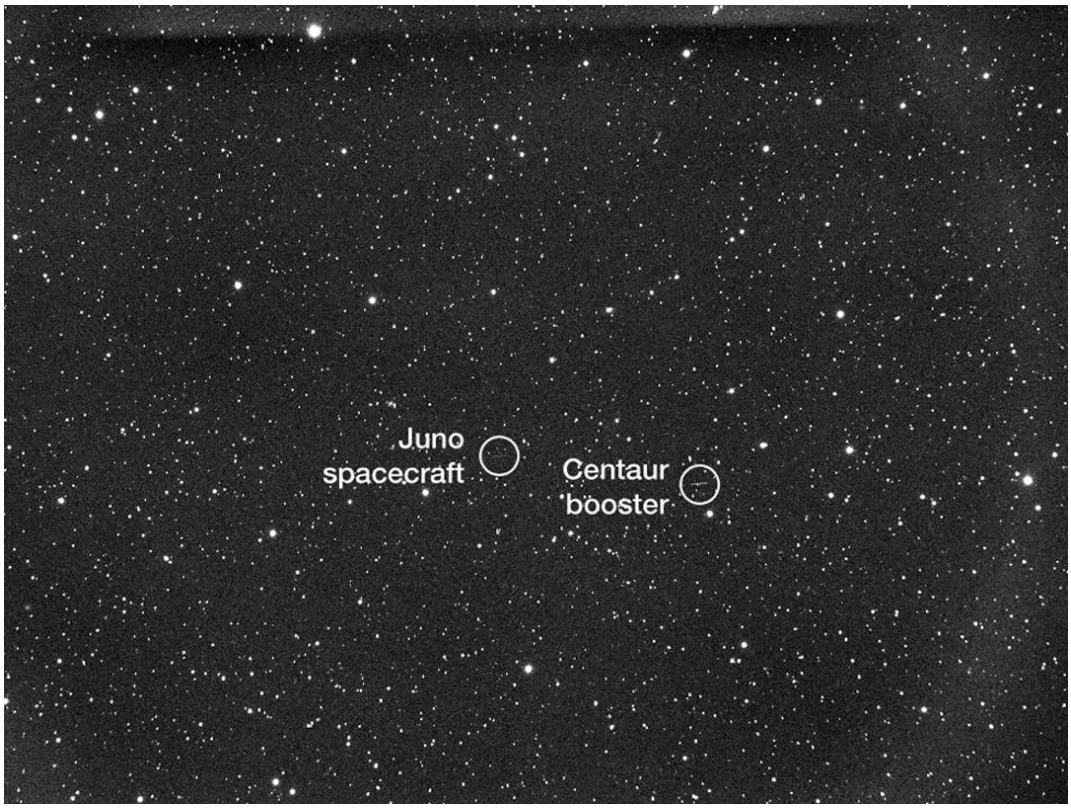

Figura 7: Imagen inicial de patrones

Utilizando la imagen 7, elegimos un patrón (una estrella) que queramos encontrar coincidencias. Una vez encontradas las coincidencias sobre el patrón establecido, volvemos a representar la imagen original resaltando dichas coincidencias.

Código 4: Implementación detección de patrones en MATLAB

```
% 7 - Patrones
% Enrique
% ref: https://es.mathworks.com/matlabcentral/answers/90094-how-to-convolved-two-
    image
clear;

img_rgb = 'patrones.jpg';
threshold_corre = 0.55;
threshold_stars = 0.85;

img_rgb = single(imread(img_rgb));
img_rgb = mat2gray(img_rgb,[0 255]);
img = rgb2gray(img_rgb);
```

```
13
14 % Funciones en una linea
15 lambda_covar = @(A,B) conv2(A-mean(A(:)), B(end:-1:1,end:-1:1)-mean(B(:)), 'valid')
       ;
16 lambda_autocovar = @(A,B) conv2(A.*A, ones(size(B)), 'valid')-conv2(A, ones(size(B)
       ), 'valid').^2/numel(B);
17
18 % Imagen original
19 figure,
20 imshow(img, [0 1]),
21 axis off image,
22 title('Imagen original [RGB]')
23
24 % Ejemplo de pattern: Pattern = PatternDetection(163:178,84:100);
25 p_img = round(ginput(2));        % Warning, must be integer to use as index
26 selected_pattern = img(p_img(1,2):p_img(2,2), p_img(1,1):p_img(2,1));
27
28 % Imagen del pattern seleccionado
29 figure,
30 subplot(1,2,1),
31 imshow(selected_pattern, [0 1]),
32 axis off image,
33 title('Patron seleccionado')
34
35 subplot(1,2,2),
36 mesh(selected_pattern),
37 axis off square,
38 set(gca,'XDir','reverse'),
39 title('Patron seleccionado (plot en elevacion)')
40
41
42 %% Deteccion del patron
43 coef_pearson = lambda_covar(single(img), single(selected_pattern));
44 coef_pearson_dem = sqrt(lambda_autocovar(img, selected_pattern).*lambda_autocovar(
       selected_pattern, selected_pattern));
45 index = find(coef_pearson_dem ~= 0);
46 coef_pearson(index) = coef_pearson(index)./coef_pearson_dem(index);
47
48 coef_pearson = padarray(coef_pearson, floor((size(selected_pattern)-1)/2), 0, 'post
       ');
49 coef_pearson = padarray(coef_pearson, ceil((size(selected_pattern)-1)/2), 0, 'pre')
       ;
50 coef_pearson = coef_pearson.*(coef_pearson > threshold_corre);
51
52 figure,
53 subplot(1,2,1),
54 imshow(coef_pearson*50+img, [0 1]),
55 axis off image,
56 title('Patrones detectados resaltados en amarillo')
57 % Resaltamos las ocurrencias con amarillo
58 map=colormap('gray');
59 map(256,:)=[1 1 0]; colormap(map)
60
61
```

```
subplot(1,2,2),
mesh(coef_pearson),
axis square,
set(gca,'YDir','reverse'),
title('Ocurrencias de la deteccion del patron (plot en elevacion)')

% Display number of ocurrencies on coef pearson
number_stars = sum(sum(coef_pearson > threshold_stars));
disp(['Numero de estrellas detectadas: ' num2str(number_stars)]);

```

Una vez ejecutamos el código, primero nos aparece una figura con título "*Imagen original [RGB]*", en la que tendremos que seleccionar la estrella a detectar (ver Figura 8). Tendremos que hacer click en las diagonales superior izquierda e inferior derecha del patrón (estrella) que queramos detectar, nos aparecerá una figura con el patrón que hemos seleccionado (ver Figura 9)

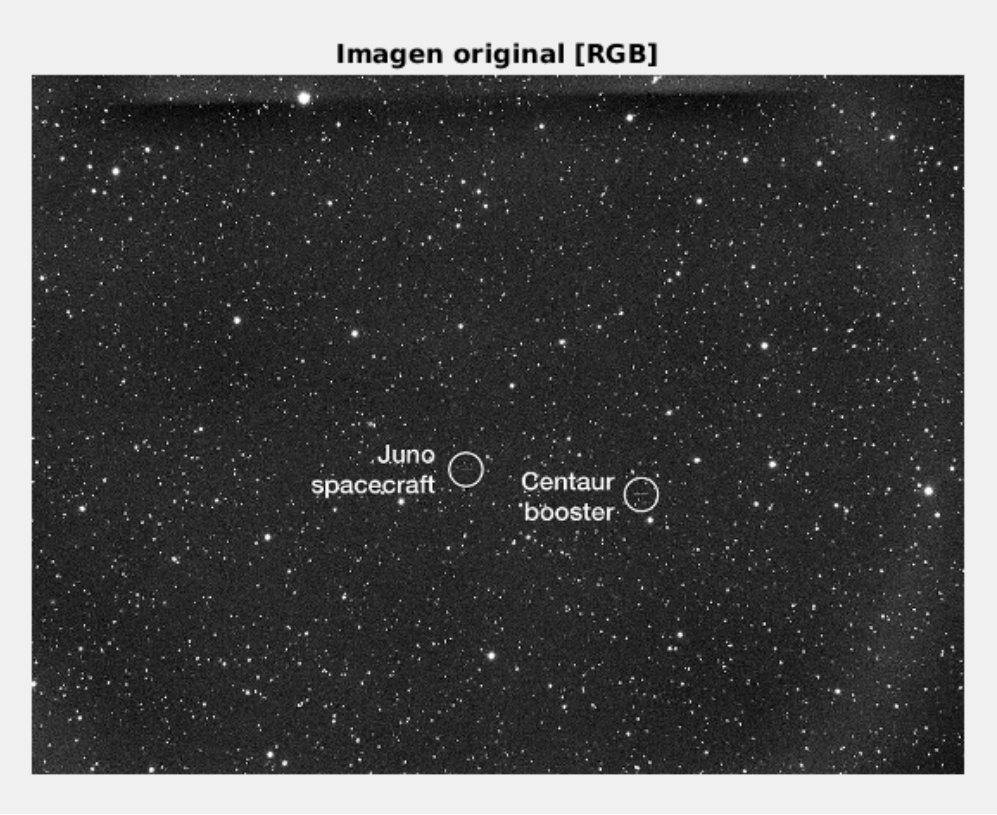

Figura 8: Imagen para seleccionar el patrón a buscar

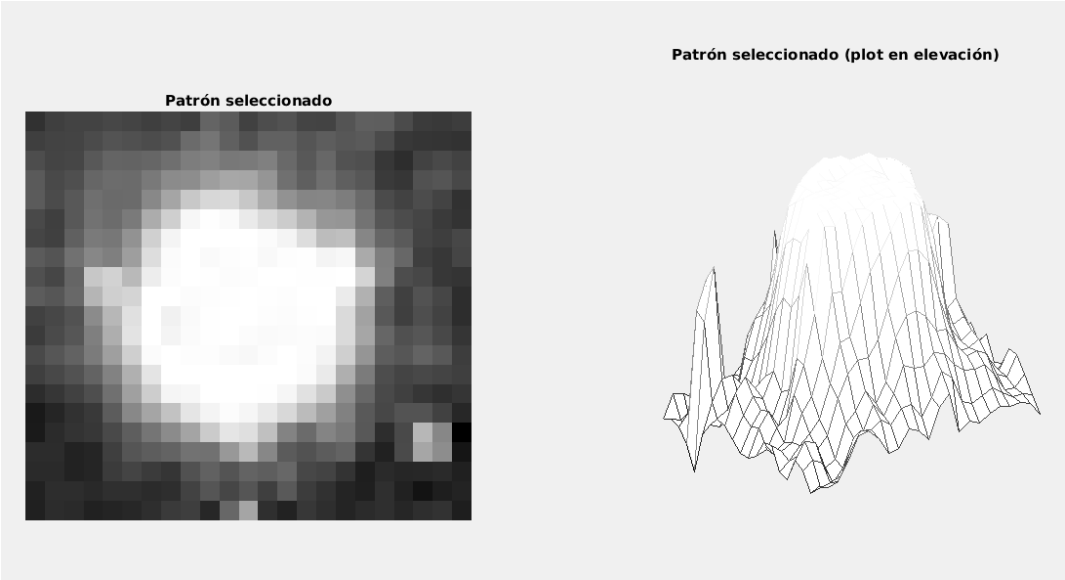

Figura 9: Figura resultante una vez seleccionamos el patrón

Una vez seleccionamos el patrón, podemos comprobar que además de la figura de la imagen 9, nos aparecer otra figura en la que se nos resalta en color amarillo las ocurrencias que tenemos en la imagen, en comparación con nuestro patrón (ver Figura 10).

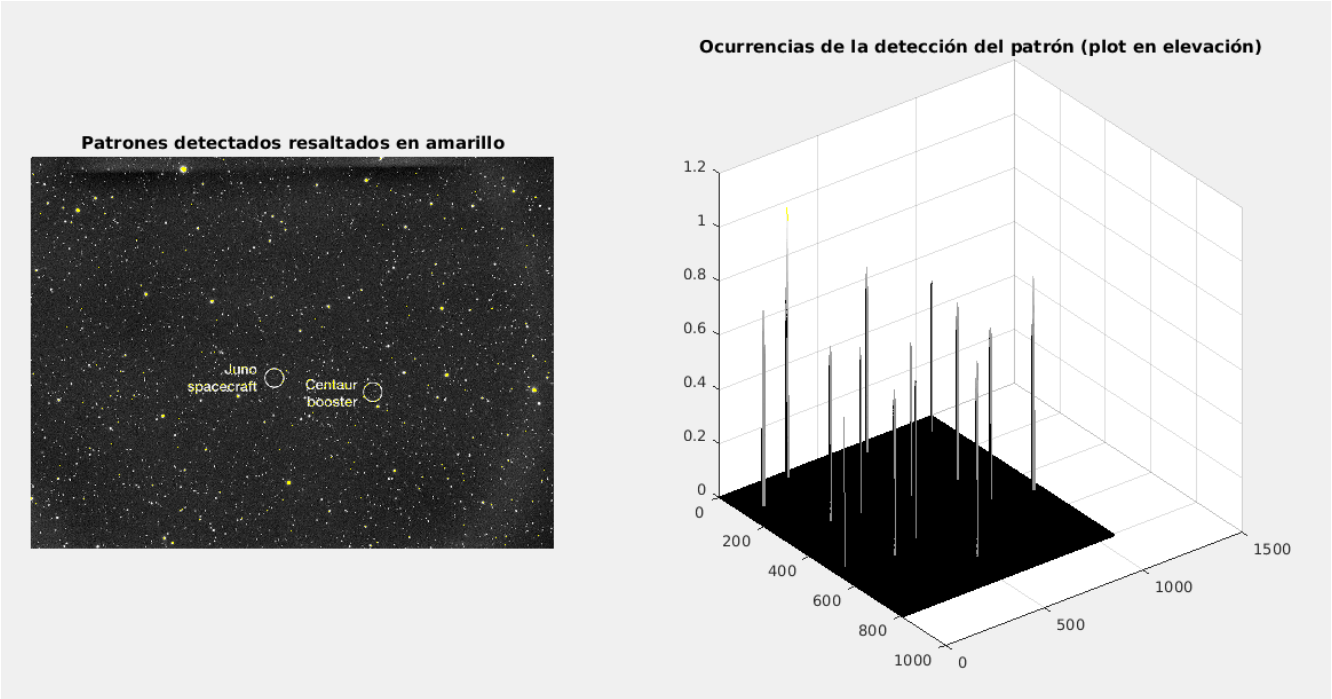

Figura 10: Figura resultante una vez seleccionamos el patrón

Además, hemos cuantificado el número de estrellas detectadas. Para ello, hemos tenido en cuenta un valor umbral por el cual consideramos una ocurrencia como una estrella. En la consola de MATLAB podemos comprobar cuantas estrellas hemos detectado (ver Figura 11).

```
>> patrones
Warning: Image is too big to fit on screen; displaying at 67%
> In images.internal.initSize (line 71)
  In imshow (line 305)
  In patrones (line 20)
Número de estrellas detectadas: 9
>>
```

Figura 11: Número de estrellas detectadas, mensaje en la consola en MATLAB

8 Pseudocoloración

En este ejercicio tenemos que aplicar técnicas de pseudocoloración a la imagen de un planeta (ver Figura X), con esto pretendemos ver detalles, que en el caso de ver la imagen en niveles de gris no veríamos.

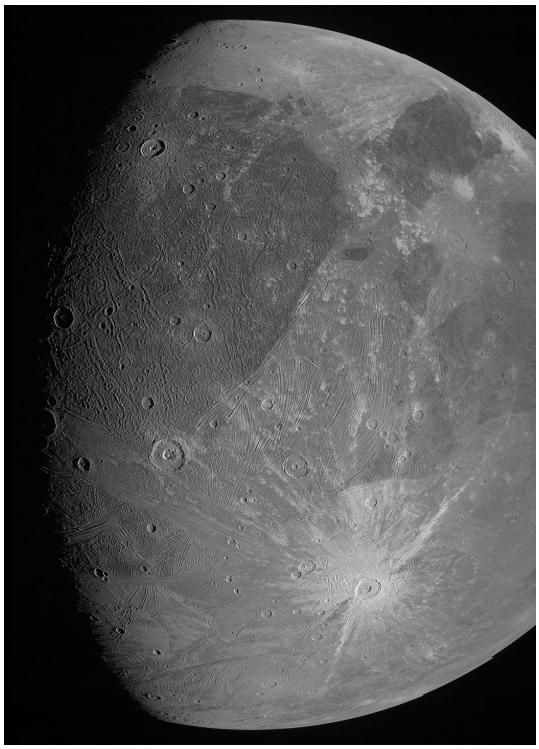

Figura 12: Imagen inicial pseudocoloración

Para la resolución de este ejercicio hemos usado la técnica de coloración de una imagen dependiendo de su nivel de gris. Para ello, hemos probado todos los mapas de colores que MATLAB nos ofrece, sin embargo solo `copper`, `pink`, `bone`, `hot` y `parula` nos aportaban resultados útiles.

Código 5: Implementación de pseucoloración en `MATLAB`

```
% 8 - Pseudocoloracion por nivel de gris
% Enrique

img_rgb = 'pseudocoloracion.jpg';
gray_lvl = 64;

img = imread(img_rgb);
img = mat2gray(img,[0 255]);
img = rgb2gray(img);

% Utilizamos 64 niveles de gris
img_colored = grayslice(img, gray_lvl);


```

```
    % Representamos cada una de las coloraciones
    figure
    subplot(2,3,1),
    subimage(img);
    axis off image,
    title('Imagen original en escala de gris')

    subplot(2,3,2),
    subimage(img_colored, copper(gray_lvl));
    axis off image,
    title('Pseudocoloracion (copper)')

    subplot(2,3,3),
    subimage(img_colored, pink(gray_lvl));
    axis off image,
    title('Pseudocoloracion (pink)')

    subplot(2,3,4),
    subimage(img_colored, bone(gray_lvl));
    axis off image,
    title('Pseudocoloracion (bone)')

    subplot(2,3,5),
    subimage(img_colored, hot(gray_lvl));
    axis off image,
    title('Pseudocoloracion (hot)')

    subplot(2,3,6),
    subimage(img_colored, parula(gray_lvl));
    axis off image,
    title('Pseudocoloracion (parula)')

```

Una vez ejecutado el código, tendremos como salida una comparativa entre la imagen inicial (ver figura 12) y las diferentes imágenes resultantes tras aplicar pseudocoloración:

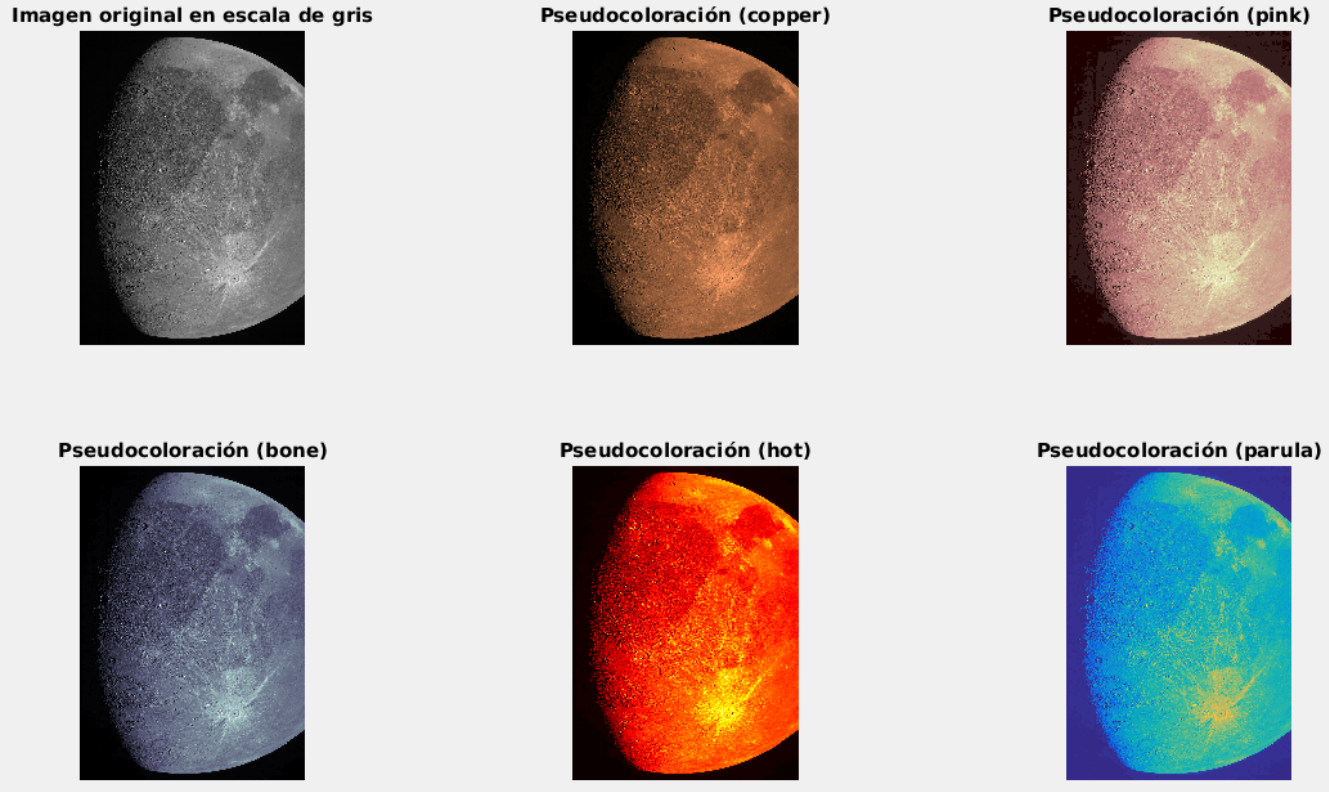

Figura 13: Figura de salida al ejecutar script de MATLAB `pseudocoloracion.m`

Si tuviera que elegir entre alguna de las imágenes obtenidas, considero que con `hot` y `parula` podemos distinguir más detalles, sin embargo el resto de coloraciones también aportan más detalle que la original.