

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

22 de noviembre de 2021

Índice

1	Anonimizado	3
2	Contraste	5
3	Iluminación	15
4	Suavizado	21
5	Realzado	25
6	Ruido	28
7	Patrones	30
8	Pseudocoloración	35

Listado de códigos

1	Implementación anonimizado con MATLAB	3
2	Implementación contraste con expansión de histograma en MATLAB	6
3	Implementación contraste usando <code>imcontrast</code> en MATLAB	8
4	Implementación contraste utilizando <code>histeq</code> en MATLAB	10
5	Implementación contraste utilizando <code>imadjust</code> , <code>histeq</code> y <code>adapthisteq</code> en LAB	12
6	Implementación iluminación utilizando ajustes en el histograma en MATLAB	15
7	Implementación iluminación utilizando filtrado homomórfico en MATLAB	18
8	Implementación de suavizado en el dominio espacial en MATLAB	21
9	Implementación suavizado en el dominio de la frecuencia en MATLAB	23
10	Implementación de filtros para el realzado en MATLAB	25
11	Implementación filtro mediana para reducir ruido en MATLAB	28
12	Implementación detección de patrones en MATLAB	30
13	Implementación de pseudocoloración en MATLAB	35

Índice de figuras

1	Imagen inicial anonimizado	3
2	Figura de salida al ejecutar script de MATLAB <code>anonimizado.m</code>	4
3	Imagen inicial para técnicas de contraste.	5
4	Figura resultante tras ejecución del script <code>contrast_imadjust.m</code>	7
5	Resultado tras ajuste con herramienta <code>imcontrast</code>	9
6	Ajuste de histograma utilizando herramienta <code>imcontrast</code>	10
7	Figura resultante tras la ejecución del script <code>contrast_histeq.m</code>	11
8	Figura resultante tras la ejecución del script <code>contrast_shadow.m</code>	13
9	Figura inicial iluminación	15
10	Figura resultante tras la ejecución del script <code>iluminacion_imadjust.m</code>	17
11	Figura resultante al ejecutar el script <code>iluminacion_homomorfico.m</code>	19
12	Imagen inicial suavizado	21
13	Figura resultante al ejecutar el script <code>suavizado_esp.m</code>	22
14	Figura resultante al ejecutar el script <code>suavizado_freq.m</code>	24
15	Imagen inicial realzado	25
16	Imagen inicial ruido	27
17	Imagen inicial ruido	28
18	Figura de salida al ejecutar script de MATLAB <code>ruido.m</code>	29
19	Imagen inicial de patrones	30
20	Imagen para seleccionar el patrón a buscar	32
21	Figura resultante una vez seleccionamos el patrón	33
22	Figura resultante una vez seleccionamos el patrón	33
23	Número de estrellas detectadas, mensaje en la consola en MATLAB	34
24	Imagen inicial pseudocoloración	35
25	Figura de salida al ejecutar script de MATLAB <code>pseudocoloracion.m</code>	37

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.

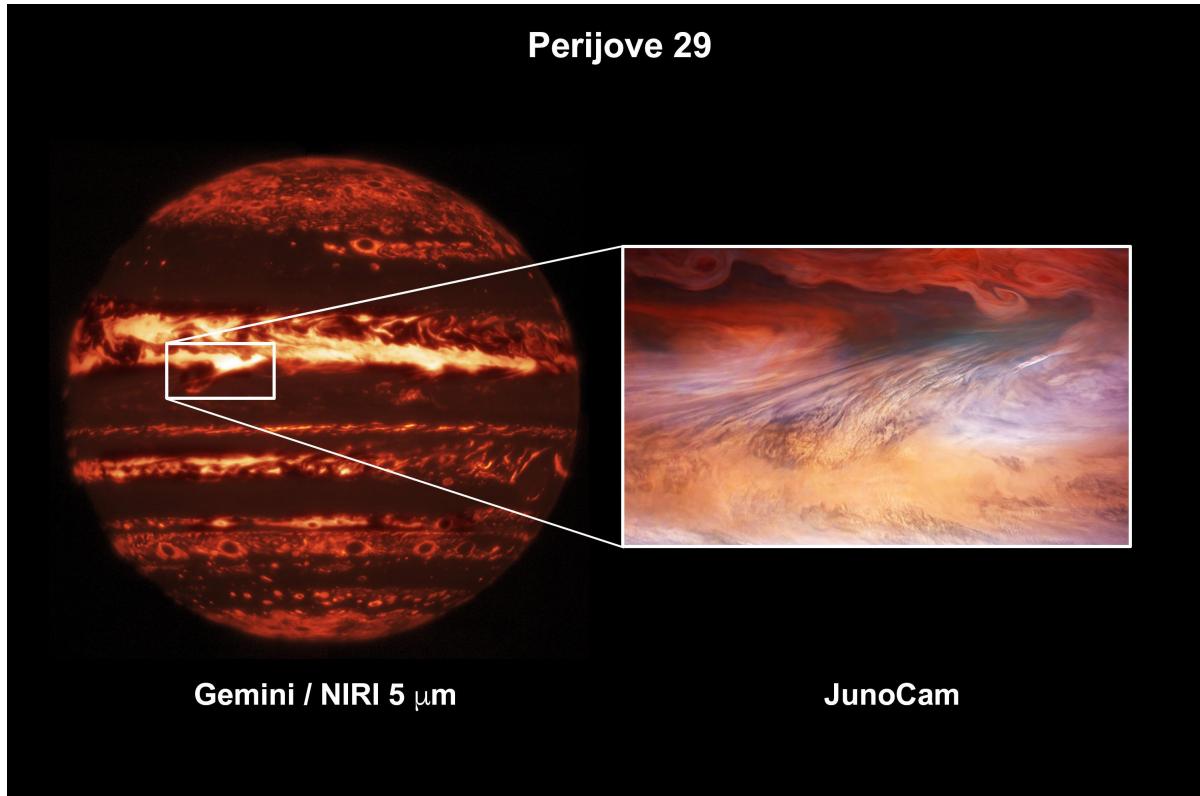


Figura 1: Imagen inicial anonimizado

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 % 1 - Anonimizado
2 % Enrique
3 % Ref: https://es.mathworks.com/help/images/ref/imcrop.html
4 clear;
5
6 % Cargamos la imagen
7 img = imread('anonimizado.jpg');
```

```

10 % Representamos la imagen original
11 figure
12 imshow(img);
13 title('Imagen inicial (sin recortar)')
14
15 % Recortamos la imagen con un rectangulo
16 [crop_img, rect_crop] = imcrop(img);
17
18 % Manera automatica
19 %rect_crop = [120 300 2750 1400];      % [xmin ymin width height]
20 %crop_img = imcrop(img, rect_crop);
21
22 % Comparativa imagen original vs imagen recortada
23 figure
24 %suptitle('Imagen Anonimizada')
25
26 subplot(1,2,1)
27 imshow(img)
28 title('Imagen inicial')
29
30 subplot(1,2,2)
31 imshow(crop_img)
32 title('Imagen recortada utilizando un rectangulo')
33

```

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

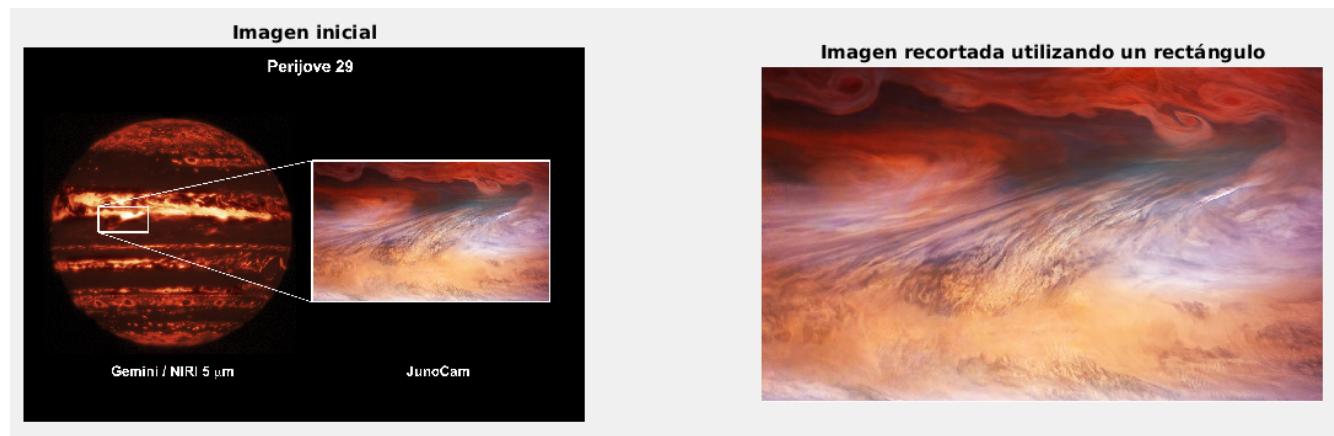


Figura 2: Figura de salida al ejecutar script de MATLAB anonimizado.m

2 Contraste

En este ejercicio trabajamos las diferentes técnicas que podemos utilizar para mejorar el contraste de una imagen. Para ello vamos a utilizar la siguiente imagen (ver Figura 3).

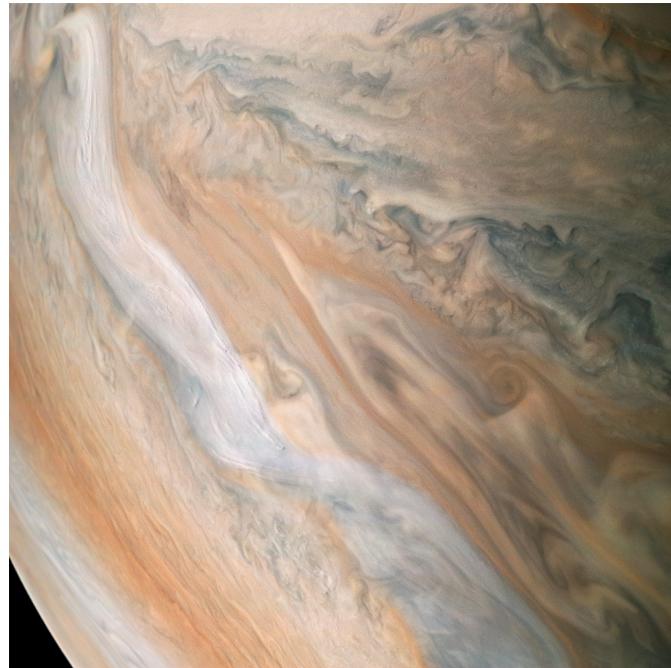


Figura 3: Imagen inicial para técnicas de contraste.

Puesto que tenemos diferentes maneras de mejorar el contraste de la imagen, hemos realizado diferentes ejemplos con el fin de explorar las posibilidades disponibles. Estas pruebas se resumen en:

- Expansión del histograma, ajuste de iluminación con gamma y filtros espaciales.
(`contraste_imadjust.m`)
- Corrección de histograma utilizando herramienta `imcontrast`.
(`contraste_imcontrast.m`)
- Ajuste canal por canal utilizando `histeq`.
(`contrast_histeq.m`)
- Ajustes de expansión, `histeq` y `adaphisteq` utilizando LAB.
(`contrast_shadow.m`)

Por lo tanto, procedemos a comentar cada una de las diferentes implementaciones.

Expansión del histograma, ajuste de iluminación y filtros espaciales.

En esta implementación lo que hacemos es aplicar una técnica de expansión del histograma, además de corregir la curva de histograma con el valor `cte_gamma`. Por último, intentamos aplicar un filtro espacial (`prewitt`) para resaltar los cambios de color.

Código 2: Implementación contraste con expansión de histograma en MATLAB

```
1 % 2 - Contraste
2 % Enrique
3 % Ref: https://es.mathworks.com/help/images/contrast-enhancement-techniques.html
4 clear;
5
6 % Ajuste del gamma
7 cte_gamma = 0.95;
8
9 % Cargamos la imagen
10 img = imread('contraste.jpg');
11 gray_img = rgb2gray(img);
12 hnorm_org = imhist(gray_img) ./ numel(gray_img);
13
14 % Representamos la imagen original
15 figure
16 subplot(2,3,1)
17 imshow(img);
18 title('Imagen inicial [RGB]')
19
20 subplot(2,3,4)
21 bar(hnorm_org,'stacked');
22 axis square off, axis([-2 255 0 0.03]),
23 title('Histograma original'),
24
25 % Expansion histograma
26 img_adj = imadjust(img, stretchlim(img), [ ], cte_gamma);
27 gray_adj = rgb2gray(img_adj);
28 hnorm_adj = imhist(gray_adj) ./ numel(gray_adj);
29
30 % Representamos la imagen ajustada
31 subplot(2,3,2)
32 imshow(img_adj);
33 title('Imagen expansion histograma')
34
35 subplot(2,3,5)
36 bar(hnorm_adj,'stacked');
37 axis square off, axis([-2 255 0 0.03]),
38 title('Histograma ADJ'),
39
40 % Filtro espacial
41 filter = fspecial('prewitt');
42 img_filtered = imfilter(img_adj, filter);
43 img_2 = imadd(img_adj, img_filtered);
44
45
```

```

46 gray_filtered = rgb2gray(img_2);
47 hnorm_filtered = imhist(gray_filtered)./numel(gray_filtered);
48
49 % Representamos la imagen filtrada
50 subplot(2,3,3)
51 imshow(img_2)
52 title('Imagen hist expandido + filtro')
53
54 subplot(2,3,6)
55 bar(hnorm_filtered,'stacked');
56 axis square off, axis([-2 255 0 0.03]),
57 title('Histograma expandido + filtro'),
58

```

Una vez ejecutamos dicho código, nos aparece la siguiente figura (ver Figuras 4).

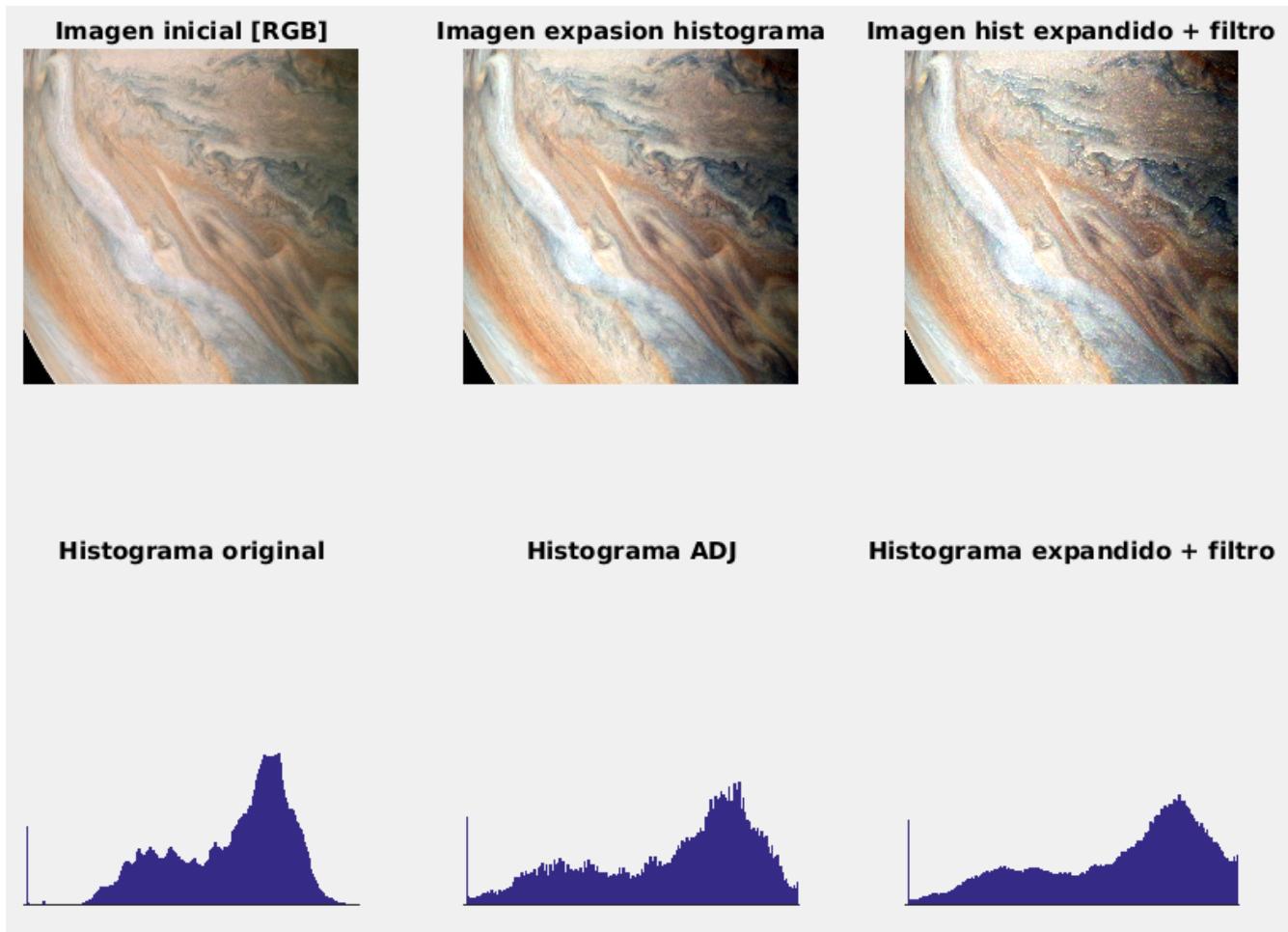


Figura 4: Figura resultante tras ejecución del script `contrast_imadjust.m`

Corrección de histograma utilizando herramienta **imcontrast**.

En este ejemplo lo que hacemos es utilizar la herramienta **imcontrast** para obtener los valores mínimo y máximo del histograma que creemos que mejora el contraste de nuestra imagen. Para ello, primero ajustamos con dicha herramienta utilizando la imagen en blanco y negro, y luego aplicamos dichos valores en la imagen a color.

Código 3: Implementación contraste usando **imcontrast** en MATLAB

```
1 % 2 - Contraste
2 % Enrique
3 % Ref: https://es.mathworks.com/help/images/contrast-enhancement-techniques.html
4 clear;
5
6 % Cargamos la imagen
7 img = imread('contraste.jpg');
8 img = mat2gray(img,[0 255]);
9 gray_img = rgb2gray(img);
10 hnrm_org = imhist(gray_img)./numel(gray_img);
11
12 figure
13 imshow(gray_img);
14 title('Imagen inicial [gris]')
15 % DESCOMENTAR PARA MOSTRAR HERRAMIENTA
16 imcontrast;
17
18 % Ajustamos con la herramienta imcontrast y sustituimos:
19 min_hist = 0.2198;
20 max_hist = 0.8866;
21 cte_gamma = 0.75;
22 img_contrast = imadjust(img, [min_hist max_hist], [0 1], cte_gamma);
23 gray_contrast = rgb2gray(img_contrast);
24 hnrm_contrast = imhist(gray_contrast)./numel(gray_contrast);
25
26 figure,
27 subplot(2,2,1)
28 imshow(img, [0 1])
29 title('Imagen inicial [RGB]')
30
31 subplot(2,2,3)
32 bar(hnrm_org,'stacked');
33 axis square off, axis([-2 255 0 0.03]),
34 title('Histograma imagen inicial'),
35
36 subplot(2,2,2)
37 imshow(img_contrast, [0 1])
38 title('Imagen ajustada con imcontrast')
39
40 subplot(2,2,4)
41 bar(hnrm_contrast,'stacked');
42 axis square off, axis([-2 255 0 0.03]),
43 title('Histograma post ajuste de contraste'),
```

Al ejecutar el script `contrast_imcontrast.m`, nos aparecerá la herramienta de `imcontrast` y diferentes figuras. Deberemos ajustar los valores de la ventana del margen dinámico utilizando dicha herramienta (ver Figura 6), después tendremos que volver a ejecutar el código con esos nuevos valores. De tal manera, nos aparecerá la imagen a color con la corrección de histograma aplicada (ver Figura 5).

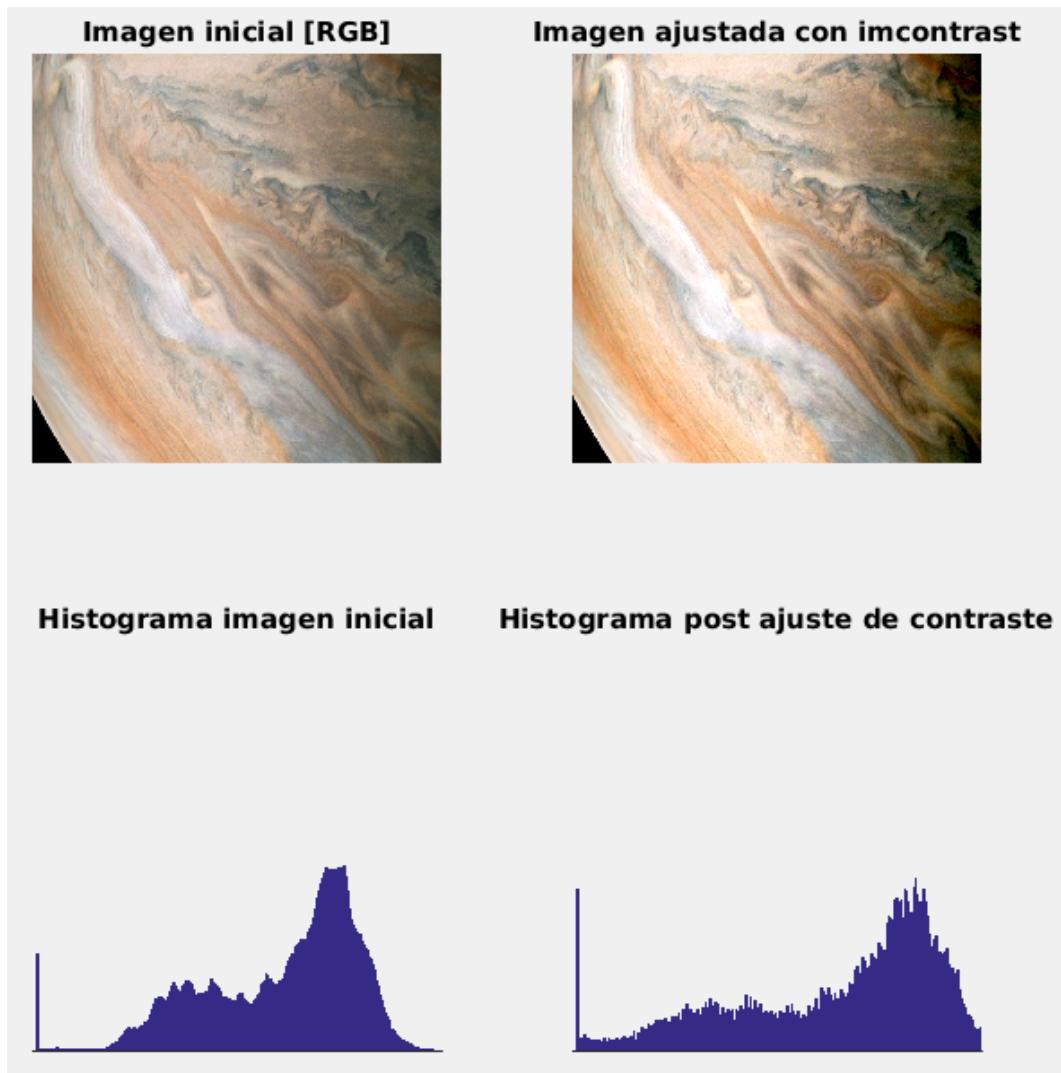


Figura 5: Resultado tras ajuste con herramienta `imcontrast`

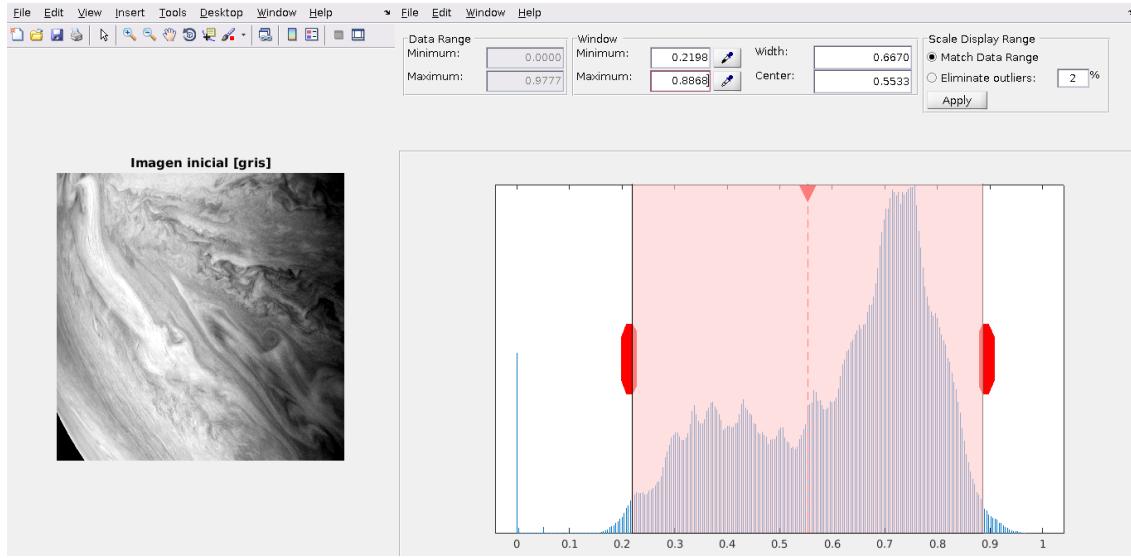


Figura 6: Ajuste de histograma utilizando herramienta `imcontrast`

Ajuste canal por canal utilizando `histeq`.

En este ejemplo lo que hacemos es utilizar técnicas de ecualización de histograma. Sin embargo, la función `histeq` de MATLAB solo admite imágenes con un único canal, por lo que realizamos dicha ecualización por cada canal de color de la imagen. Una vez tenemos cada canal ecualizado, creamos una imagen nueva con la información de esos tres canales.

Código 4: Implementación contraste utilizando `histeq` en MATLAB

```

1 % 2 - Contraste
2 % Enrique
3 % Ref: https://es.mathworks.com/help/images/contrast-enhancement-techniques.html
4 clear;
5
6 % Cargamos la imagen
7 img = imread('contraste.jpg');
8 gray_img = rgb2gray(img);
9 hnorm_org = imhist(gray_img) ./ numel(gray_img);
10
11 figure
12 subplot(2,2,1)
13 imshow(img);
14 title('Imagen inicial')
15 subplot(2,2,3)
16 bar(hnorm_org,'stacked');
17 axis square off, axis([-2 255 0 0.03]),
18 title('Histograma original'),
19
20 % Hacemos histeq a cada canal
21 R = histeq(img(:,:,1));
22 G = histeq(img(:,:,2));
23 B = histeq(img(:,:,3));

```

```

24
25 % Reconstruimos la imagen
26 img_contrast(:,:,1) = R;
27 img_contrast(:,:,2) = G;
28 img_contrast(:,:,3) = B;
29
30 contrast_gray = rgb2gray(img_contrast);
31 hnorm_gray = imhist(contrast_gray)./numel(contrast_gray);
32
33 % Representamos
34 subplot(2,2,2)
35 imshow(img_contrast, [])
36 title('Contraste canal por canal usando histeq')
37
38 subplot(2,2,4)
39 bar(hnorm_gray,'stacked');
40 axis square off, axis([-2 255 0 0.03]),
41 title('Histograma post ajuste de contraste'),
42

```

Una vez ejecutamos el script `contrast_histeq.m`, nos aparece una figura con la comparativa entre la imagen original y la imagen ajustada. Como podemos comprobar, si que conseguimos aumentar el contraste, pero los colores de la imagen original se ven modificados (ver Figura 7).

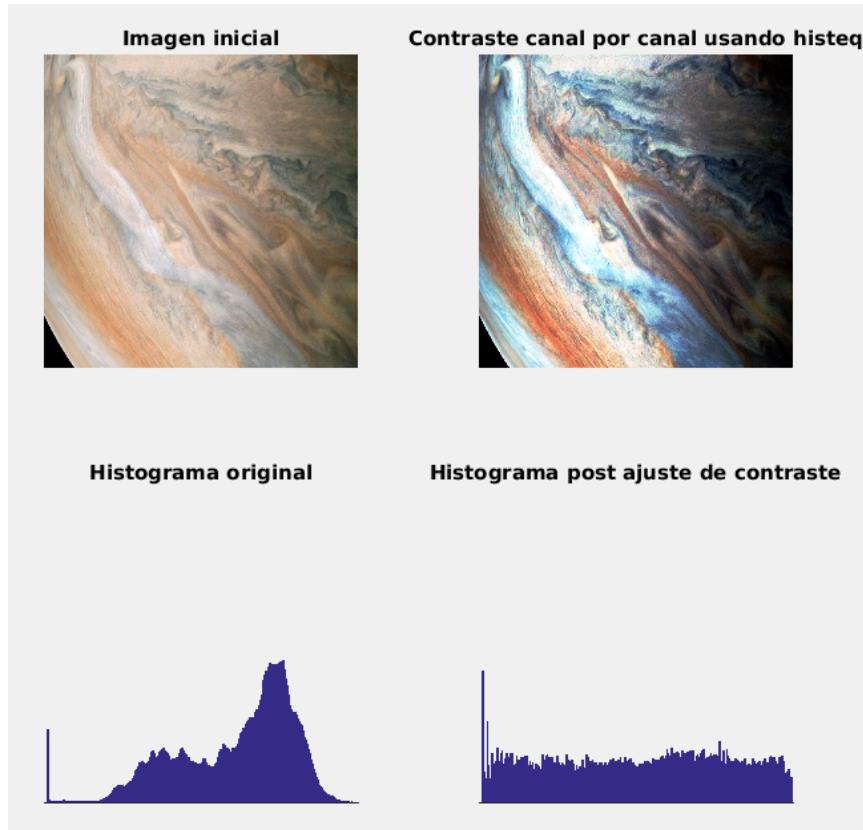


Figura 7: Figura resultante tras la ejecución del script `contrast_histeq.m`

Ajuste de expansión, **histeq** y **adapthisteq** utilizando **LAB**.

En este apartado hemos utilizado técnicas anteriores, pero cambiando a espacio de color tipo LAB, que consiste en un canal para luminancia y dos canales para colores. Hemos comprobado como resultaría utilizar técnicas de expansión de histograma, ecualización de histograma o adaptación de histograma en dicho espacio de color, y posteriormente convertir el resultado a imagen RGB.

Código 5: Implementación contraste utilizando `imadjust`, `histeq` y `adapthisteq` en LAB

```
1 % 2 - Contraste
2 % Enrique
3 % Ref: https://es.mathworks.com/help/images/contrast-enhancement-techniques.html
4 clear;
5
6 % Cargamos la imagen
7 X = imread('contraste.jpg');
8 shadow_lab = rgb2lab(X);
9
10 max_luminosity = 100;
11 L = shadow_lab(:, :, 1)/max_luminosity;
12
13 shadow_imadjust = shadow_lab;
14 shadow_imadjust(:, :, 1) = imadjust(L)*max_luminosity;
15 shadow_imadjust = lab2rgb(shadow_imadjust);
16 gray_adj = rgb2gray(shadow_imadjust);
17 hnorm_imadj = imhist(gray_adj)./numel(gray_adj);
18
19 shadow_histeq = shadow_lab;
20 shadow_histeq(:, :, 1) = histeq(L)*max_luminosity;
21 shadow_histeq = lab2rgb(shadow_histeq);
22 gray_histeq = rgb2gray(shadow_histeq);
23 hnorm_histeq = imhist(gray_histeq)./numel(gray_histeq);
24
25 shadow_adapthisteq = shadow_lab;
26 shadow_adapthisteq(:, :, 1) = adapthisteq(L)*max_luminosity;
27 shadow_adapthisteq = lab2rgb(shadow_adapthisteq);
28 gray_adapt = rgb2gray(shadow_adapthisteq);
29 hnorm_adapt = imhist(gray_adapt)./numel(gray_adapt);
30
31 %% Representamos
32 figure
33 subplot(2, 3, 1)
34 imshow(shadow_imadjust)
35 title('Imagen usando imadjust [LAB]')
36
37 subplot(2, 3, 4)
38 bar(hnorm_imadj, 'stacked');
39 axis square off, axis([-2 255 0 0.03]),
40 title('Histograma usando imadjust [LAB]'),
41
42 subplot(2, 3, 2)
43 imshow(shadow_histeq)
44 title('Imagen usando histeq [LAB]')
45
```

```

46 subplot(2,3,5)
47 bar(hnorm_histeq,'stacked');
48 axis square off, axis([-2 255 0 0.03]),
49 title('Histograma usando histeq [LAB]'),
50
51 subplot(2,3,3)
52 imshow(shadow_adapthisteq)
53 title('Imagen usando adapthiseq [LAB]')
54
55 subplot(2,3,6)
56 bar(hnorm_adapt,'stacked');
57 axis square off, axis([-2 255 0 0.03]),
58 title('Histograma usando adapthiseq [LAB]'),
59

```

Una vez ejecutamos el script `contrast_shadow.m`, nos aparece una figura comparativa entre las tres diferentes técnicas utilizadas.

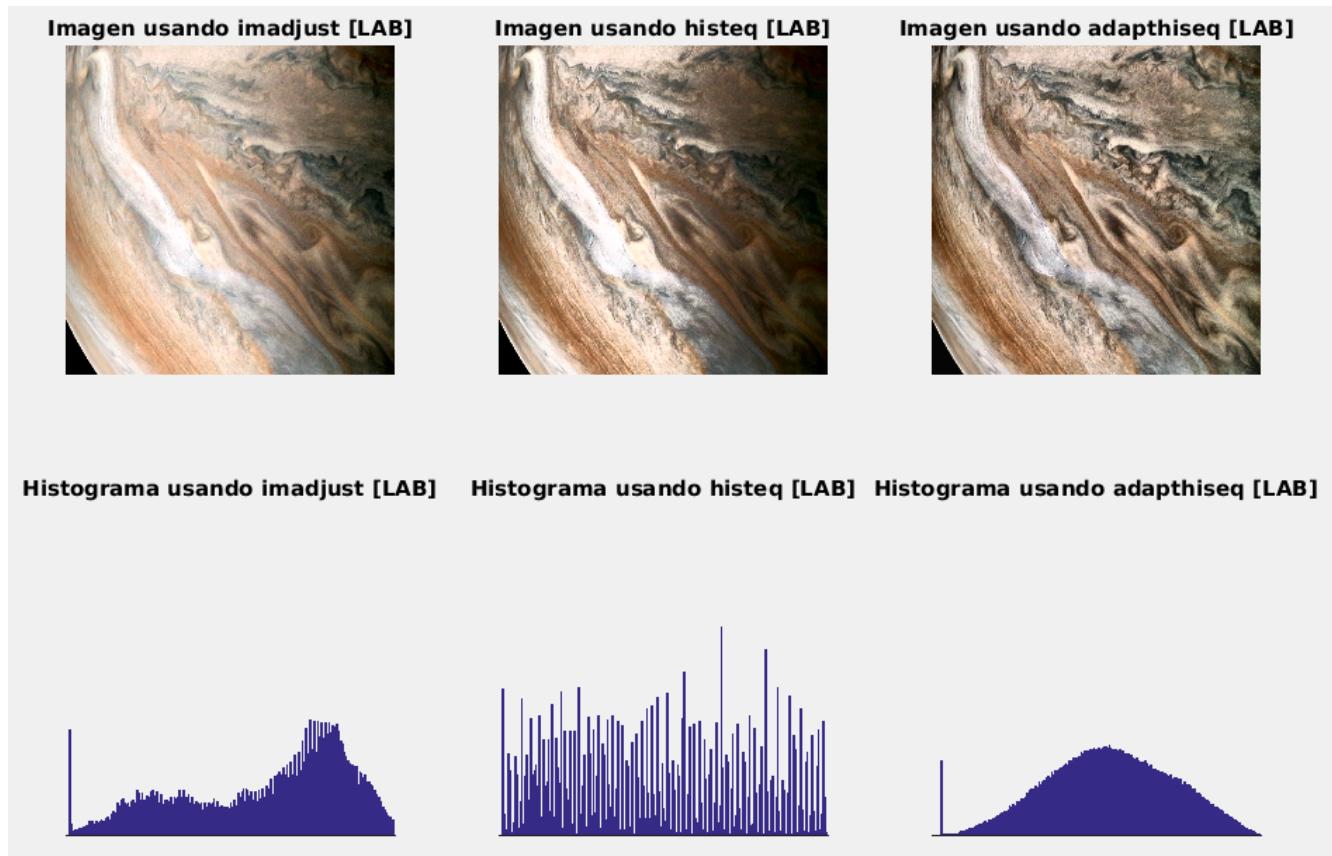


Figura 8: Figura resultante tras la ejecución del script `contrast_shadow.m`

Comparando las diferentes implementaciones que hemos realizado, llegamos a las siguientes conclusiones:

- La técnica de ajustar con `imcontrast` es la más fiel a la imagen inicial, en lo que a colores respecta. También la técnica de expansión de histograma funciona bien, aunque el resaltado con el filtro `prewitt` no realza mucho la imagen.
- La implementación utilizando `histeq` si que nos aporta una mejora considerable en el contraste, sin embargo los colores se ven modificados. Además, el histograma se ve muy modificado respecto la imagen inicial.
- Para la implementación en LAB comprobamos que las técnicas funcionan bien, pero tanto los colores como los bordes se intensifican mucho, sin embargo, la forma del histograma si que se parece a la de la imagen inicial.

3 Iluminación

En este ejercicio tenemos por objetivo mejorar la iluminación de la imagen 9. Para ello utilizaremos varias técnicas, una de ajuste de histograma y otra de filtrado.



Figura 9: Figura inicial iluminación

Para mejorar la iluminación de dicha imagen, vamos a probar con dos técnicas:

- Corrección del histograma. (`iluminacion_imadjust.m`)
- Filtrado homomórfico. (`iluminacion_homomorfico.m`)

Corrección de histograma.

En este apartado hacemos uso de la función `imadjust` para intentar ajustar el histograma de la imagen para que nos salga más iluminada. Para ello, seleccionamos la parte del histograma que consideremos, y la “copiamos” al histograma de la nueva imagen, comprobando que el margen dinámico ocupa todo el histograma.

Código 6: Implementación iluminación utilizando ajustes en el histograma en MATLAB

```
1 % 3 - Iluminacion usando imadjust
2 % Enrique
3 clear;
4
5 % Ajuste del gamma
6 cte_gamma = 5;
```

```

9 % Cargamos la imagen
10 img = imread('iluminacion.jpg');
11 gray_img = rgb2gray(img);
12 hnorm_org = imhist(gray_img) ./ numel(gray_img);
13 cdf_org = cumsum(hnorm_org);
14
15 figure
16 subplot(3,3,1)
17 imshow(img);
18 title('Imagen inicial')
19 subplot(3,3,4)
20 bar(hnorm_org,'stacked');
21 axis square off, axis([-2 255 0 0.03]),
22 title('Histograma original'),
23
24 subplot(3,3,7)
25 plot(linspace(0,1,length(cdf_org)),cdf_org),
26 axis([0 1 0 1]),
27 axis square,
28 grid,
29 title('CDF Inicial');
30
31 % Pre adj
32 img_pre = imadjust(img, [0 0.85], []);
33 gray_pre = rgb2gray(img_pre);
34 hnorm_pre = imhist(gray_pre) ./ numel(gray_pre);
35 cdf_pre = cumsum(hnorm_pre);
36
37 subplot(3,3,2)
38 imshow(img_pre)
39 title('Imagen preajustada ')
40 subplot(3,3,5)
41 bar(hnorm_pre,'stacked');
42 axis square off, axis([-2 255 0 0.03]),
43 title('Histograma preajustada'),
44
45 subplot(3,3,8)
46 plot(linspace(0,1,length(cdf_pre)),cdf_pre),
47 axis([0 1 0 1]),
48 axis square,
49 grid,
50 title('CDF Preajuste');
51
52 img_adj = imadjust(img_pre, [0 0.6], [0 1], 0.85); % [low_in high_in] [low_out
53 % high_out]
54
55 gray_adj = rgb2gray(img_adj);
56 hnorm_adj = imhist(gray_adj) ./ numel(gray_adj);
57 cdf_adj = cumsum(hnorm_adj);
58
59 % Filtrado sobel, le da mas realzado
60 %filter = fspecial('sobel');
61 %img_filtered = imfilter(img_pre, filter);
62 %img_adj = imadd(img_adj, img_filtered);

```

```

62 subplot(3,3,3)
63 imshow(img_adj);
64 title('Imagen iluminada')
65
66 subplot(3,3,6)
67 bar(hnorm_adj,'stacked');
68 axis square off, axis([-2 255 0 0.03]),
69 title('Histograma iluminado'),
70
71 subplot(3,3,9)
72 plot(linspace(0,1,length(cdf_adj)),cdf_adj),
73 axis([0 1 0 1]),
74 axis square,
75 grid,
76 title('CDF iluminado');
77

```

Una vez ejecutamos el script `iluminacion_imadjust.m`, obtenemos una imagen comparativa entre la imagen original y la imagen con la iluminación ajustada, además podemos ver la representación del histograma y de su CDF, respectivamente (ver Figura 10).

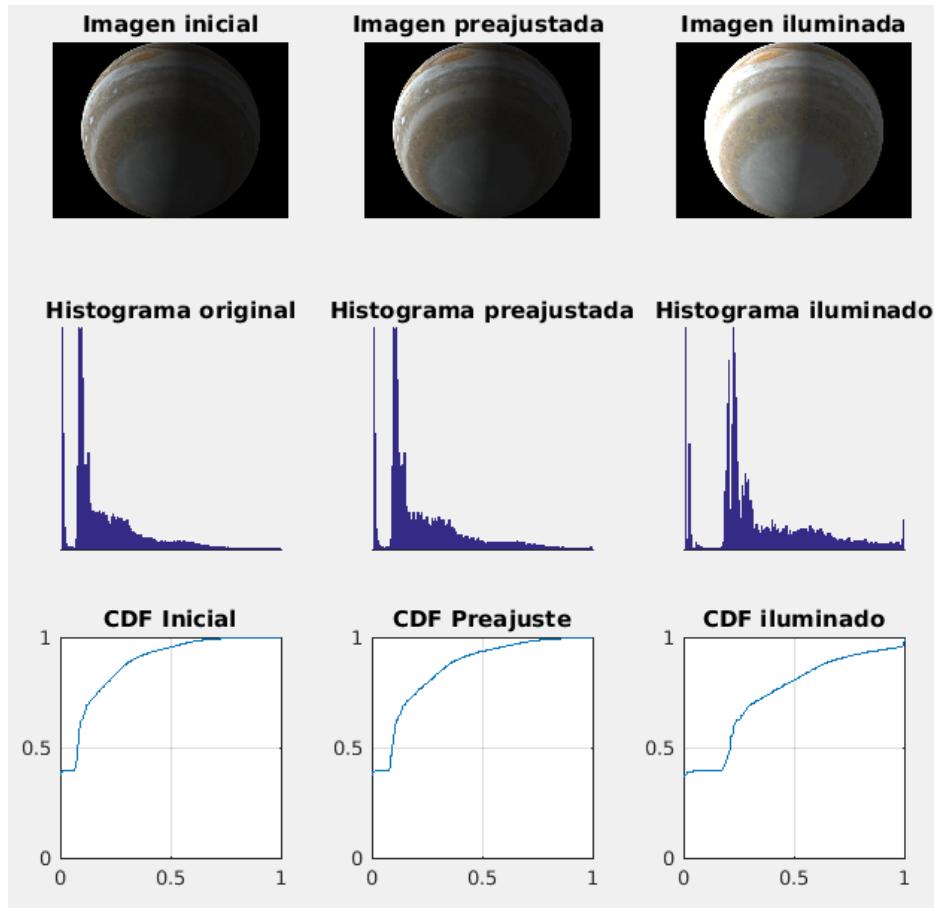


Figura 10: Figura resultante tras la ejecución del script `iluminacion_imadjust.m`

Filtrado homomórfico.

En este apartado hacemos aplicamos la técnica de filtrado homomórfico, con el objetivo de mejorar la iluminación de la zona más “oscura”, intentando afectar lo menos posible a la zona más “clara”. Para ello trabajaremos en el dominio de la frecuencia, en donde aplicaremos el filtrado por cada uno de los canales R,G y B de la imagen, para más tarde reconstruir la imagen una vez filtrada.

Código 7: Implementación iluminación utilizando filtrado homomórfico en MATLAB

```
1 % 3 - Iluminacion usando filtro homomorfico
2 % Enrique
3 clear;
4
5 img = 'iluminacion.jpg';
6 img_rgb = imread(img);
7
8 img_src = single(imread(img));
9 img_src = mat2gray(img_src, [0 255]);
10 %img_src = rgb2gray(img_src);
11
12 freq = 0.004;
13
14 Slope = 8;
15 N = size(img_src);
16 [u,v] = freqspace(N,'meshgrid');
17 D = sqrt(u.^2+v.^2);
18 H = fftshift(1-exp(-Slope*(D.^2/(2*freq.^2)) ));
19
20 %gammaL = 1 - mean(img_src(:));
21 %gammaL = 0.95;
22 gammaL= 0.92;
23 %gammaH = 1 + mean(img_src(:));
24 %gammaH = 1.1;
25 gammaH = 1.03;
26
27 H = ((gammaH - gammaL) .* H) + gammaL;
28
29 % Pasamos al dominio de la frecuencia
30 fft2_rgb = fft2(reallog(img_src+1e-6));
31
32 % Canal R
33 HomomorphicFilterHF_1 = exp(real(ifft2(H.*fft2_rgb(:,:,1))))-1e-6;
34
35 % Canal G
36 HomomorphicFilterHF_2 = exp(real(ifft2(H.*fft2_rgb(:,:,2))))-1e-6;
37
38 % Canal B
39 HomomorphicFilterHF_3 = exp(real(ifft2(H.*fft2_rgb(:,:,3))))-1e-6;
40
41 % Reconstruimos la imagen
42 img_rgb_adj(:,: , 1) = HomomorphicFilterHF_1;
43 img_rgb_adj(:,: , 2) = HomomorphicFilterHF_2;
44 img_rgb_adj(:,: , 3) = HomomorphicFilterHF_3;
```

```

45
46
47 % Representamos ambas imagenes
48 figure,
49 subplot(1,2,1),
50 imshow(img_src, [0 1]),
51 axis off image,
52 title('Imagen original [RGB]')
53
54 subplot(1,2,2),
55 imshow(img_rgb_adj, [0 1]),
56 axis off image,
57 title(['Imagen filtrada [RGB] (Freq corte = ' num2str(freq,3) ', pendiente = '
58     num2str(Slope,3) ', \gamma_L = ' num2str(gammaL,3) ', \gamma_H = ' num2str(
      gammaH,3) ')'])

```

Una vez ejecutamos el script `iluminacion_homomorfico.m`, aparece una figura en la que podemos comparar la imagen inicial con la imagen resultante tras el filtrado homomórfico (ver Figura 11). Podemos comprobar como se iluminan más zonas respecto a lo que obtuvimos en la técnica de corrección de histograma, aunque en la zona de la izquierda la imagen ha quedado un poco “quemada”, algo que se podría arreglar con un ajuste fino de γ_L y γ_H .

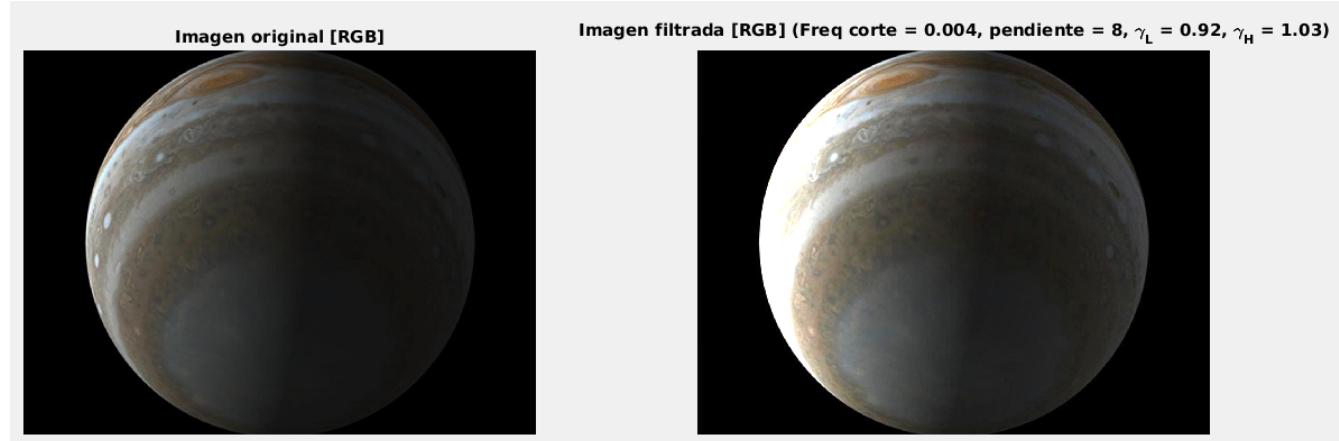


Figura 11: Figura resultante al ejecutar el script `iluminacion_homomorfico.m`

Una vez comprobadas las diferentes técnicas, extraemos las siguientes conclusiones:

- Ninguna de las técnicas implementadas mejora considerablemente la iluminación de la imagen, ya que cuando conseguimos mejorar la zona más oscura, la zona que ya estaba iluminada empieza a iluminarse de más (perdiendo información, ya que parece como si la imagen estuviera quemada).
- Las técnicas necesitan un ajuste muy fino para llegar a resultados que consideremos muy buenos.
- Ambas técnicas funcionan bien para iluminar lo suficiente la parte más oscura, de modo que podamos interpretar nuevos detalles de la imagen.

4 Suavizado

En este apartado tenemos que aplicar técnicas de suavizado a la imagen 12. El objetivo es obtener una imagen en la que se vean suavizados los diferentes desperfectos.

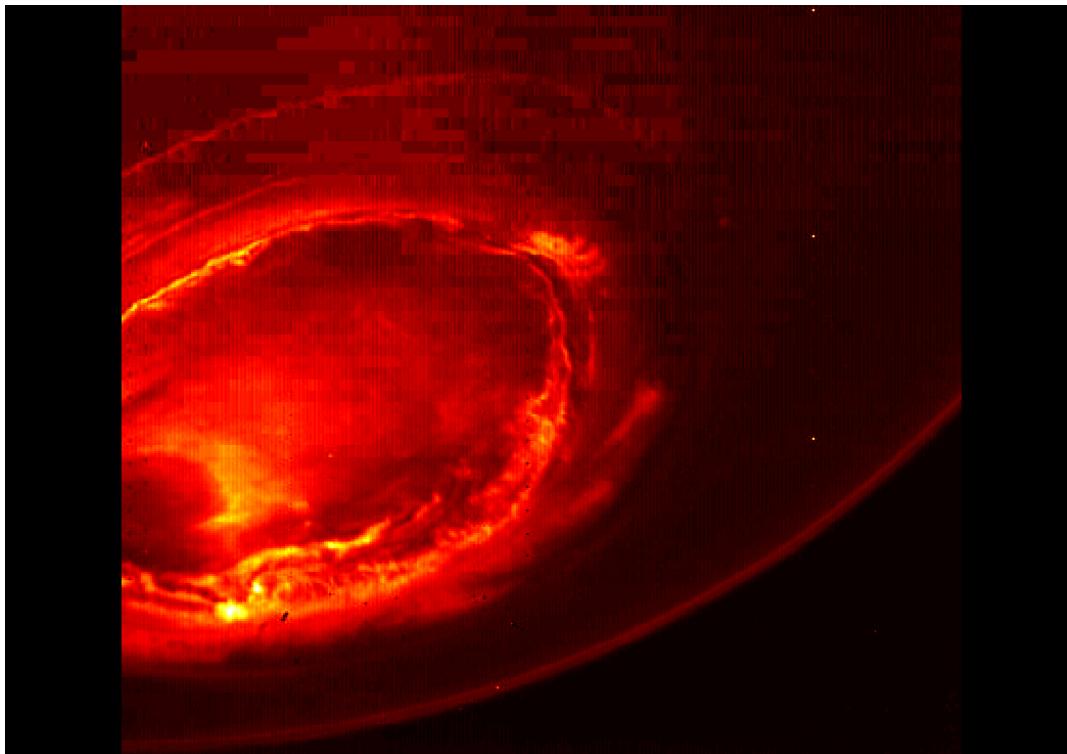


Figura 12: Imagen inicial suavizado

Para este caso, vamos a utilizar técnicas de filtrado de imagen. Además, vamos a comprar como realizar el filtrado en el dominio del espacio (`suavizado_esp.m`) y en el dominio de la frecuencia (`suavizado_freq.m`).

Dominio del espacio

En el dominio del espacio, vamos a utilizar un filtrado gaussiano, este filtrado lo podríamos implementar con las funciones `fspecial()` y `imfilter()`, sin embargo, vamos a utilizar una función especial del paquete de Image Processing Toolbox, llamada `imgaussfilt()`, que sería equivalente a utilizar las dos funciones (`fspecial` y `imfilter`).

Código 8: Implementación de suavizado en el dominio espacial en MATLAB

```
1 % 4 - Suavizado
2 % Enrique
3 clear;
4
5 img_rgb = imread('suavizado.jpg');
6 cte_gauss = 3;
```

```

7
8 % Filtramos en el espacio
9 img_filt = imgaussfilt(img_rgb, cte_gauss,'FilterSize', 19, 'Padding', 'circular');
10
11 % Representamos ambas imágenes
12 figure
13 subplot(1,2,1),
14 imshow(img_rgb, []),
15 axis off image,
16 title('Imagen original [RGB]')
17
18 subplot(1,2,2)
19 imshow(img_filt, [])
20 title('Imagen filtrada en el dominio del espacio [RGB]')
21

```

Como podemos comprobar en la implementación, los parametros del filtro se aplican de manera equivalente a como lo haríamos con las funciones `fspecial` y `imfilter`. En este caso, hemos probado con diferentes valores de filtrado para obtener un mejor suavizado. Una vez ejecutamos el script `suavizado_esp.m`, obtenemos la siguiente figura comparativa entre la imagen original y la imagen suavizada (ver Figura 13).

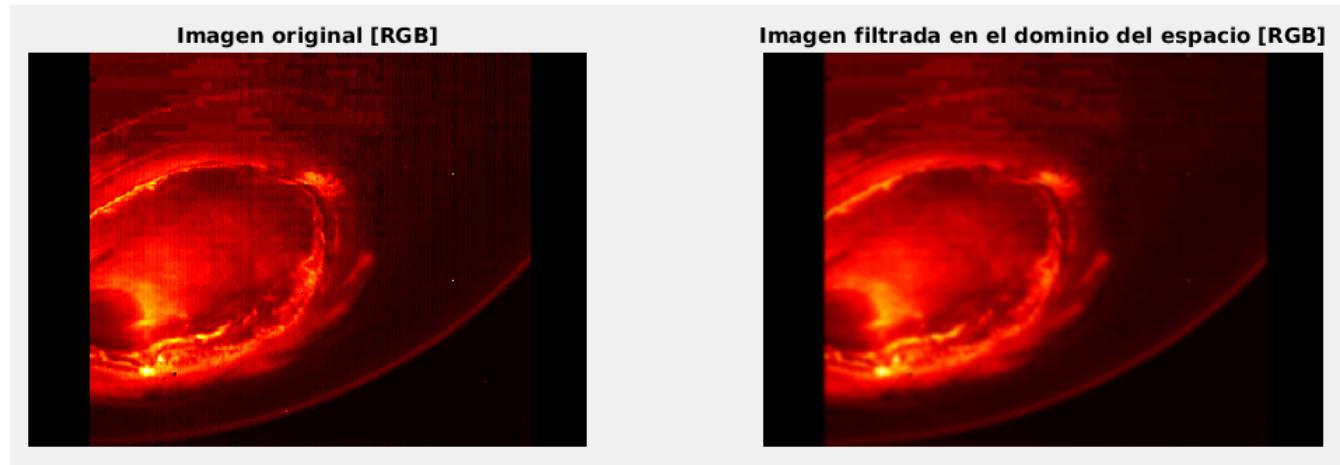


Figura 13: Figura resultante al ejecutar el script `suavizado_esp.m`

Dominio de la frecuencia

En el dominio del espacio también podemos filtrar la imagen, con el objetivo de quitar componentes en frecuencia que nos provoquen esos “cuadrados” en el dominio del espacio. Para ello, utilizamos un filtro tipo gaussiano, pero aplicándolo sobre la imagen en frecuencia.

Código 9: Implementación suavizado en el dominio de la frecuencia en MATLAB

```
1 % 4 - Suavizado
2 % Enrique
3 clear;
4
5 img = im2single(imread('suavizado.jpg'));
6 cte_gauss = 4.5;
7
8 % Representamos imagen original
9 figure,
10 subplot(1,2,1)
11 imshow(img),
12 title('Imagen original [RGB]')
13
14 % Aplicamos filtro gaussiano sobre las dimensiones de la imagen (733 1041)
15 filter = fspecial('gaussian', [733 1041], cte_gauss);
16
17 % Canal R
18 F1 = fftshift(fft2(img,:,:1));
19 mask1 = fftshift(psf2otf(filter,[size(img,:,:1), 1], size(img,:,:1), 2]));
20
21 % Canal G
22 F2 = fftshift(fft2(img,:,:2));
23 mask2 = fftshift(psf2otf(filter,[size(img,:,:3), 1], size(img,:,:3), 2]));
24
25 % Canal B
26 F3 = fftshift(fft2(img,:,:3));
27 mask3 = fftshift(psf2otf(filter,[size(img,:,:3), 1], size(img,:,:3), 2)));
28
29 % Reconstruimos la imagen a color
30 filtered_img(:,:1) = ifft2(ifftshift(F1 .* mask1));
31 filtered_img(:,:2) = ifft2(ifftshift(F2 .* mask2));
32 filtered_img(:,:3) = ifft2(ifftshift(F3 .* mask3));
33
34 % Representamos en el dominio de la frecuencia
35 %F = fft2(img_rgb(:,:,2));
36 %S = fftshift(log(1+abs(F)));
37 %figure
38 %imshow(S, []),
39 %title('Representacion del espectro (log)')
40
41 % Representamos la imagen suavizada
42 subplot(1,2,2)
43 imshow(abs(filtered_img)),
44 axis off image,
45 title('Imagen filtrada en el dominio de la frecuencia [RGB]')
46
```

Para este caso, lo que hacemos es aplicar un filtro a la imagen en frecuencia. Como el filtro solo nos permite aplicarlo sobre un canal a la vez, descomponemos la imagen en los canales R, G y B, para aplicar el filtro y posteriormente reconstruir la imagen. Una vez ejecutamos el script `suavizado_freq.m`, nos aparece una figura comparativa entre la imagen inicial y la imagen suavizada en frecuencia (ver Figura 14).

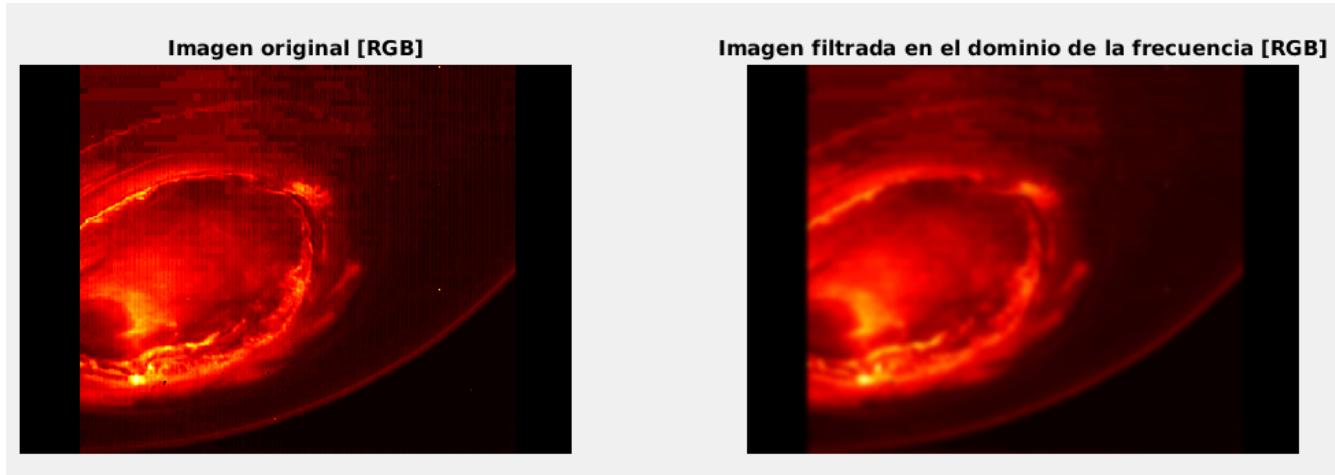


Figura 14: Figura resultante al ejecutar el script `suavizado_freq.m`

Una vez comprobadas las diferentes técnicas, extraemos las siguientes conclusiones:

- Ambas soluciones aportan un suavizado respecto a la imagen inicial.
- Por mucho ajuste que realicemos a los filtros, siempre habrá zonas que nos cueste más eliminar. Por ejemplo, en ambas imágenes suavizadas podemos apreciar restos de “cuadrados” en las zonas suavizadas.
- Podemos aplicar un filtrado de tipo guassiano tanto en espacio como en frecuencia.

5 Realzado

En este ejercicio tenemos que aplicar técnicas de realzado para obtener más detalles en la imagen 15. Para ello, aplicaremos diferentes filtros, con el objetivo de resaltar los cambios entre las diferentes zonas de color de la imagen.



Figura 15: Imagen inicial realzado

Para realizar 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 10: Implementación de filtros para el realzado en MATLAB

```
1 % 5 - Realzado
2 % Enrique
3 clear;
4
5 img_rgb = imread('realzado.jpg');
6
7 % Preajuste expansion margen dinamico
8 img_adj = imadjust(img_rgb, stretchlim(img_rgb), [ ]);
9
10 % Caso uno, prewitt
11 filter = fspecial('prewitt');
12 img_filtered = imfilter(img_adj, filter, 'conv');
13 %img_rgb_1 = imadd(img_adj, img_filtered);
14 img_rgb_1 = img_adj - img_filtered;
15
```

```
16 % Caso dos, sobel
17 filter = fspecial('sobel');
18 img_filtered = imfilter(img_adj, filter, 'conv');
19 %img_rgb_2 = imadd(img_adj, img_filtered);
20 img_rgb_2 = img_adj - img_filtered;
21
22 figure
23 subplot(2,2,1)
24 imshow(img_rgb, [])
25 title('Imagen original')
26
27 subplot(2,2,2)
28 imshow(img_adj, [])
29 title('Imagen preajustada')
30
31 subplot(2,2,3)
32 imshow(img_rgb_1, [])
33 title('Imagen realzada (prewitt)')
34
35 subplot(2,2,4)
36 imshow(img_rgb_2, [])
37 title('Imagen realzada (sobel)')
38
39
```

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).

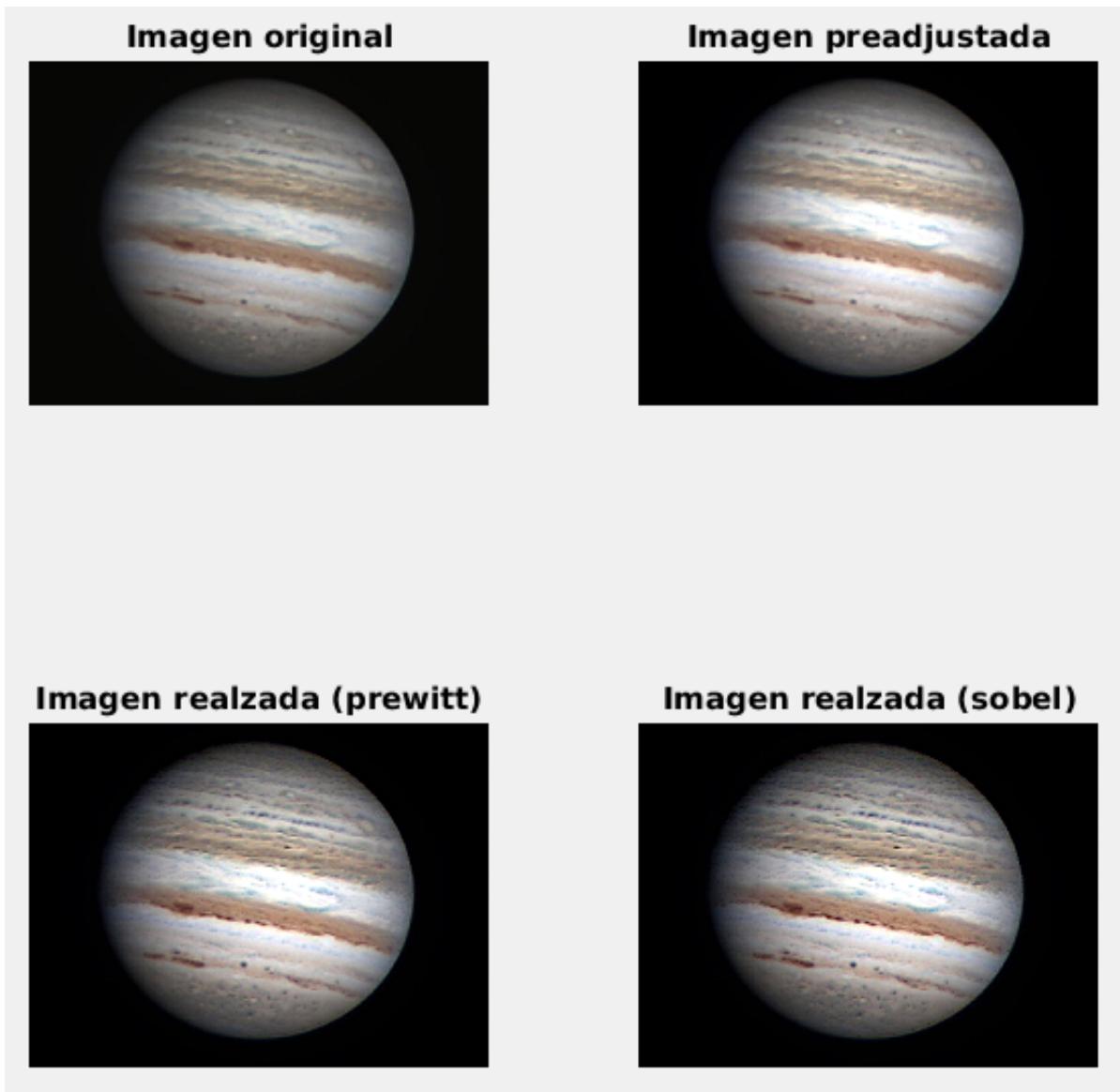


Figura 16: Imagen inicial ruido

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 17. 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” o impulsivo.



Figura 17: Imagen inicial ruido

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

Código 11: Implementación filtro mediana para reducir ruido en MATLAB

```
1 % 6 - Ruido Sal y Pimienta
2 % Enrique
3 clear;
4
5 img_rgb = 'ruido.jpg';
6 neighbor = 4;
7 times = 5;
```

```

9 img = imread(img_rgb);
10 img = mat2gray(img, [0 255]);
11 img = rgb2gray(img);
12
13 figure
14 subplot(1,2,1)
15 imshow(img, [])
16 title('Imagen original [RGB]')
17
18 % Aplicamos filtro mediana
19 img_median = img;
20 for n=1:times,
21 img_median = medfilt2(img_median, [1 1]*neighorn);
22 end
23
24 % Representamos resultado
25 subplot(1,2,2)
26 imshow(img_median, []),
27 axis off image,
28 title(['Imagen eliminado el ruido usando filtro mediana con N=' num2str(neighorn)
29 ' T=' num2str(times) ])

```

Una vez ejecutado el código, tendremos como salida una comparativa entre la imagen inicial (ver figura 17) y imagen resultante tras eliminar el ruido “sal y pimienta”. Podemos comprobar como se ha eliminado prácticamente la totalidad del ruido.

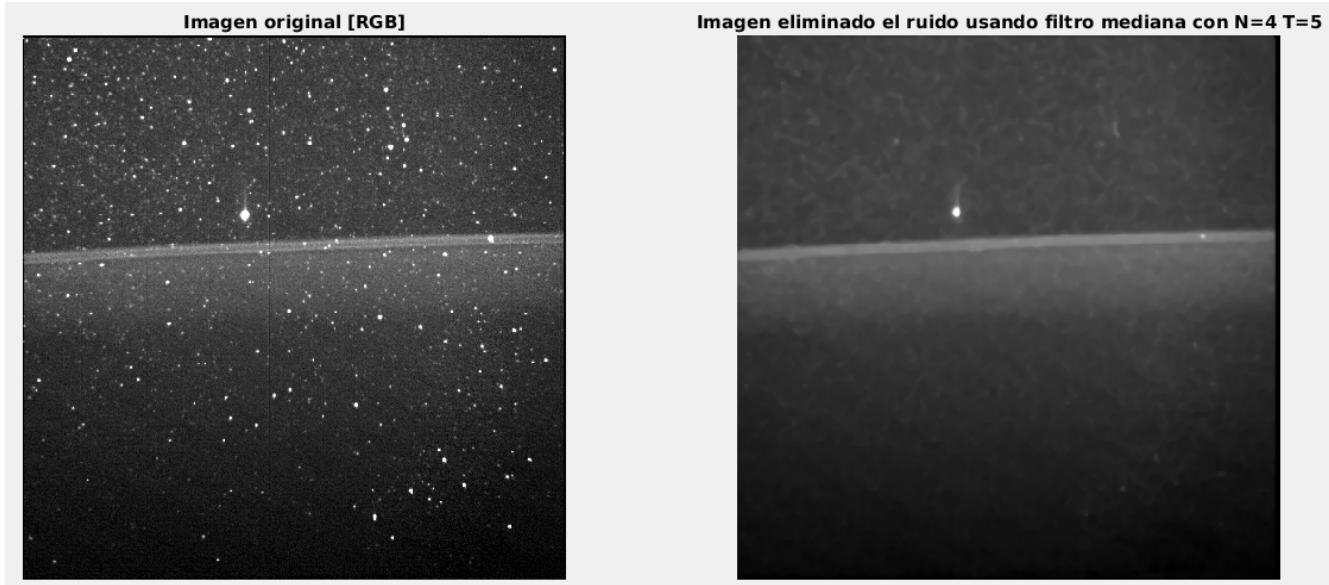


Figura 18: Figura de salida al ejecutar script de MATLAB `ruido.m`

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 19.

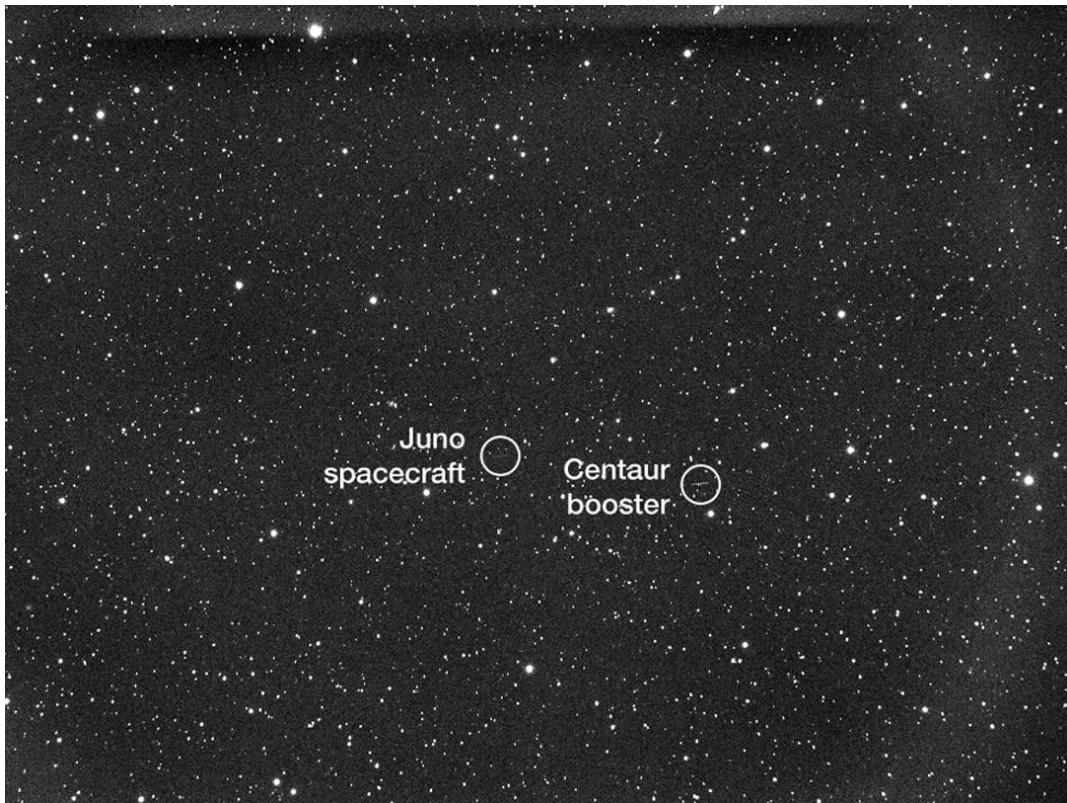


Figura 19: Imagen inicial de patrones

Utilizando la imagen 19, 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 12: Implementación detección de patrones en MATLAB

```
1 % 7 - Patrones
2 % Enrique
3 % ref: https://es.mathworks.com/matlabcentral/answers/90094-how-to-convolved-two-image
4 clear;
5
6 img_rgb = 'patrones.jpg';
7 threshold_corre = 0.55;
8 threshold_stars = 0.85;
9
10 img_rgb = single(imread(img_rgb));
11 img_rgb = mat2gray(img_rgb,[0 255]);
12 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 ;
16 lambda_autocovar = @(A,B) conv2(A.*A, ones(size(B)), 'valid')-conv2(A, ones(size(B))
17 ), '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(
45     selected_pattern, selected_pattern));
46 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 );
49 coef_pearson = padarray(coef_pearson, ceil((size(selected_pattern)-1)/2), 0, 'pre')
50 ;
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

```

```

62 subplot(1,2,2),
63 mesh(coef_pearson),
64 axis square,
65 set(gca,'YDir','reverse'),
66 title('Ocurrencias de la deteccion del patron (plot en elevacion)')
67
68 % Display number of ocurrences on coef pearson
69 number_stars = sum(sum(coef_pearson > threshold_stars));
70 disp(['Numero de estrellas detectadas: ' num2str(number_stars)]);
71

```

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 20). 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 21)

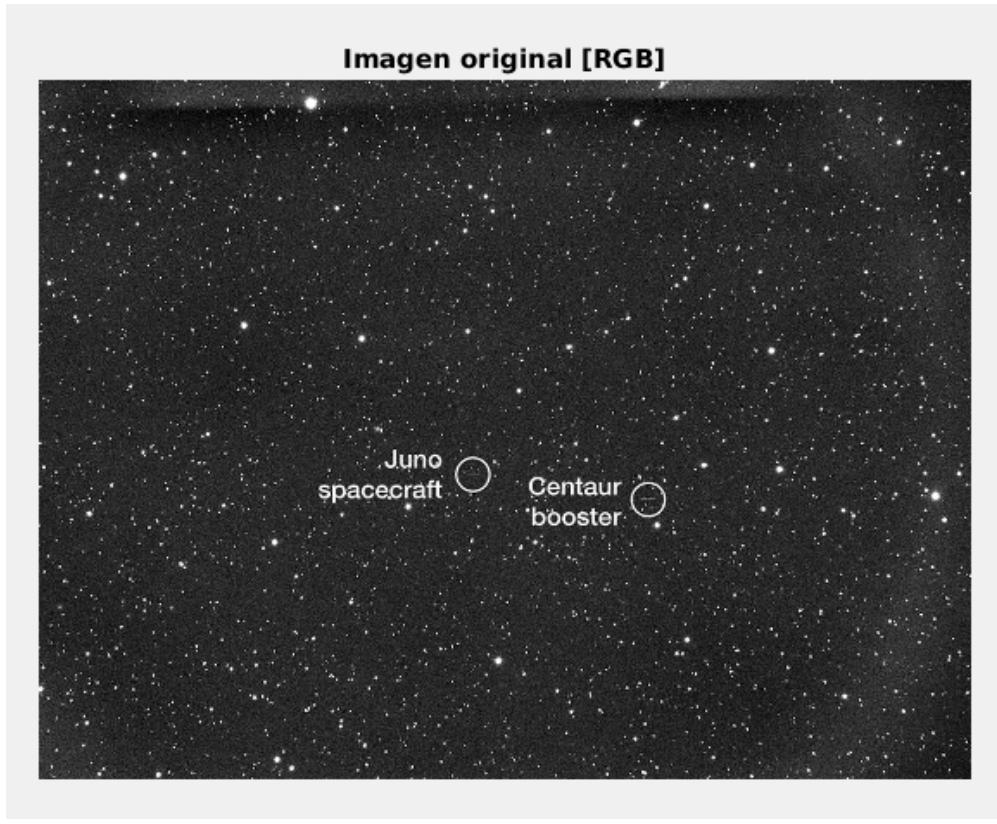


Figura 20: Imagen para seleccionar el patrón a buscar

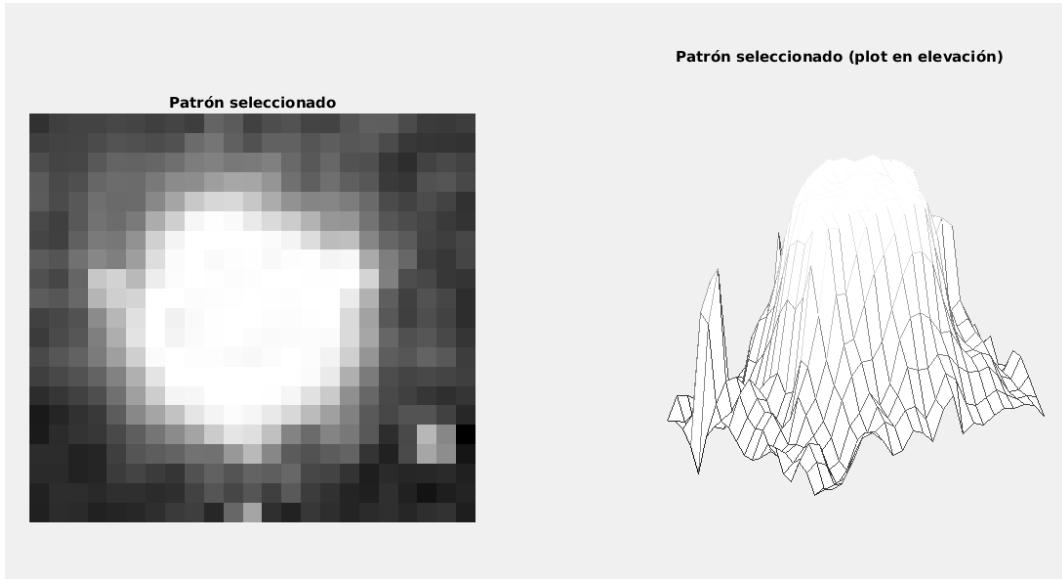


Figura 21: Figura resultante una vez seleccionamos el patrón

Una vez seleccionamos el patrón, podemos comprobar que además de la figura de la imagen 21, 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 22).

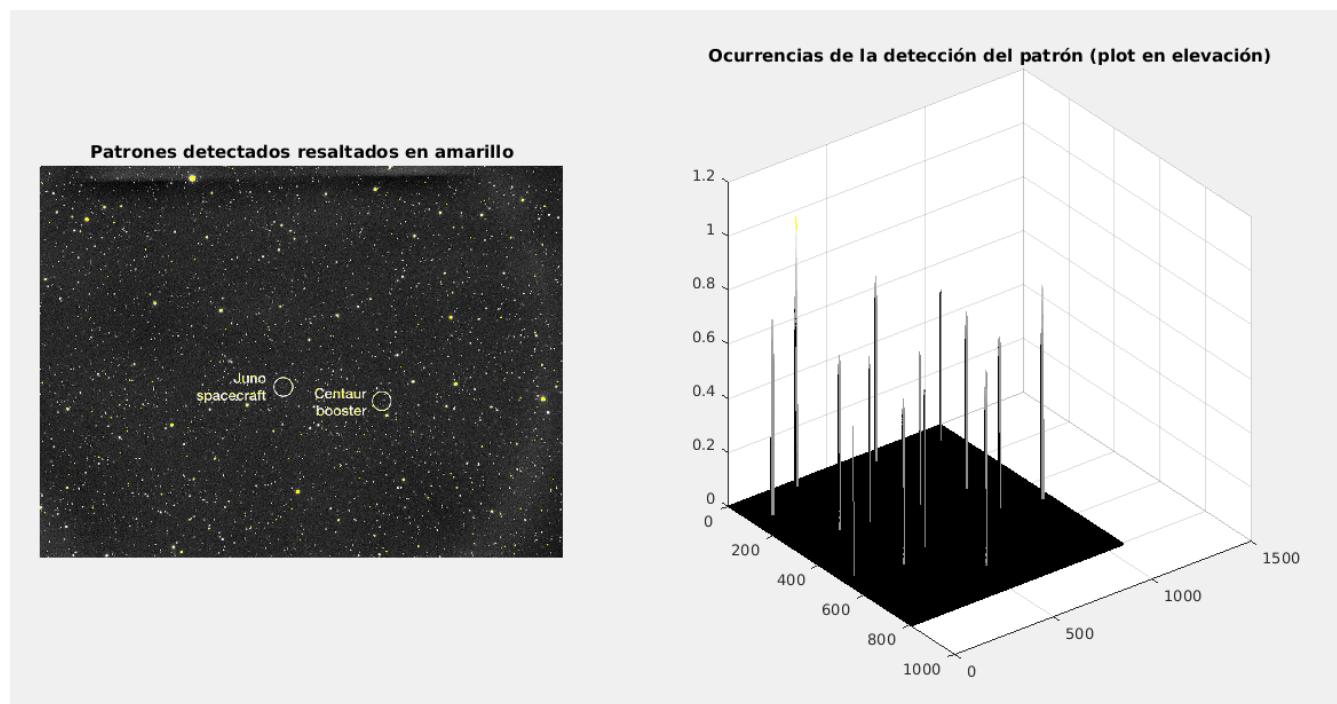


Figura 22: Figura resultante una vez seleccionamos el patrón

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 23).

```
>> 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 23: 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 24), con esto pretendemos ver detalles, que en el caso de ver la imagen en niveles de gris no veríamos.



Figura 24: 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 13: Implementación de pseudocoloración en MATLAB

```
1 % 8 - Pseudocoloracion por nivel de gris
2 % Enrique
3
4 img_rgb = 'pseudocoloracion.jpg';
5 gray_lvl = 64;
6
7 img = imread(img_rgb);
8 img = mat2gray(img, [0 255]);
9 img = rgb2gray(img);
10
11 % Utilizamos 64 niveles de gris
12 img_colored = grayslice(img, gray_lvl);
13
14
```

```

15 % Representamos cada una de las coloraciones
16 figure
17 subplot(2,3,1),
18 subimage(img);
19 axis off image,
20 title('Imagen original en escala de gris')
21
22 subplot(2,3,2),
23 subimage(img_colored, copper(gray_lvl));
24 axis off image,
25 title('Pseudocoloracion (copper)')
26
27 subplot(2,3,3),
28 subimage(img_colored, pink(gray_lvl));
29 axis off image,
30 title('Pseudocoloracion (pink)')
31
32 subplot(2,3,4),
33 subimage(img_colored, bone(gray_lvl));
34 axis off image,
35 title('Pseudocoloracion (bone)')
36
37 subplot(2,3,5),
38 subimage(img_colored, hot(gray_lvl));
39 axis off image,
40 title('Pseudocoloracion (hot)')
41
42 subplot(2,3,6),
43 subimage(img_colored, parula(gray_lvl));
44 axis off image,
45 title('Pseudocoloracion (parula)')
46

```

Una vez ejecutado el código, tendremos como salida una comparativa entre la imagen inicial (ver Figura 24) y las diferentes imágenes resultantes tras aplicar pseudocoloración (ver Figura 25):

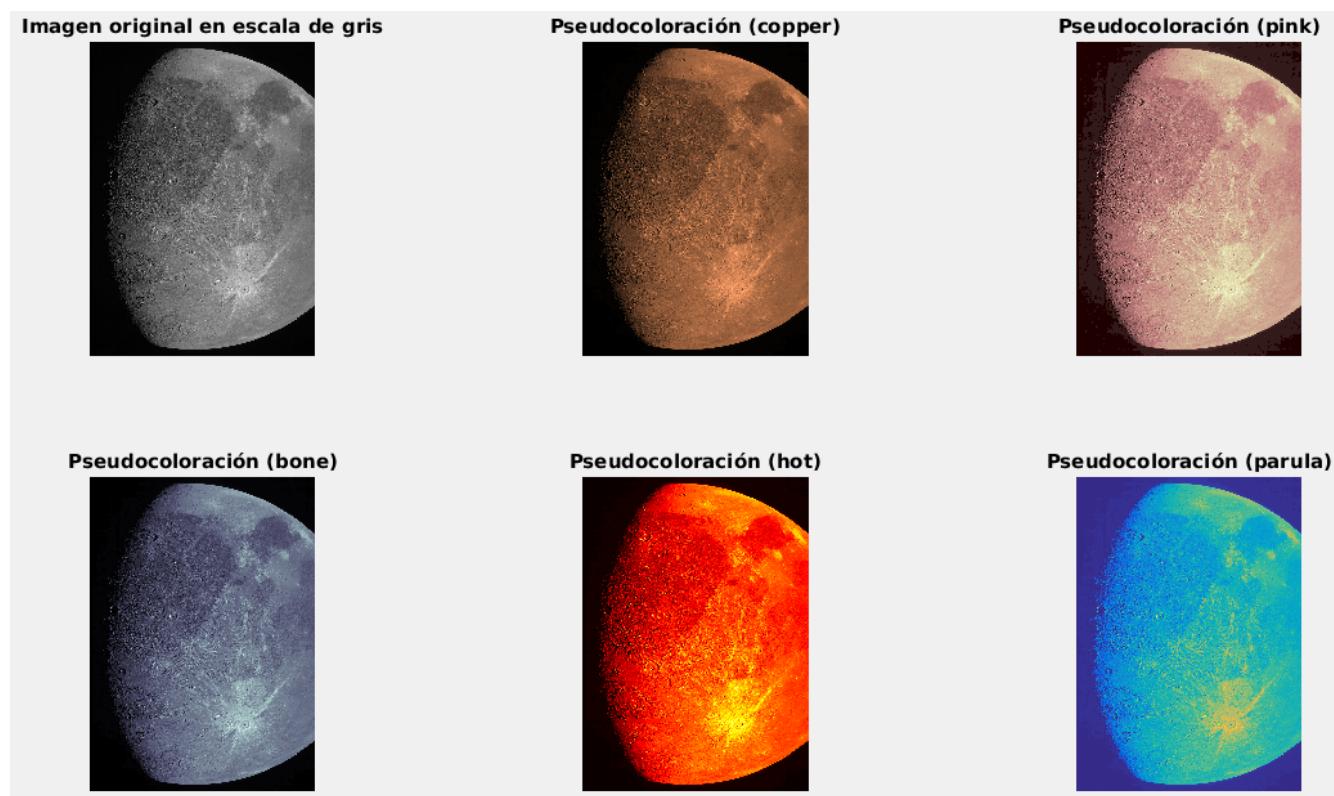


Figura 25: Figura de salida al ejecutar script de MATLAB `pseudocoloracion.m`

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.