

Procesamiento digital de imágenes

Toolbox Básico

Versión 1, Septiembre 2022

Para uso con:

Python V3.6

Guía de Usuario

Ing. Juliana Gómez Osorio

Msc. Francisco Alejandro Medina

Ph.D Jimmy Alexander Cortés

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

PEREIRA

2022

INDICE GENERAL.

Introducción	1
1. Función imread	2
1.1 Uso	2
1.2 Ejemplo	2
1.3 Resultado	2
2. Función imshow	4
2.1 Uso	4
2.2 Ejemplo	4
2.3 Resultado	5
3. Función rgb2gray	7
3.1 Uso:	7
3.2 Ejemplo:	7
3.3 Resultado	8
4. Función imcrop	10
4.1. Uso	10
4.2. Ejemplo	11
4.3. Resultado	12
5. Función imresize	14
5.1. Uso	14
5.2. Ejemplo	14
5.3. Resultado	15
6. Función imtranslate	19
6.1. Uso	19
6.2. Ejemplo	19
6.3. Resultado	20
7. Función imhist	24
7.1. Uso	24
7.2. Ejemplo	24
7.3. Resultado	25
8. Función imhist2	27
8.2. Ejemplo	27

8.3. Resultado	28
9. Función mean2	30
9.1. Uso	30
9.2. Ejemplo	30
9.3. Resultado	30
10. Función std2.....	33
10.1. Uso	33
10.2. Ejemplo	33
10.3. Resultado	33
11. Función imabsdiff	35
11.1. Uso	35
11.2. Ejemplo	35
11.3. Resultado	36
12. Función imadd	39
12.1. Uso	39
12.2. Ejemplo	39
12.3. Resultado	40
13. Función imdivide	43
13.1. Uso	43
13.2. Ejemplo	44
13.3. Resultado	44
14. Función immultiply.....	47
14.1. Uso	47
14.2. Ejemplo	48
14.3. Resultado	48
15. Función imsubtract.....	51
15.1. Uso	51
15.2. Ejemplo	51
15.3. Resultado	52
16. Función imadjust	55
16.1. Uso	55
16.2. Ejemplo	56
16.3. Resultado	56

17. Función stretchlim	59
17.1. Uso	59
17.2. Ejemplo	59
17.3 Resultado:.....	59
18. Función imnoise.....	62
18.1. Uso	62
18.2 Ejemplo.....	62
18.2. Resultado.....	63
19. Función medfilt2.....	67
19.1. Uso	67
19.2. Ejemplo	67
19.3. Resultado	68
20. Función ordfilt2	71
20.1. Uso	71
20.2. Ejemplo	71
20.3. Resultado	72
21. Función fspecial	75
21.1. Uso	75
21.2. Ejemplo	76
21.3. Resultado	76
22. Función strel	80
22.1. Uso	80
22.2. Ejemplo:.....	80
22.3. Resultado	80
23. Función graythresh.....	86
23.1. Uso	86
23.2. Ejemplo	86
23.3. Resultado	87
24. Función im2bw	90
24.1. Uso	90
24.2. Ejemplo:.....	90
24.3. Resultado	91
25. Función histeq.....	93

25.1. Uso	93
25.2. Ejemplo	93
25.3. Resultado	93
26. Función mat2gray	96
26.1 Uso	96
26.2 Ejemplo:	96
26.3 Resultado	97
27. Función imwrite.....	100
27.1 Uso	100
27.2 Ejemplo:	100
27.3 Resultado:.....	101
28. Función imfilter.....	103
28.1 Uso:	103
28.2 Ejemplo:	103
28.3 Resultado:.....	104
29. Función Hough.....	107
29.1 Uso	107
29.2 Ejemplo:	108
29.3 Resultado:.....	108
30. Función imerode.....	111
30.1 Uso	111
30.2 Ejemplo:	111
30.3 Resultado:.....	112
31. Función imdilate.....	116
31.1 Uso.	116
31.2 Ejemplo:	116
31.3 Resultado:.....	117
32. Función imclose.....	120
32.1 Uso	120
32.2 Ejemplo:	120
32.3 Resultado:.....	121
33. Función fft2.....	124
33.1 Uso	124

33.2 Resultado:	124
34. Función fftshift	126
34.1 Uso	126
34.2 Resultado:	126
35. Función iift2	127
35.1 Uso	127
35.2 Resultado:	128
36. Ejemplos	129
36.1 Filtro pasa bajo ideal.	129
36.2 Filtro pasa bajo Butterworth.	131
36.3 Filtro pasa bajo Gaussiano.	133
36.4 Filtro pasa alto ideal.	135
36.5 Filtro pasa alto Butterworth.	137
36.6 Filtro pasa alto Gausiano.	139
36.7 Filtro eliminador de banda.	141
36.8 Filtro pasa banda.	143
	145
37. Función bwperim	145
37.1 Uso	145
37.2 Ejemplo:	146
37.3 Resultado.	146
38. Función rgb2HSV.	149
38.1 Uso.	149
38.2 Ejemplo:	149
38.3 Resultado:	150
39. Función HSV2RGB.	154
39.1 Uso.	154
39.2 Ejemplo:	154
39.3 Resultado:	155
40. Función RGB2XYZ.	160
40.1 Uso.	160
40.2 Ejemplo:	160
40.3 Resultado.	161

41. Función xyz2rgb.....	165
41.1 Uso.	165
41.2 Ejemplo:	165
41.3 Resultado.....	166
42. Función xyz2lab	170
42.1 Uso.	170
42.2 Ejemplo:	170
42.3 Resultado.....	171
43. Función lab2xyz.....	175
43.1 Uso.	175
43.2 Ejemplo:	175
43.3 Resultado.....	176
44. Función rgb2lab.....	180
44.1 Uso.	180
44.2 Ejemplo:	180
44.3 Resultado.....	181
45. Función lab2rgb.....	183
45.1 Uso.	183
45.2 Ejemplo:	183
45.3 Resultado.....	183
BIBLIOGRAFÍA	186
REFERENCIAS.....	187

ÍNDICE DE FIGURAS.

Fig. 1.1 Diagrama de flujo de la función.....	3
Fig. 1.2 Función en pseudolenguaje pse	3
Fig. 1.3 Algoritmo en Python.....	4
Fig. 2.1. Diagrama de flujo de la función.....	5
Fig. 2.2. Función en pseudolenguaje pse	6
Fig. 2.3. Algoritmo en Python.....	6
Fig. 2.4. Imagen generada en Python.....	7
Fig. 3.1. Diagrama de flujo de la función.....	8
Fig. 3.2. Función en pseudolenguaje pse	9
Fig. 3.3. Algoritmo en Python.....	9
Fig. 3.4. Conversión de RGB a escala de grises en Python	10
Fig. 4.1. Diagrama de flujo de la función.....	12
Fig. 4.2. Función en pseudolenguaje pse	12
Fig. 4.3. Algoritmo en Python.....	13
Fig. 4.4. Imagen cortada en Python.....	13
Fig. 5.1. Diagrama de flujo de la función.....	16
Fig. 5.2. Función en pseudolenguaje pse	17
Fig. 5.3. Algoritmo en Python.....	18
Fig. 5.4 Imagen reducida.	18
Fig. 6.1. Diagrama de flujo de la función.....	21
Fig. 6.2. Función en pseudolenguaje pse	22
Fig. 6.3. Algoritmo en python.....	23
Fig. 6.4. Imagen trasladada.	23
Fig. 7.1. Diagrama de flujo de la función.....	25
Fig. 7.2. Función en pseudolenguaje pse	26
Fig. 7.3. Algoritmo en Python.....	26
Fig. 7.4 Histograma de la imagen.	26
Fig. 8.1 Diagrama de flujo de la función.....	28
Fig. 8.2 Función en pseudolenguaje pse	29

Fig. 8.3. Algoritmo en Python.....	29
Fig. 9.1. Diagrama de flujo de la función.....	31
Fig. 9.2. Función en pseudolenguaje pse	32
Fig. 9.3 Algoritmo en Python.....	32
Fig. 10.1. Diagrama de flujo de la función.....	34
Fig. 10.2. Función en pseudolenguaje pse	34
Fig. 10.3 Algoritmo en Pyhton.....	35
Fig. 11.1. Diagrama de flujo de la función.....	37
Fig. 11.2. Función en pseudolenguaje pse	38
Fig. 11.3. Algoritmo en Python.....	38
Fig. 11.4. Diferencia absoluta en Python	39
Fig. 12.1. Diagrama de flujo de la función.....	41
Fig. 12.2. Función en pseudolenguaje pse	42
Fig. 12.3 .Algoritmo en Python.....	42
Fig. 12.4. Suma de dos imágenes en Python.....	43
Fig. 13.1. Diagrama de flujo de la función.....	45
Fig. 13.2. Función en pseudolenguaje pse	46
Fig. 13.3. Algoritmo en Python.....	46
Fig. 13.4. División de imágenes en Python.....	47
Fig. 14.1. Diagrama de la función.....	49
Fig. 14.2. Función en pseudolenguaje pse	50
Fig. 14.3 Algoritmo en python.....	50
Fig. 14.4 Multiplicación de imagen y multiplicación de imagen por constante.....	51
Fig. 15.1. Diagrama de flujo de la función.....	53
Fig. 15.2. Función en pseudolenguaje pse	54
Fig. 15.3. Algoritmo en Python.....	54
Fig. 15.4. Resta de dos imágenes en Python.....	55
Fig. 16.1. Diagrama de flujo de la función.....	57
Fig. 16.2. Función en Pseudolenguaje pse	57
Fig. 16.3. Algoritmo en Python.....	58
Fig. 16.4. Imagen ajustada en Python.....	58
Fig. 17.1. Diagrama de flujo de la función.....	60
Fig. 17.2. Función en pseudolenguaje pse	61

Fig. 17.3 Algoritmo en Python.....	61
Fig. 18.1. Diagrama de flujo de la función.....	64
Fig. 18.2. Función en pseudolenguaje pse	65
Fig. 18.3. Algoritmo en Python.....	66
Fig. 18.4. Imagen con ruido Salt & pepper , Gaussian y speckle en Python.	66
Fig. 19.1. Diagrama de flujo de la función.....	69
Fig. 19.2. Función en pseudolenguaje pse	70
Fig. 19.3. Algoritmo en Python.....	70
Fig. 19.4 Imagen filtrada en Python.	71
Fig. 20.1. Diagrama de flujo de la función.....	73
Fig. 20.2. Función en pseudolenguaje pse	74
Fig. 20.3 Algoritmo en Python.....	74
Fig. 20.4 Imagen con filtrado estadístico.	75
Fig. 21.1. Diagrama de flujo de la función.....	77
Fig. 21.2. Función en pseudolenguaje pse.	79
Fig. 21.3 Algoritmo en Python.....	79
Fig. 22.1. Diagrama de flujo de la función.....	82
Fig. 22.2. Función en pseudolenguaje pse	84
Fig. 22.3. Algoritmo en Python.....	85
Fig. 23.1. Diagrama de flujo de la función.....	88
Fig. 23.2. Función en pseudolenguaje pse	89
Fig. 23.3 Algoritmo en Python.....	89
Fig. 24.1. Diagrama de flujo de la función.....	91
Fig. 24.2 Función en pseudolenguaje pse	91
Fig. 24.3 Algoritmo en Python.....	92
Fig. 24.4 Imagen binarizada	92
Fig. 25.1. Diagrama de flujo de la función.....	94
Fig. 25.2. Función en pseudolenguaje pse.	95
Fig. 25.3. Algoritmo en Python.....	95
Fig. 25.4 Ecualización en Python.	96
Fig. 26.1. Diagrama de flujo de la función.....	98
Fig. 26.2. Función en pseudolenguaje pse.	99
Fig. 26.3 Algoritmo en Python.....	99

Fig. 26.4. Imagen creada mat2gray en Python.....	100
Fig. 27.1. Diagrama de flujo de la función.....	101
Fig. 27.2. Función en pseudolenguaje pse	102
Fig. 27.3. Algoritmo en Python.....	102
.....	102
Fig. 27.4. Imágenes creadas imwrite Python.	102
Fig. 28.1. Diagrama de flujo de la función.....	105
Fig. 28.2. Función en pseudolenguaje pse	106
Fig. 28.3. Algoritmo en Python.....	106
Fig. 28.4. Filtrado de correlación en Python.	107
Fig. 29.1. Diagrama de flujo de la función.....	109
Fig. 29.2. Función en pseudolenguaje pse	110
Fig. 29.3. Algoritmo en Python.....	110
Fig. 30.1. Diagrama de flujo de la función.....	113
Fig. 30.2. Función en pseudolenguaje pse.	114
Fig. 30.3. Algoritmo en Python.....	115
Fig. 30.4. Imágenes creadas imerode en Python.	115
Fig. 31.1. Diagrama de flujo de la función.....	118
Fig. 31.2. Función en pseudolenguaje pse	119
Fig. 31.3 Algoritmo en Python.....	119
Fig.31.4. Imágenes generadas imdilate en Python.	120
Fig. 32.1. Diagrama de flujo de la función.....	122
Fig. 32.2. Función en pseudolenguaje pse	123
Fig. 32.3. Algoritmo en Python.....	123
Fig. 32.4. Imágenes generadas imclose en Python.	124
Fig. 33.1. Diagrama de flujo de la función.....	125
Fig. 33.2. Función en pseudolenguaje pse.	125
Fig. 33.3. Algoritmo en Python.....	125
Fig. 34.1. Diagrama de flujo de la función.....	126
Fig. 34.2. Función en pseudolenguaje pse.	127
Fig. 34.3. Algoritmo en Python.....	127
Fig. 35.1. Diagrama de flujo de función.	128
Fig. 35.2. Función en pseudolenguaje pse.	129

Fig. 35.3. Algoritmo en Python.....	129
Fig. 36.1. Filtro pasa bajo ideal.....	131
Fig. 36.2. Filtro pasa bajo ideal.....	133
Fig. 36.3. Filtro pasa bajo Gaussiano.....	135
Fig. 36.4. Filtro pasa alto ideal.	137
Fig. 36.5.: Filtro pasa alto Butterworth.	139
Fig. 36.6. Filtro pasa alto Gaussiano.....	141
Fig. 36.7. Filtro eliminador de banda.	143
Fig.36.8 Filtro eliminador de banda.	145
Fig. 37.1 Diagrama de flujo de la función.....	147
Fig. 37.2. Función en pseudolenguaje pse.	148
Fig.37.3. Algoritmo en Python.....	148
Fig.37.4. Imágenes generadas de bwperim en Python.	149
Fig 38.1. Diagrama de flujo de la función.....	151
Fig.38.1 Diagrama de flujo de la función.....	152
Fig.38.2. Función en pseudolenguaje pse.	153
Fig. 38.3. Algoritmo en Python.....	153
Fig.38.4. Conversión de RGB a HSV en Python.....	154
Fig.39.1.Diagrama de flujo de la función.	156
Fig.39.2. Función en pseudolenguaje pse.	158
Fig.39.3. Algoritmo en Python.....	159
Fig.40.1. Diagrama de flujo de la función.....	162
Fig.40.2. Función en pseudolenguaje pse.	163
Fig.40.3. Agoritmo en Python.	164
Fig 40.4. Conversión de RGB a XYZ en Python.	165
Fig 41.1.Diagrama de flujo de la función.....	167
Fig.41.2 Función en pseudolenguaje pse.	168
Fig.41.3.Algoritmo en Python.	169
Fig.41.4. Conversión de XYZ a RGB en Python.	170
Fig.42.1.Diagrama d flujo de la función.	172
Fig.42.2 Función en pseudolenguaje pse.	173
Fig.42.3 Algoritmo en Python.....	174
Fig.42.4. Conversión de XYZ a LAB en Python.	175

Fig.43.1. Diagrama de flujo de la función.....	177
Fig.43.2.Funcióen pseudolenguaje pse.....	178
Fig.43.3 Algoritmo en Python.....	179
Fig.43.4 Conversión de LAB a XYZ en Python.....	180
Fig.44.1.Diagrama de flujo de la función.	181
Fig.44.2 Función en pseudolenguaje pse.....	182
Fig.44.3. Algoritmo en Python.....	182
Fig.44.4 Conversión RGB a LAB en Python.....	183
Fig.45.1 Diagrama de flujo de la Función.	184
Fig.45.2. Función en pseudolenguaje pse.....	185
Fig.45.3.Algoritmo en Python.	185
Fig.45.4 Conversión de LAB a RGB en Python.....	185

Introducción

El Toolbox o Caja de Herramientas para el procesamiento digital de imágenes, es un conjunto de funciones básicas creadas originalmente para el sistema de cómputo numérico Matlab v2016a ,FreeMat v4 y Octave v4.0.2, y posteriormente migradas al lenguaje de programación Python v3.6 dando uso de librerías básicas para operaciones matemáticas como **Numpy** para crear vectores y matrices grandes multidimensionales, y **Matplotlib** para la creación de gráficos a partir de datos contenidos en listas o arrays . La caja de herramientas soporta una amplia gama de operaciones de procesamiento de imágenes, entre ellas:

1. Funciones de Entrada/Salida.
2. Muestra de imagen.
3. Transformaciones espaciales.
4. Valores de pixeles y estadísticas.
5. Aritmética de imagen.
6. Mejora de imagen.
7. Filtrado lineal.
8. Operaciones morfológicas.
9. Estructuración del elemento, creación y manipulación.
10. Conversiones de espacio de color.
11. Tipos de imágenes y tipos de conversiones.
12. Filtrado en el dominio de la frecuencia.

Matlab v2016a tiene una lista de procesamiento de imágenes amplio, pero este no es libre y solo puede usarse sobre su plataforma de desarrollo. El presente trabajo logra desarrollar una serie de funciones (49) en las cuales en su mayoría mantienen los nombres similares y así mismo con una misma entrada y salida para mayor flexibilidad implementadas en Python.

1. Función imread

1.1 Uso

imread Lee una imagen desde un archivo específico dado.

La sintaxis para su uso es:

- `imread('Imagen')`

Retorna “A”: Contiene las matrices devueltas con los datos de la imagen.

Donde “Imagen” es el nombre del archivo a leer. El uso de comillas simples (`''`) delimita el nombre del archivo.

1.2 Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imread*.

1. Se lee una imagen llamada “cartagena.jpg”, se usa la instrucción *imread*.

- `A = plt.imread('cartagena.jpg')`

1.3 Resultado

```
[[ 94  94  94 ... 223 222 222]
 [ 94  94  94 ... 236 241 241]
 [ 94  94  94 ... 251 252 252]
 ...
 [ 10   9   7 ... 190 189 189]
 [  9   8   8 ... 189 190 190]
 [  8   8  10 ... 189 189 189]]
```

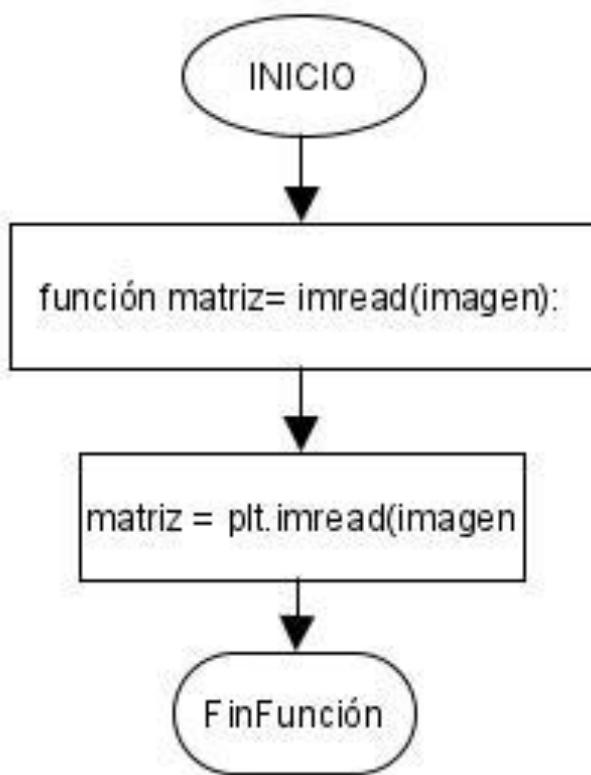


Fig. 1.1 Diagrama de flujo de la función.

```

1 //Nombre: imread
2 //Parámetros: Imagen es la imagen a leer.
3 //Devuelve: Matriz con los datos de la imagen.
4
5 Algoritmo imread
6
7 Función matriz=imread(imagen)
8
9     matriz=imread (imagen)
10
11 FinFuncion
12
13 FinAlgoritmo

```

Fig. 1.2 Función en pseudolenguaje pse

```
def imread(imagen):  
  
    matriz = plt.imread(imagen)  
  
    return matriz
```

Fig. 1.3 Algoritmo en Python.

2. Función imshow

2.1 Uso

imshow Muestra la imagen en escala de grises, su imagen de entrada está en modelo RGB.

La sintaxis para su uso es:

- *imshow(I)*

Donde “I” es la imagen a leer.

2.2 Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imshow*.

1. Para mostrar una imagen, se lee la imagen con la instrucción *imread*, se almacena en una variable “A” y se muestra la imagen usando la instrucción *imshow*.

- A = *imread('cartagena.jpg')*
- *imshow(A)*

2.3 Resultado

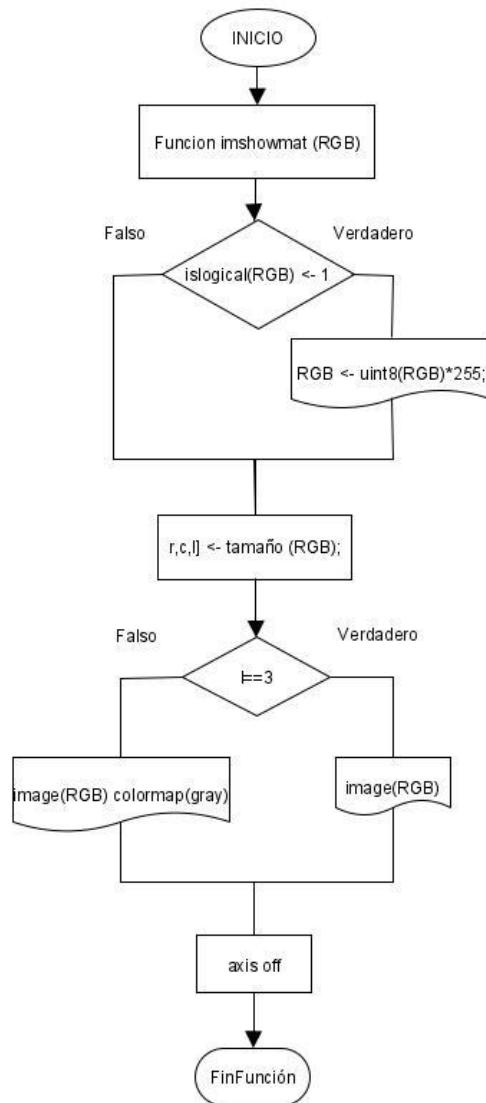
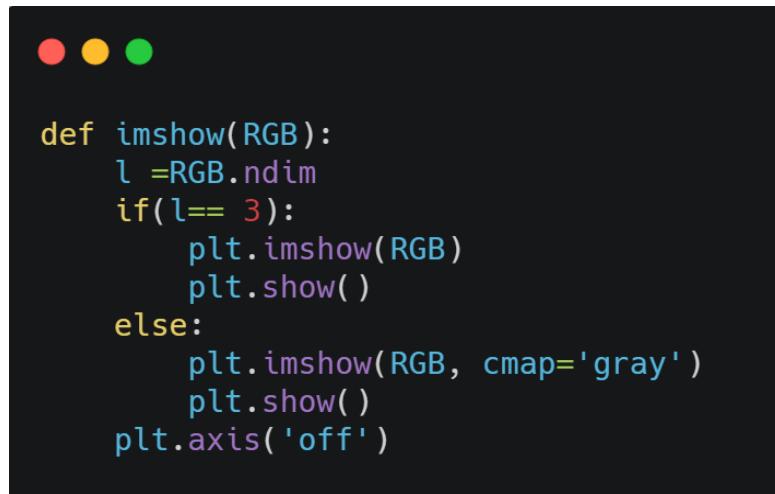


Fig. 2.1. Diagrama de flujo de la función.

```

1 //Nombre: imshowat
2 //Parámetros: donde    RGB (color verdadero) o binarizada.
3 // Devuelve, muestra la imagegn RGB
4 Algoritmo imshowat
5     Funcion imshowat (RGB)
6     ...
7     Si  islogical(RGB) <- 1
8         RGB <- uint8(RGB)*255;
9     FinSi
10
11    [r,c,l] <- tamaño (RGB);      //Muestra el tamaño de la imagen donde l es el numero de capas de la imagen
12
13    Si  l==3
14        image(RGB)           //Muestra la imagen en color verdadero
15    SiNo
16        image(RGB)
17        colormap(gray)
18    FinSi
19
20    axis off
21 FinFuncion
22 FinAlgoritmo
23
24 //NOTA:
25
26 //colormap: Ver y establecer el mapa de colores actual.
27 //islogical: Determina si la entrada es una matriz lógica
28 //axis off: No se muestra las lineas y el fondo de los ejes.|
```

Fig. 2.2. Función en pseudolenguaje pse



```

def imshow(RGB):
    l =RGB.ndim
    if(l== 3):
        plt.imshow(RGB)
        plt.show()
    else:
        plt.imshow(RGB, cmap='gray')
        plt.show()
    plt.axis('off')
```

Fig. 2.3. Algoritmo en Python



Fig. 2.4. Imagen generada en Python¹

3. Función rgb2gray

3.1 Uso:

rgb2gray Cambia una imagen en modelo RGB a imagen en escala de grises.

La sintaxis para su uso es:

- `rgb2gray(I)`

Donde “I” es la imagen en modelo RGB.

Retorna “gris”: Imagen en escala de grises.

3.2 Ejemplo:

¹ Imagen tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *rgb2gray*.

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en la variable “A”.

- A = imread ('herramientas.jpg')

2. Se cambia la imagen a escala de grises y se almacena en la variable “gris”.

- gris = rgb2gray (A)
- imshow (gris)

3.3 Resultado

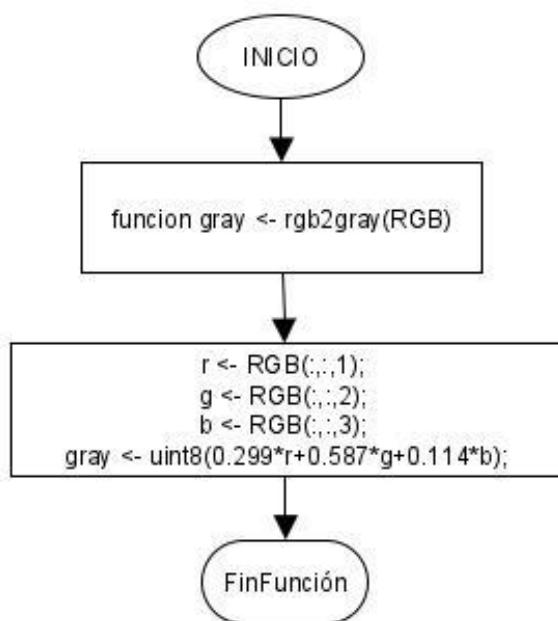


Fig. 3.1. Diagrama de flujo de la función.

```

1 //Nombre: rgb2gray
2 // Parámetros: Donde RGB es la imagen a color
3 //           Y g es la imagen en escala de grises.
4 // Devuelve conversión de una imagen RGB a escala de grises
5 Algoritmo rgb2gray
6 función gray <- rgb2gray(RGB)
7     r <- RGB(:,:,1);
8     g <- RGB(:,:,2);
9     b <- RGB(:,:,3);
10    gray <- uint8(0.299*r+0.587*g+0.114*b);
11    ...
12 FinAlgoritmo

```

Fig. 3.2. Función en pseudolenguaje pse



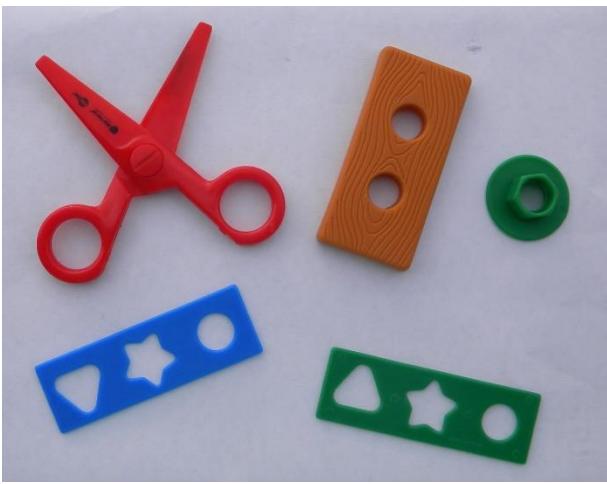
```

def rgb2gray(RGB):
    r= RGB[:, :, 0]
    g= RGB[:, :, 1]
    b= RGB[:, :, 2]
    gray = (0.299*r)+(0.587*g)+(0.114*b)

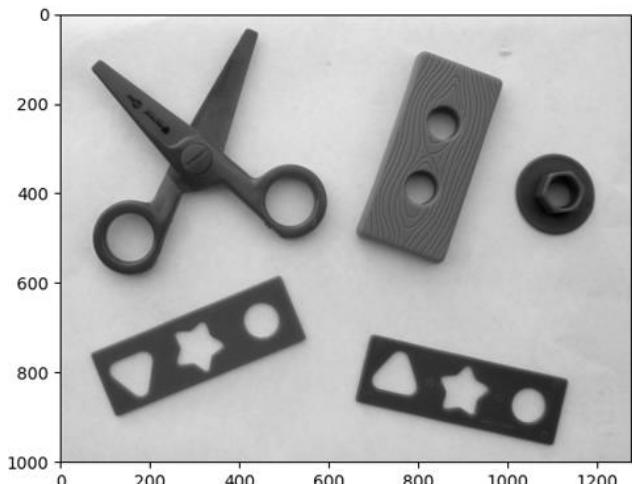
    return gray

```

Fig. 3.3. Algoritmo en Python



(a) Imagen RGB



(b) Imagen en escala de grises

Fig. 3.4. Conversión de RGB a escala de grises en Python²

4. Función imcrop

4.1. Uso

imcrop Recorta una imagen mostrada entre dos puntos específicos.

La sintaxis para su uso es:

- `imcrop (I,S)`

Donde “I” es la imagen a cortar, “S” es la distancia desde y donde se quiere recortar, distancias en pixeles.

Retorna C: Imagen recortada.

² Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

4.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la *función imcrop*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:

- `A = imread ('cartagena.jpg')`

2. Se usa la instrucción *imcrop* para recortar la imagen almacenada anteriormente en la variable “A” con dimensiones 200x250 y se almacena en una variable “B”:

- `B = imcrop (A, 1, 1, 199,249)`

3. Se usa la instrucción *imshow*, para mostrar la imagen recortada.

- `imshow (B)`

4.3. Resultado

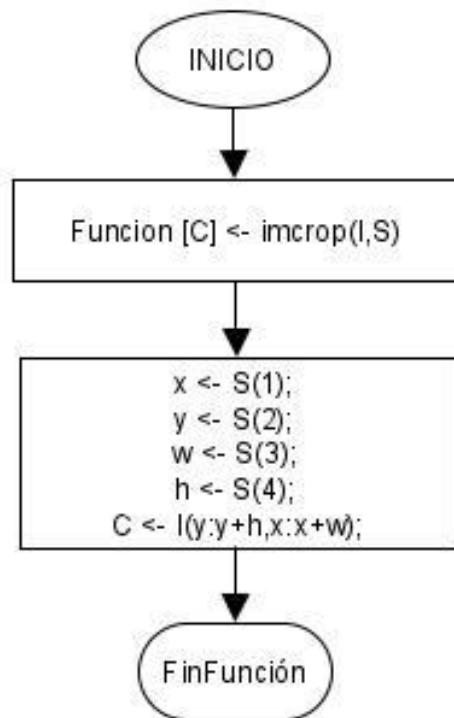


Fig. 4.1. Diagrama de flujo de la función

```
1 //Nombre: imcrop
2 //Parámetros: I es la imagen a recortar
3 //           S es la distancia desde y que se quiere cortar, distancias en pixeles.
4 //Devuelve: recorta una Imagen entre un par de puntos específicos.
5
6 Algoritmo imcrop
7   funcion [C] <- imcrop(I,S)
8     x <- S(1);
9     y <- S(2);
10    w <- S(3);
11    h <- S(4);
12    C <- I(y:y+h,x:x+w);
13 FinAlgoritmo
```

Fig. 4.2. Función en pseudolenguaje pse

```
● ● ●  
def imcrop(I,S):  
    x=S[0]  
    y=S[1]  
    w=S[2]  
    h=S[3]  
    c=I[x:x+w, y:y+h, :]  
    return c
```

Fig. 4.3. Algoritmo en Python

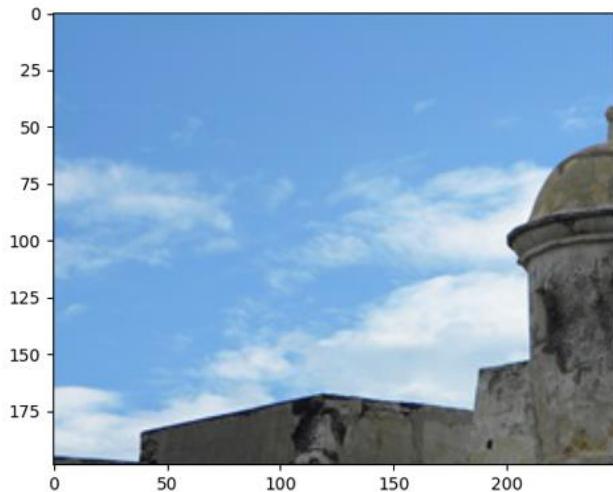


Fig. 4.4. Imagen cortada en Python

5. Función imresize

5.1. Uso

imresize Cambia el tamaño o realiza el escalamiento de una imagen.

La sintaxis para su uso es:

- `imresize (I, s)`

Donde “I” es la imagen para cambiar o escalar, “s” es la escala a la cual se quiere ajustar.

Si su entrada “m” es un vector de dos posiciones, mls y mcols es el valor del número de las filas, columnas y de su tamaño respectivamente.

Retorna A: Tamaño o escalamiento de una imagen.

5.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imresize*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:

- `A = imread ('cartagena.jpg')`

2. Se usa la instrucción *rgb2gray* para cambiar la imagen a escala de grises y se almacena en una variable “B”:

- `B = rgb2gray (A)`

3. Se usa la instrucción *imresize* para reducir la imagen a un 50 % de su escala normal y se almacena en una variable “C”.

- `C = imresize (B,0.5)`

4. Se usa la instrucción *imshow* para mostrar la imagen reducida:

- `imshow (C)`

5.3. Resultado

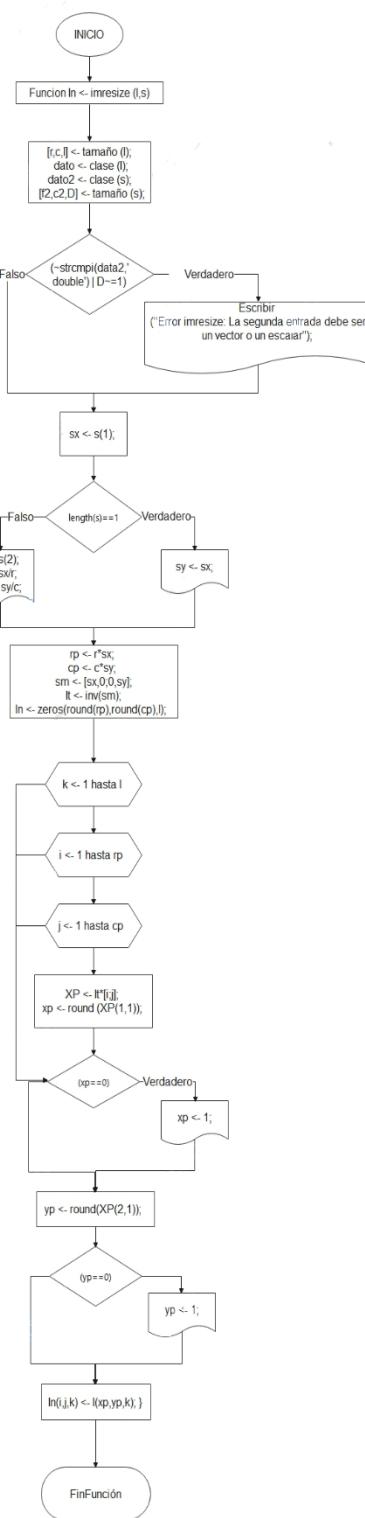


Fig. 5.1. Diagrama de flujo de la función

```

1 //Nombre: imresize
2 //Parámetros:Donde I es la imagen para hacer escalamiento,
3 //           s es la escala a la cual se quiere ajustar, r es un valor numérico.
4 //Devuelve una imagen s que es mmultiplicada por el tamaño de I
5
6
7 Algoritmo imresize
8
9 Funcion In <- imresize (I,s)
10    [r,c,l] <- tamaño (I);
11    dato <- clase (I);
12    dato2 <- clase (s); //segunda validacion
13    [f2,c2,D] <- tamaño (s);
14
15    Si (~strcmpi(data2,'double') | D==1)
16        Escribir ("Error imresize: La segunda entrada debe ser un vector o un escalar");
17    FinSi
18
19    sx <- s(1);
20
21    Si length(s)==1
22        sy <- sx;
23    SiNo
24        sy <- s(2);
25        sx <- sx/r;
26        sy<- sy/c;
27    FinSi
28
29
30    rp <- r*sx;                      //Filas de la imagen escalada
31    cp <- c*sy;                      //Columnas de la imagen escalada
32    sm <- [sx,0;0,sy];               //Matriz de escalamiento
33    It <- inv(sm);                  //Inversa de la matriz de escalamiento
34    In <- zeros(round(rp),round(cp),l); //Se crea la imagen escalada a partir de las nuevas filas y las nuevas columnas escaladas
35
36    Para k <- 1 hasta l
37        Para i <- 1 hasta rp          //Se recorren las filas escaladas
38            Para j <- 1 hasta cp          //Se recorren las columnas escaladas
39                XP <- It*[i;j];          //Se multiplica la inversa de la matriz de escalamiento por las coordenadas de la imagen escalada
40                xp <- round (XP(1,1)); //Se asigna la posicion 1,1 a la variable xp y se aproxima
41
42                Si (xp==0)
43                    xp <- 1;
44                FinSi
45
46
47                yp <- round(XP(2,1));      //Se asigna la posicion 2,1 a la variable xp y se aproxima
48
49                Si (yp==0)
50                    yp <- 1;
51                FinSi
52
53                In(i,j,k) <- I(xp,yp,k); //
54
55            FinPara
56        FinPara
57    FinPara
58
59    In=cast(In,data);
60 FinFuncion
61
62 FinAlgoritmo

```

Fig. 5.2. Función en pseudolenguaje pse

```

def imresize(I, s):
    I = I.astype(int)
    [r, c] = I.shape
    l = I.ndim
    data2 = type(s).__name__
    if((data2 != 'float' and data2 != 'list') or len(s) > 2):
        print("Error imresize: La segunda entrada debe ser un vector o un escalar")
    else:
        sx = s[0]
        if len(s) == 1:
            sy = sx
        else:
            sy = s[1]
            sx = sx/r
            sy = sy/c
        rp = int(r*sx)
        cp = int(c*sy)
        sm = [[sx, 0],[0,sy]]
        It = np.linalg.inv(sm).astype(int)
        In = np.zeros((round(rp),round(cp)), dtype=int)
        for k in range(1, l):
            for i in range(1, rp):
                for j in range(1, cp):
                    XP = It * [i,j]
                    xp = XP[0,0]
                    if xp == 0:
                        xp = 1
                    yp = round(XP[1,1])
                    if(yp == 0):
                        yp = 1
                    In[i,j] = I[xp,yp]
    return In

```

Fig. 5.3. Algoritmo en Python.

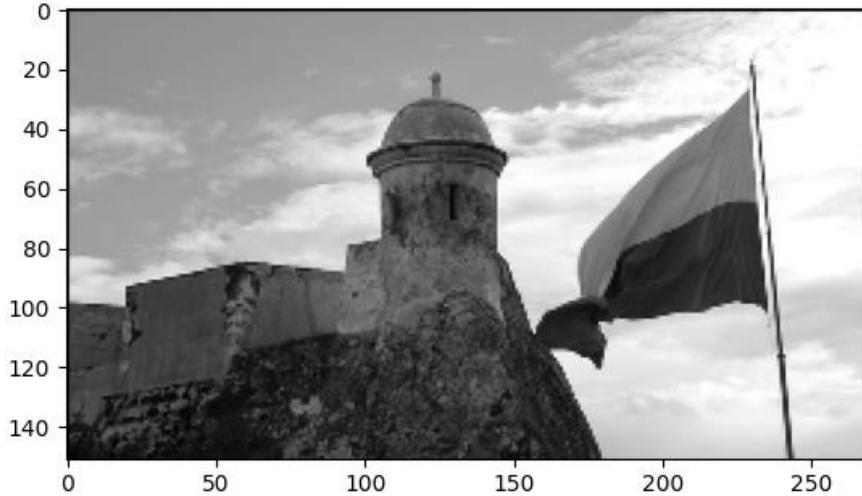


Fig. 5.4 Imagen reducida.

6. Función imtranslate

6.1. Uso

imtranslate Realiza la translación de una imagen.

La sintaxis para su uso es:

- `imtranslate(I, t,varargin)`

Retorna In: Traslación de una imagen.

Donde “I” es la imagen a trasladar, “t” es el valor que se va a trasladar, “t” es un número entero,”varargin” se le puede asignar como entrada cualquier número de argumentos.

Como parámetros posee las siguientes diferentes entradas tipo string:

- '**full**': La imagen se traslada y crea un zero-pad para no recortar la imagen.
- '**same**': La imagen de entrada de salida tienen el mismo tamaño.

6.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imtranslate*.

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:

- `A = imread ('cartagena.jpg')`

2. Se traslada la imagen a la mitad usando la instrucción *imtranslate*, es decir 100 píxeles en las filas y columnas; y se usa la instrucción *imshow* para mostrar la imagen.

- $C = \text{imtranslate}(A, 100, \text{'full'})$
- $\text{imshow}(C)$

6.3. Resultado

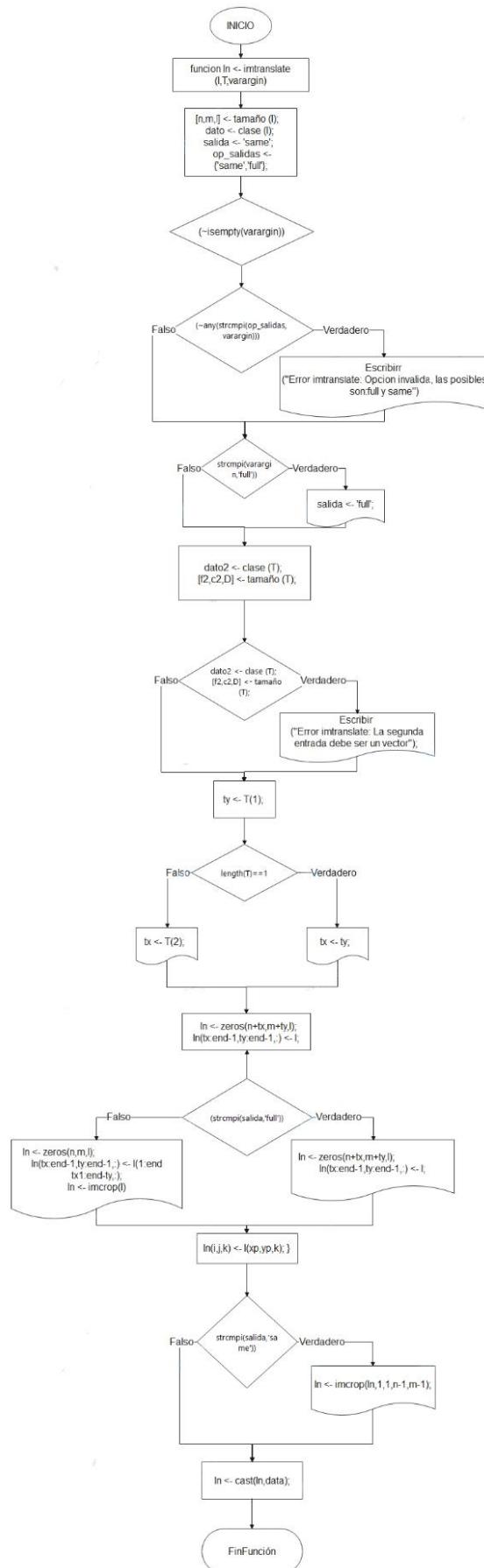


Fig. 6.1. Diagrama de flujo de la función.

```

1 //Nombre: imtranslate
2 //Parámetros: Donde I es la imagen que se va a trasladar.
3 //              T es el valor en pixeles que se va a trasladar, t es un valor numérico entero.
4 //Devuelve la imagen de referencia espacial traducida, con su objeto de referencia espacial asociado.
5
6 Algoritmo imtranslate
7
8 funcion In <- imtranslate (I,T,varargin)
9     [n,m,l] <- tamaño (I);
10    dato <- clase (I);
11    salida <- 'same';
12    op_salidas <- {'same','full'};
13
14    Si (~isempty(varargin))
15        Si (~any(strcmpi(op_salidas,varargin)))
16            Escribir ("Error imtranslate: Opcion invalida, las posibles son:full y same")
17            FinSi
18
19        Si (strcmpi(varargin,'full'))
20            salida <- 'full';
21            FinSi
22    FinSi
23
24 //segunda validacion
25    dato2 <- clase (T);
26    [f2,c2,D] <- tamaño (T);
27
28    Si (~strcmpi(data2,'double') | D~=1)
29        Escribir ("Error imtranslate: La segunda entrada debe ser un vector");
30        FinSi
31
32    ty <- T(1);
33
34    Si length(T)==1
35        tx <- ty;
36    SiNo
37        tx <- T(2);
38    FinSi
39
40    In <- zeros(n+tx,m+ty,l);
41    In(tx:end-l,ty:end-l,:) <- I;
42
43    Si (strcmpi(salida,'full'))
44        In <- zeros(n+tx,m+ty,l);
45        In(tx:end-l,ty:end-l,:) <- I;
46    SiNo
47        In <- zeros(n,m,l);
48        In(tx:end-l,ty:end-l,:) <- I(l:end-tx,l:end-ty,:);
49        In <- imcrop(I)
50    FinSi
51
52    Si(strcmpi(salida,'same'))
53        In <- imcrop(In,1,1,n-1,m-1);
54    FinSi
55
56    In <- cast(In,data);
57
58 FinFuncion
59
60 FinAlgoritmo
61
62 //NOTA:
63
64 //full: La imagen se traslada y genera un zero-pad para no recortar la imagen.
65 //same: La imagen de salida tiene el mismo tamaño de la imagen de entrada (defecto).
66 //cast: Convierte variable a diferente tipo de datos.
67 //strcmpi: compara cadenas ignorando el caso.
68 //isempty: Determinar si la matriz está vacía

```

Fig. 6.2. Función en pseudolenguaje pse.

```

def imtranslate(I,T,varargin):
    [n,m,l]= I.shape
    salida='same'
    op_salidas=['same','full']
    if varargin != '':
        if varargin not in op_salidas:
            print ("Error imtranslate: Opcion invalida, las posibles son: full y same")
        if varargin == 'full':
            salida = 'full'

    ty=T[0]
    if len(T) == 1:
        tx = ty
    else:
        tx=T[1]

    In=np.zeros([n+tx,m+ty, l], dtype=int)
    In[tx:, ty:, :] = I

    if salida == 'same':
        In = crp.imcrop(In, [1,1,n-1,m-1])
    else:
        In = In

    return In

```

Fig. 6.3. Algoritmo en python



Fig. 6.4. Imagen trasladada.

7. Función imhist

7.1. Uso

imhist Evalúa el histograma de una imagen.

La sintaxis para su uso es:

- `imhist (r)`

Donde “r” es la imagen.

7.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imhist*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:

- `A= imread ('cartagena.jpg')`

2. Se extrae la capa roja de la imagen “A”.

- `capa_R = A (:, :, 1)`

3. Se usa la instrucción *imhist* para mostrar la matriz de la capa.

- `print (imhist (capa_R))`

7.3. Resultado

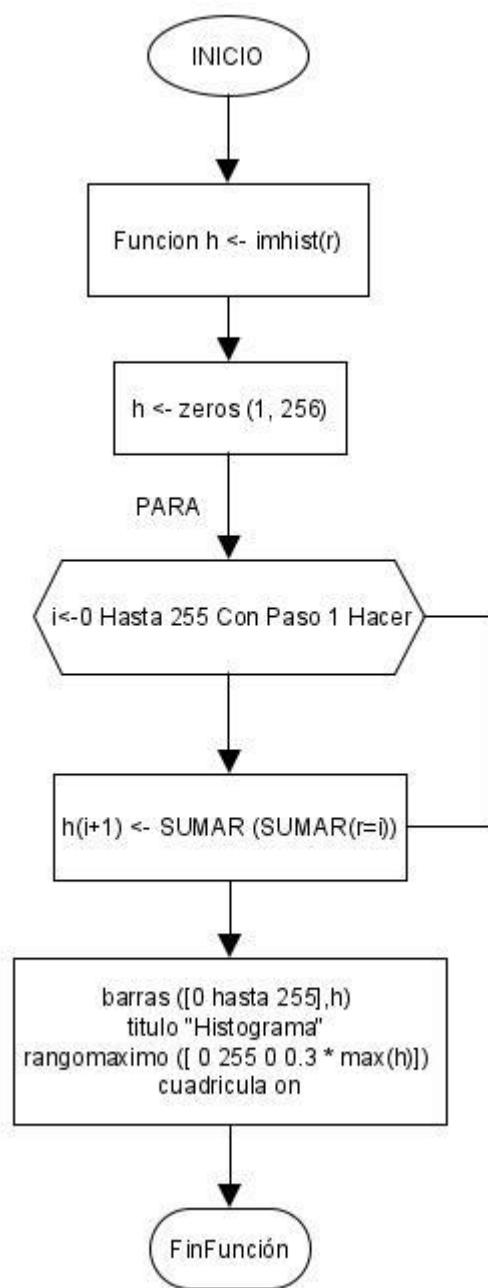


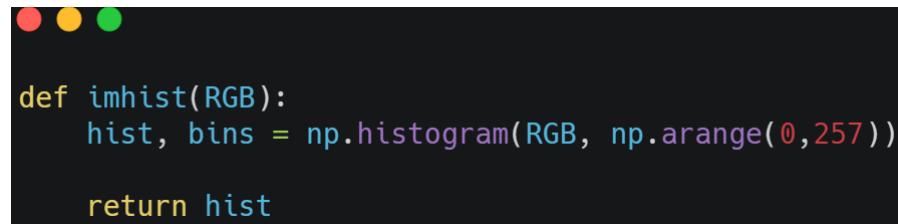
Fig. 7.1. Diagrama de flujo de la función

```

1 //Nombre: imhist
2 // Calcula y permite la visualización del histograma de una imagen.
3 // Parámetros: r es la imagen en escala de grises o una de las capas de la imagen
4 //              h es el histograma.
5
6 Algoritmo Histograma
7
8 Funcion h <- imhist(r)
9     h <- zeros (l, 256)
10    Definir i Como Entero
11    Para i<-0 Hasta 255 Con Paso 1 Hacer
12        h(i+1) <- SUMAR (SUMAR(r=i)) // suma de las columnas y suma de las filas
13    FinPara
14
15    barras ([0 hasta 255],h)
16    titulo "Histograma"
17    rangomaximo ([ 0 255 0 0.3 * max(h) ])
18    cuadricula on
19 FinFuncion
20
21 FinAlgoritmo

```

Fig. 7.2. Función en pseudolenguaje pse



```

def imhist(RGB):
    hist, bins = np.histogram(RGB, np.arange(0,257))

    return hist

```

Fig. 7.3. Algoritmo en Python.

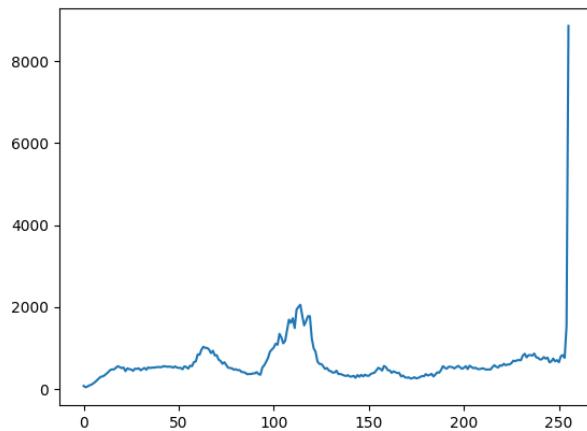


Fig. 7.4 Histograma de la imagen.

8. Función imhist2

imhist2 Evalúa la matriz del histograma de una imagen.

La sintaxis para su uso es:

- `imhist2 (RGB)`

Donde “RGB” es la imagen a color y se extraerá solo la capa R.

8.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imhist2*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:

- `A = imread ('cartagena.jpg')`

2. Se extrae la capa roja de la imagen “A”.

- `capa_R = A (:, :, 1)`

3. Se usa la instrucción *imhist* para mostrar la matriz de la capa.

- `print (imhist (capa_R))`

8.3. Resultado

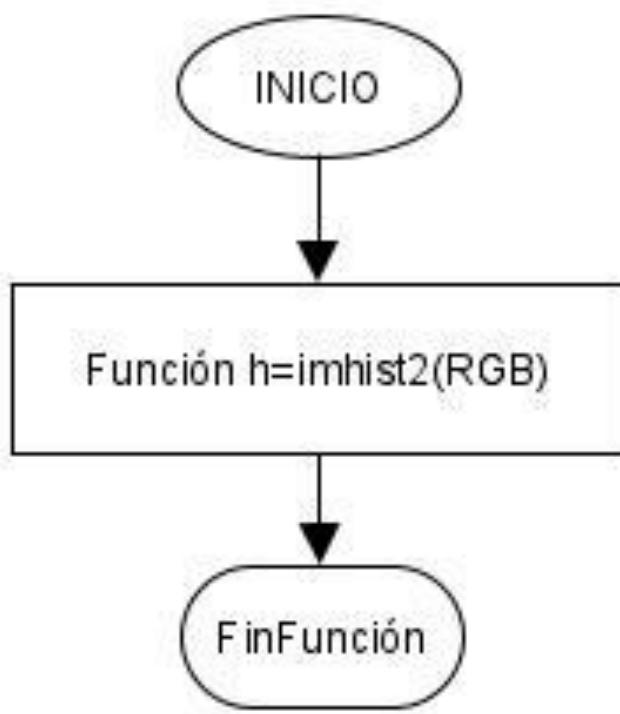
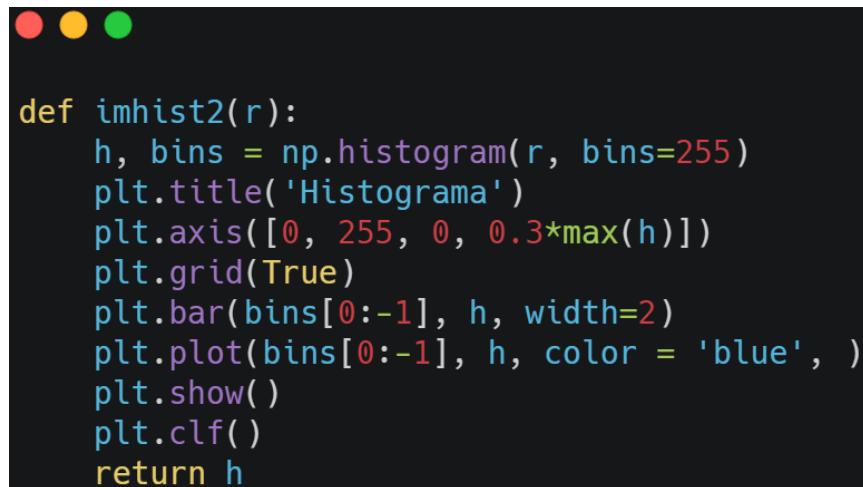


Fig. 8.1 Diagrama de flujo de la función

```
1 //Nombre: imhist2
2 //Parámetros: RGB es la imagen a color donde se extrae una capa.
3 //Devuelve: h donde es la matriz de una capa
4
5 Algoritmo imhist2
6
7 Función h=imhist2(RGB)
8     (hist, bins) = histograma (RGB,:0,257)
9
10 FinFuncion
11
12
13 FinAlgoritmo
```

Fig. 8.2 Función en pseudolenguaje pse.



```
def imhist2(r):
    h, bins = np.histogram(r, bins=255)
    plt.title('Histograma')
    plt.axis([0, 255, 0, 0.3*max(h)])
    plt.grid(True)
    plt.bar(bins[0:-1], h, width=2)
    plt.plot(bins[0:-1], h, color = 'blue', )
    plt.show()
    plt.clf()
    return h
```

Fig. 8.3. Algoritmo en Python.

9. Función mean2

9.1. Uso

mean2 Evalúa el promedio o media de los elementos de una matriz de una imagen 2D.

La sintaxis para su uso es:

- `mean2 (I)`

Retorna “promedio”: La media de los elementos de una matriz.

Donde “I” es una matriz de tipo “double”.

9.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *mean2*.

1. Se lee una imagen llamada “manzana.jpg” y se almacena en una variable “A”:

- `A = imread ('manzana.jpg')`

2. Se cambia la imagen a escala de grises y se almacena en una variable “B”:

- `B = rgb2gray (A)`

3. Se usa la instrucción *mean2* para evaluar el brillo de “B” y se almacena en una variable “C”.

- `C = mean2 (B)`

9.3. Resultado

Escalar de clase “double”:

- $C = 184.30923$

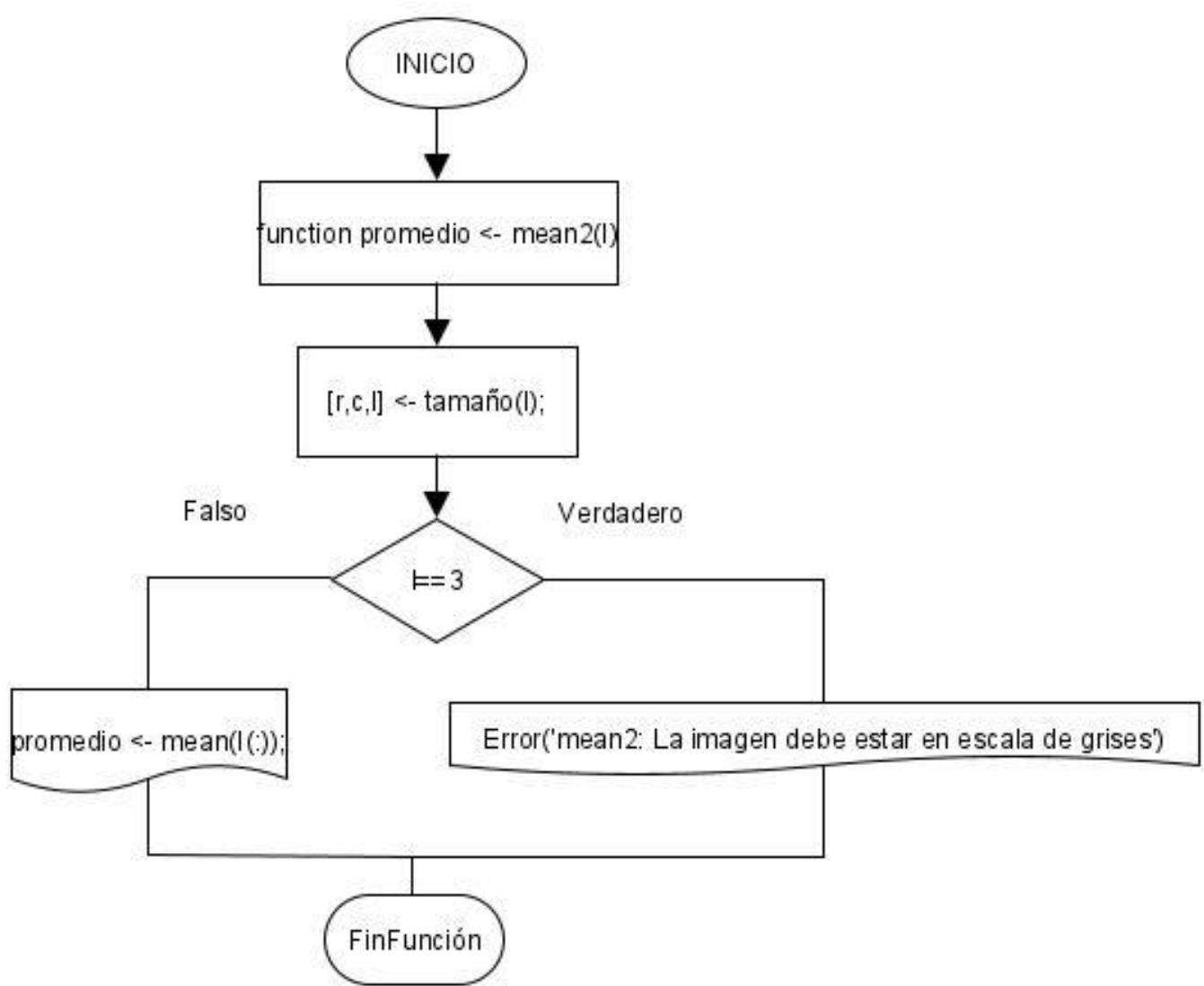


Fig. 9.1. Diagrama de flujo de la función.

```
1 //Nombre: mean2
2 //Parámetros: I es una matriz de tipo "double" o de cualquier clase de enteros.
3 //Devuelve: promedio es la media de los elementos de una matriz.
4
5 function promedio=mean2(I)
6 [r,c,l]=size(I);
7 if l==3
8     Error('mean2: La imagen debe estar en escala de grises')
9 else
10    promedio=mean(I(:));
11 end
12 end
```

Fig. 9.2. Función en pseudolenguaje pse.



```
def mean2(I):
    promedio = 0
    l = I.ndim
    if l == 3:
        print("Error mean2: La imagen debe estar en escala de grises")
    else:
        promedio = np.mean(I[:])
    return promedio
```

Fig. 9.3 Algoritmo en Python.

10. Función std2

10.1. Uso

std2 Evalúa la desviación estándar de los elementos de una matriz de una imagen en 2D.

La sintaxis para su uso es:

- `std2(I)`

Donde “I” es la matriz numérica.

Retorna “sd”: La desviación estándar de los elementos de una matriz.

10.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *std2*.

1. Se crea una matriz numérica y se almacena en una variable “A”
 - $A = [4, 7, 8; 7, 9, 0; 5, 6, 7]$
2. Se evalúa la desviación estándar de la matriz y se almacena en una variable “Des”.
 - $Des = std2 (A)$

10.3. Resultado

- $Des = 2.6667$

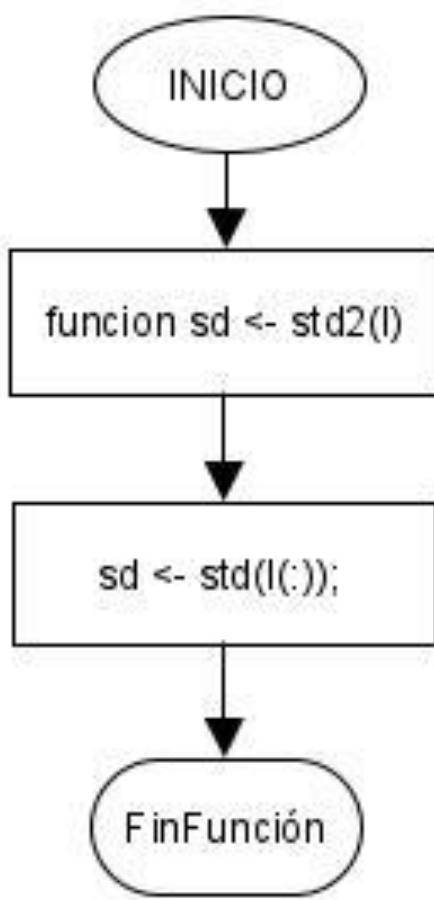


Fig. 10.1. Diagrama de flujo de la función

```

1 //Nombre: std2
2 //Parámetros: Donde I es la matriz numérica
3 //           sd es un valor numérico que contiene la desviación estandar de la matriz.
4
5 Algoritmo std2
6
7 funcion sd <- std2(I)
8     sd <- std(I(:));
9
10 FinFuncion
  
```

Fig. 10.2. Función en pseudolenguaje pse



```
M = np.matrix('4, 7, 8; 7, 9, 0; 5, 6, 7')

def std2(I):
    sd = I.std()
    return sd
```

Fig. 10.3 Algoritmo en Python

11. Función imabsdiff

11.1. Uso

imabsdiff Evalúa la diferencia absoluta de dos imágenes del mismo tamaño.

La sintaxis para su uso es:

- *imabsdiff (a,b)*

Retorna “s”: La diferencia absoluta en el elemento correspondiente de la matriz.

Resta cada elemento en la matriz “a” con el correspondiente elemento de la matriz b. “a” y “b” son reales, con el mismo tamaño y clase.

La matriz que retorna “s”, tiene el mismo tamaño y clase de “a” y “b”.

11.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imabsdiff*:

1. Se lee una imagen llamada “fondo.jpg” y se almacena en una variable “A”:

- *A = imread ('fondo.jpg')*

2. Se lee una imagen llamada “pera.jpg” y se almacena en una variable “B”:

- $B = \text{imread} ('pera.jpg')$

3. Para ajustar la diferencia absoluta de dos imágenes se requiere un alfa y se multiplica “A” y “B” por el 50 %:

- $\alpha = 0.5$
- $A_p = (\alpha * A)$
- $B_p = (1 - \alpha) * B$

4. Se evalúa la diferencia absoluta de las dos imágenes y se almacena en una variable “dif”

- $\text{dif} = \text{imabsdiff} (A_p, B_p)$

5. Se muestra la diferencia absoluta de las dos imágenes con la instrucción imshow.

- $\text{imshow} (\text{dif})$

11.3. Resultado

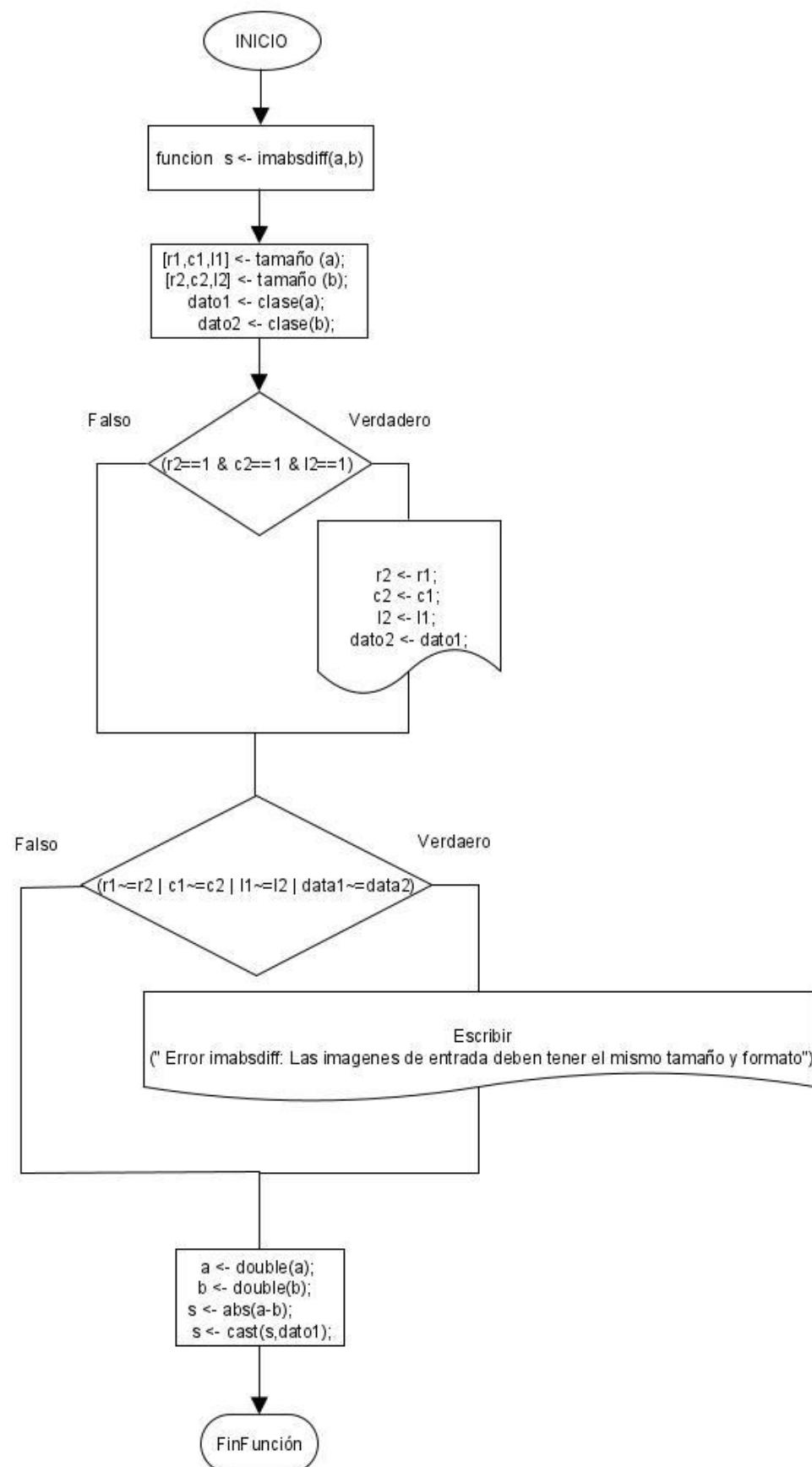


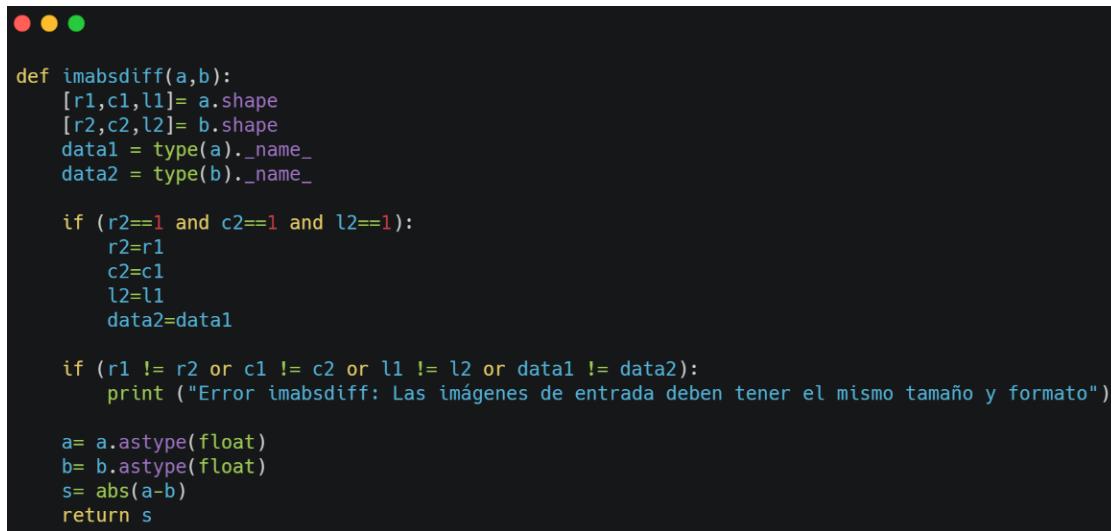
Fig. 11.1. Diagrama de flujo de la función.

```

1 //Nombre: imabsdiff
2 //Parámetros: Donde a y b son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato.
3 // Cuando la entrada b es un valor numérico (s), se realiza la diferencia entre la imagen a y el valor numérico.
4 //Devuelve la diferencia absoluta en el elemento correspondiente de la matriz de salida.
5
6 Algoritmo imabsdiff
7 función s <- imabsdiff(a,b)
8     [r1,c1,l1] <- tamaño (a);
9     [r2,c2,l2] <- tamaño (b);
10    datol <- clase(a);
11    dato2 <- clase(b);
12
13    Si (r2==l & c2==l & l2==l) //constante
14        r2 <- r1;
15        c2 <- c1;
16        l2 <- l1;
17        dato2 <- datol;
18    FinSi
19
20
21    Si (r1==r2 | c1==c2 | l1==l2 | datol==dato2)
22        Escribir (" Error imabsdiff: Las imágenes de entrada deben tener el mismo tamaño y formato")
23    FinSi
24
25
26    a <- double(a);
27    b <- double(b);
28    s <- abs(a-b);
29    s <- cast(s,datol);
30 FinFunción
31
32 FinAlgoritmo
33
34 //NOTA:
35 // double: Conversión a un número real.
36 //cast: Convierte variable S a diferente tipo de datos.

```

Fig. 11.2. Función en pseudolenguaje pse



```

def imabsdiff(a,b):
    [r1,c1,l1]= a.shape
    [r2,c2,l2]= b.shape
    data1 = type(a).__name__
    data2 = type(b).__name__

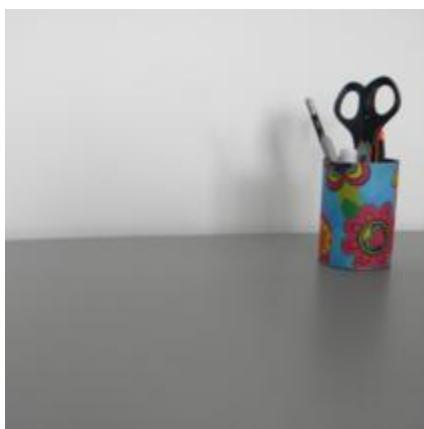
    if (r2==l1 and c2==l1 and l2==l1):
        r2=r1
        c2=c1
        l2=l1
        data2=data1

    if (r1 != r2 or c1 != c2 or l1 != l2 or data1 != data2):
        print ("Error imabsdiff: Las imágenes de entrada deben tener el mismo tamaño y formato")

    a= a.astype(float)
    b= b.astype(float)
    s= abs(a-b)
    return s

```

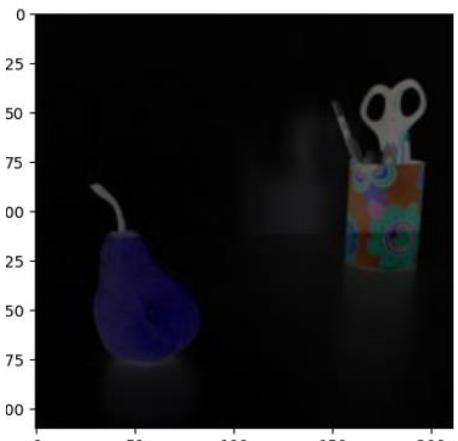
Fig. 11.3. Algoritmo en Python.



(a) Imagen de fondo



(b) Imagen con el objeto



(c) Diferencia absoluta

Fig. 11.4. Diferencia absoluta en Python³

12. Función imadd

12.1. Uso

imadd La suma de dos imágenes o una imagen por una constante del mismo tamaño.

La sintaxis para su uso es:

- `imadd (a, b)`

Donde “a” y “b” son imágenes.

Suma cada elemento en la matriz “a” con el correspondiente elemento de la matriz “b”. “a” y “b” son reales, con el mismo tamaño y clase, o “b” es un escalar de tipo “double”.

La matriz de salida “s”, tiene el mismo tamaño y clase si “a” y “b” son iguales.

Retorna “s”: La suma de dos imágenes

12.2. Ejemplo

³ Imagen (a), (b), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imadd*:

1. Se lee una imagen llamada “pera.jpg” y se almacena en una variable “A” y se lee una imagen llamada “fondo.jpg” y se almacena en una variable “B”; y ambas variables se dividen por 255 para que la matriz quede en un rango entre 0 y 1:

- A = imread ('pera.jpg')/255
- B = imread ('fondo.jpg')/255

2. Para el ajuste de la suma de dos imágenes se requiere un alfa:

- alpha = 0.5
- C = alpha*A
- D= (1-alpha)*B

3. Se suman ambas imágenes y se almacena el resultado en una variable “imadd”

- imadd = imadd (C, D)

4. Se muestra la suma de las dos imágenes con la instrucción *imshow*.

- imshow (imadd)

12.3. Resultado

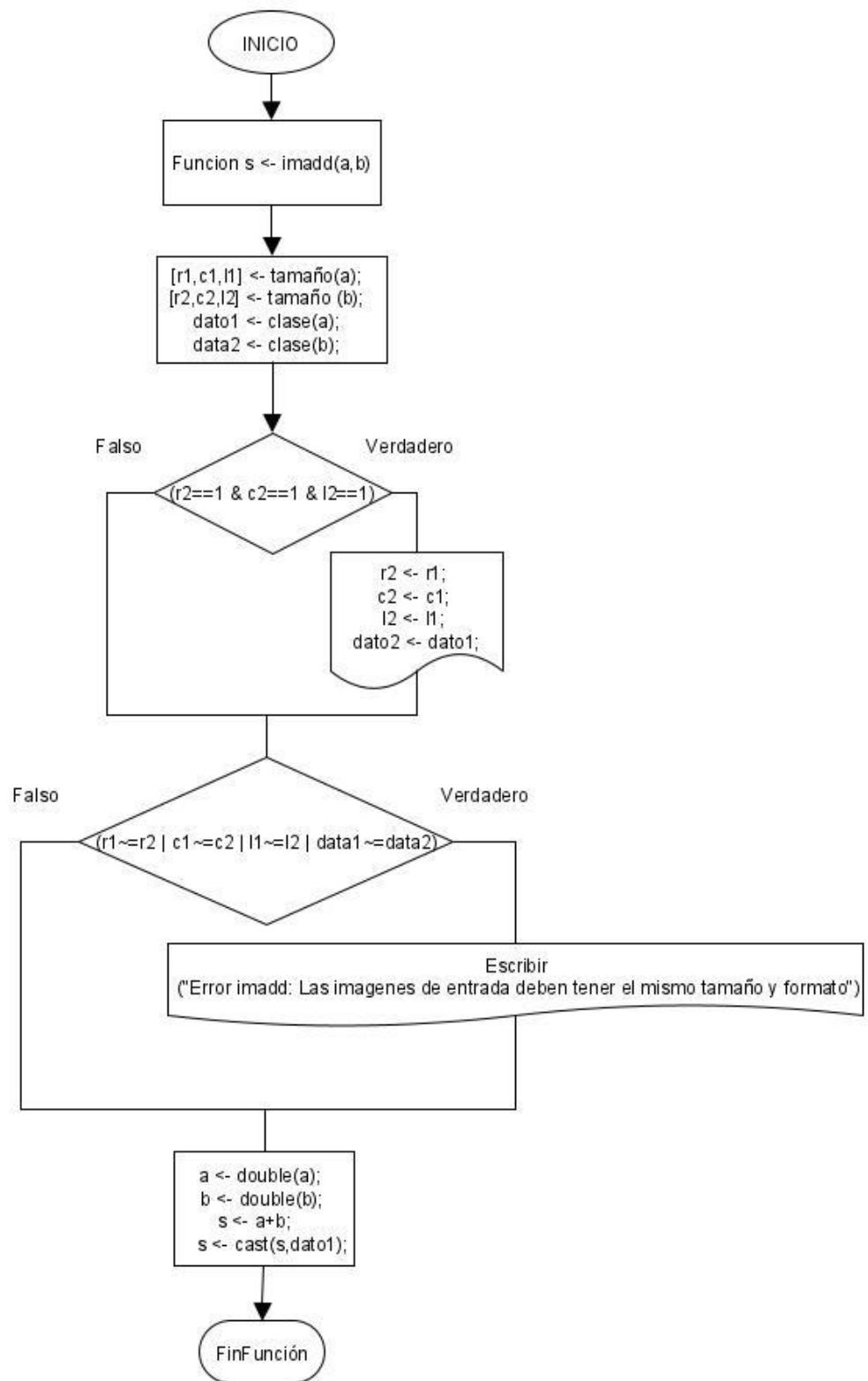


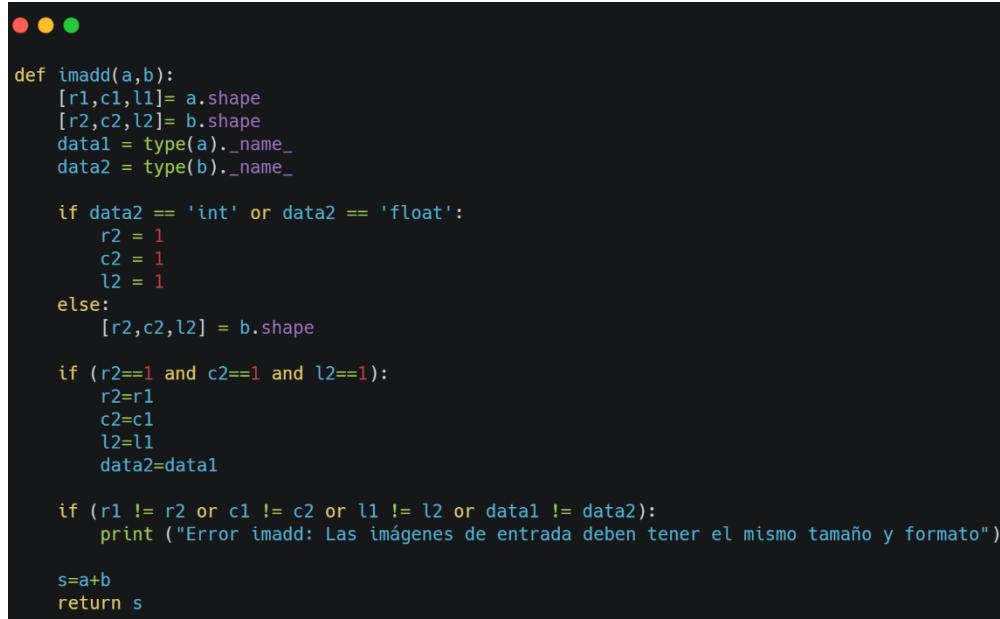
Fig. 12.1. Diagrama de flujo de la función.

```

1 //Nombre: imadd
2 //Parámetros: Donde a y b son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato.
3 // Cuando la entrada b es un valor numérico (s), se realiza la suma entre la imagen a y el valor numérico.
4 //Devuelve la suma en el elemento correspondiente de la matriz de salida
5
6 Algoritmo imadd
7
8 Funcion s <- imadd(a,b)
9     [r1,c1,l1] <- tamaño(a);
10    [r2,c2,l2] <- tamaño (b);
11    dato1 <- clase(a);
12    dato2 <- clase(b);
13
14    Si (r2==1 & c2==1 & l2==1) //constante
15        r2 <- r1;
16        c2 <- c1;
17        l2 <- l1;
18        dato2 <- dato1;
19    FinSi
20
21
22    Si (r1==r2 | c1==c2 | l1==l2 | dato1==dato2)
23        Escribir ("Error imadd: Las imágenes de entrada deben tener el mismo tamaño y formato")
24    FinSi
25
26
27    a <- double(a);
28    b <- double(b);
29    s <- a+b;
30    s <- cast(s,dato1);
31
32 FinFuncion
33
34 FinAlgoritmo
35 //NOTA:
36 // double: Conversión a un número real.
37 //cast: Convierte variable S a diferente tipo de datos.

```

Fig. 12.2. Función en pseudolenguaje pse



```

def imadd(a,b):
    [r1,c1,l1]= a.shape
    [r2,c2,l2]= b.shape
    dato1 = type(a).__name__
    dato2 = type(b).__name__

    if dato2 == 'int' or dato2 == 'float':
        r2 = 1
        c2 = 1
        l2 = 1
    else:
        [r2,c2,l2] = b.shape

    if (r2==1 and c2==1 and l2==1):
        r2=r1
        c2=c1
        l2=l1
        dato2=dato1

    if (r1 != r2 or c1 != c2 or l1 != l2 or dato1 != dato2):
        print ("Error imadd: Las imágenes de entrada deben tener el mismo tamaño y formato")

    s=a+b
    return s

```

Fig. 12.3 .Algoritmo en Python



Fig. 12.4. Suma de dos imágenes en Python⁴

13. Función imdivide

13.1. Uso

imdivide Divide una imagen en otra o divide una imagen por constante.

La sintaxis para su uso es:

- `imdivide(a,b)`

Divide cada elemento en la matriz “a” con el correspondiente elemento de la matriz “b”. “a” y “b” son reales, con el mismo tamaño y clase, o “b” es un escalar de tipo “double”.

La matriz de salida “s”, tiene el mismo tamaño y clase si “a” y “b” son iguales.

Retorna “s”: División de una imagen, el resultado en el elemento correspondiente de la matriz de salida “s”

⁴ Imagen (a), (b), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

13.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imdivide*:

1. Se lee una imagen llamada “pera.jpg” y se almacena en una variable “A” y se lee una imagen llamada “fondo.jpg” y se almacena en una variable “B”; y la variable “A” se divide por 255 para que la matriz quede en un rango entre 0 y 1:

- A = imread ('pera.jpg')/255
- B = imread ('fondo.jpg')

2. Para ajustar la división de dos imágenes se requiere un alfa:

- alfa=10
- C=alfa*A
- D=(1-alfa)*B

3. Se usa la instrucción *imdivide* para dividir las imágenes “pera.jpg” y un escalar, y se almacena el resultado en una variable “imdiv”

- imdiv = imdivide (A,2)

4. Se muestra la división de las dos imágenes con la instrucción *imshow*.

- imshow (imdiv)

13.3. Resultado

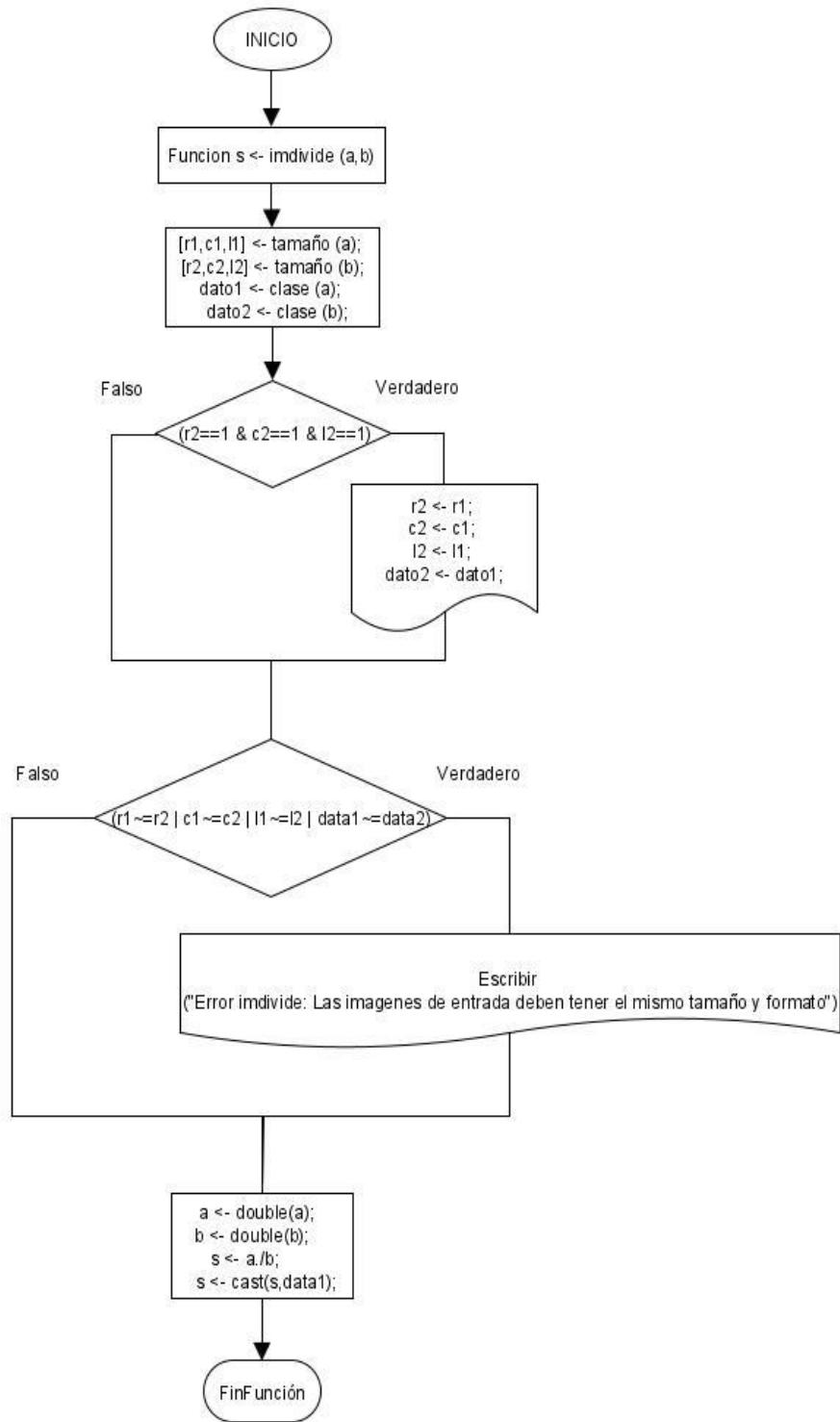


Fig. 13.1. Diagrama de flujo de la función.

```

1 //Nombre: imdivide
2 //Parámetros: Donde a y b son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato.
3 //           Cuando la entrada b es un valor numérico s, se realiza la división entre la imagen I y el valor numérico.
4 // Devuelve el resultado en el elemento correspondiente de la matriz de salida.
5
6 Algoritmo imdivide
7
8 Función s <- imdivide (a,b)
9     [r1,c1,l1] <- tamaño (a);
10    [r2,c2,l2] <- tamaño (b);
11    dato1 <- clase (a);
12    dato2 <- clase (b);
13
14 Si (r2==1 & c2==1 & l2==1) //constante
15     r2 <- r1;
16     c2 <- c1;
17     l2 <- l1;
18     dato2 <- dato1;
19 FinSi
20
21 Si (r1==r2 | c1==c2 | l1==l2 | dato1==dato2)
22     Escribir ("Error imdivide: Las imágenes de entrada deben tener el mismo tamaño y formato")
23 FinSi
24
25
26 a <- double(a);
27 b <- double(b);
28 s <- a./b;
29 s <- cast(s,dato1);
30
31 FinFunción
32
33 FinAlgoritmo
34
35 //NOTA:
36 //cast: Convierte variable S a diferente tipo de datos.
37 //double: Conversión a un número real.
38

```

Fig. 13.2. Función en pseudolenguaje pse



```

def imdivide(a,b):
    [r1,c1,l1]= a.shape
    dato1 = type(a).__name__
    dato2 = type(b).__name__
    if dato2 == 'int' or dato2 == 'float':
        r2 = 1
        c2 = 1
        l2 = 1
    else:
        [r2,c2,l2] = b.shape

    if (r2==1 and c2==1 and l2==1):
        r2= r1
        c2= c1
        l2= l1
        dato2=dato1

    if (r1 != r2 or c1 != c2 or l1 != l2 or dato1 != dato2):
        print ("Error imdivide: Las imágenes de entrada deben tener el mismo tamaño y formato")

    s = a/b

    return s

```

Fig. 13.3. Algoritmo en Python.

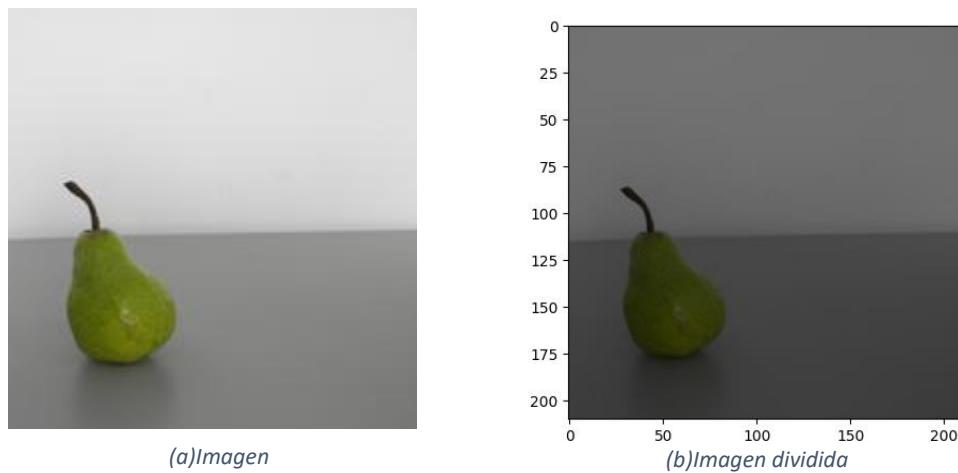


Fig. 13.4. División de imágenes en Python.⁵

14. Función immultiply

14.1. Uso

immultiply Multiplica dos imágenes o una imagen por una constante.

La sintaxis para su uso es:

- `immultiply(a,b)`

Multiplica cada elemento en la matriz “a” con el correspondiente elemento de la matriz b. “a” y “b” son reales, con el mismo tamaño y clase, o “b” es un escalar de tipo “double”.

La matriz de salida “s”, tiene el mismo tamaño y clase si “a” y “b” son iguales.

Retorna “s”: La imagen multiplicada, y devuelve el resultado en el elemento correspondiente de la matriz de salida s

⁵ Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

14.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *immultiply*:

1. Se lee una imagen llamada “pera.jpg” y se almacena en una variable “A”, y se lee una imagen llamada “fondo.jpg” y se almacena en una variable “B”; y ambas variables se dividen por 255 para que la matriz quede en un rango entre 0 y 1”:

- A = imread ('pera.jpg')/255
- B = imread ('fondo.jpg')/255

2. Se muestra las dos imágenes A y B con la instrucción imshow

- imshow (A)
- imshow(B)

3. Se usa la instrucción *immultiply* para multiplicar las imágenes “pera.jpg” y “fondo.jpg” y se almacena el resultado en una variable “C”.

- C = immultiply (A,B)

4. Se muestra la multiplicación de las dos imágenes con la instrucción imshow

- imshow(C)

14.3. Resultado

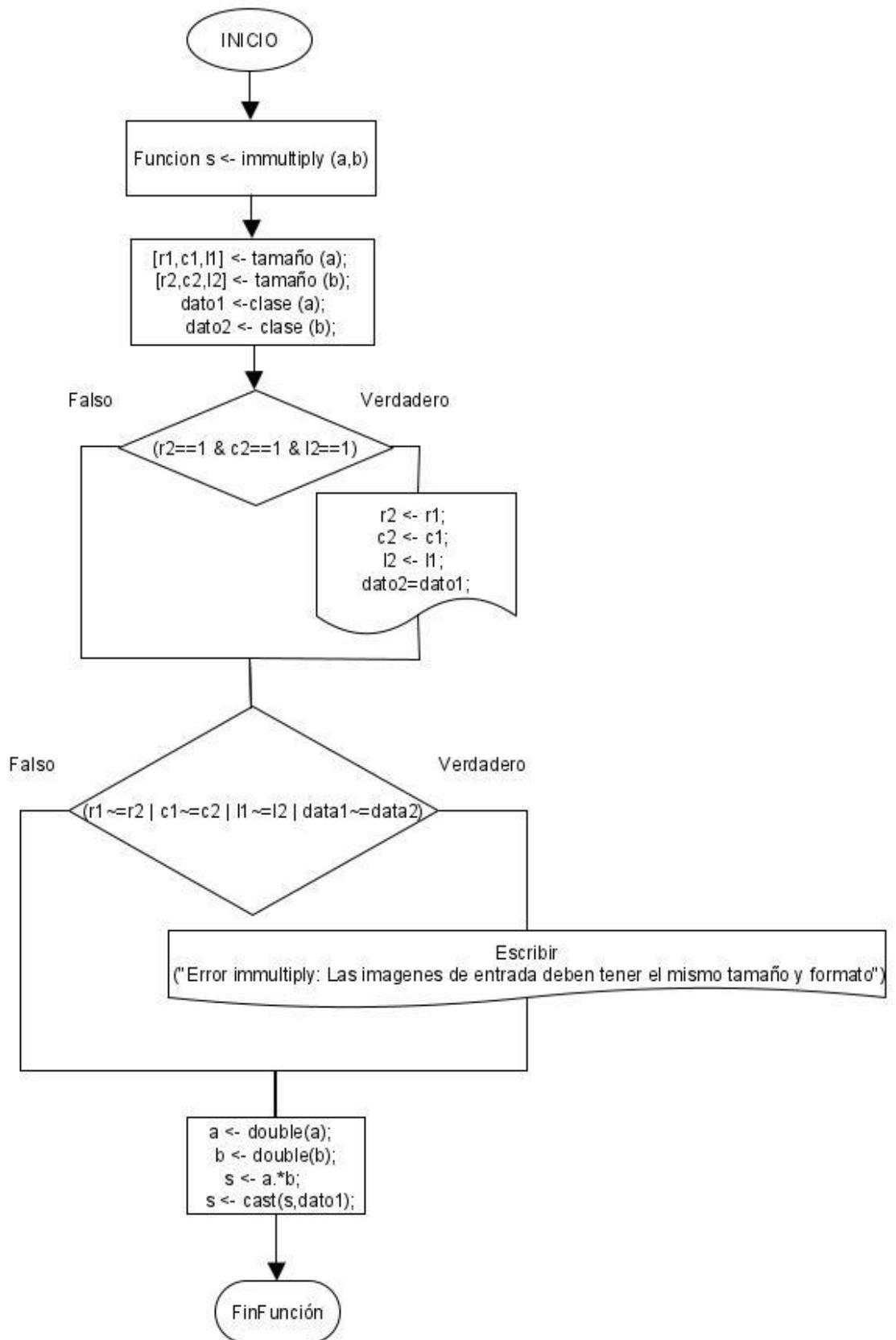


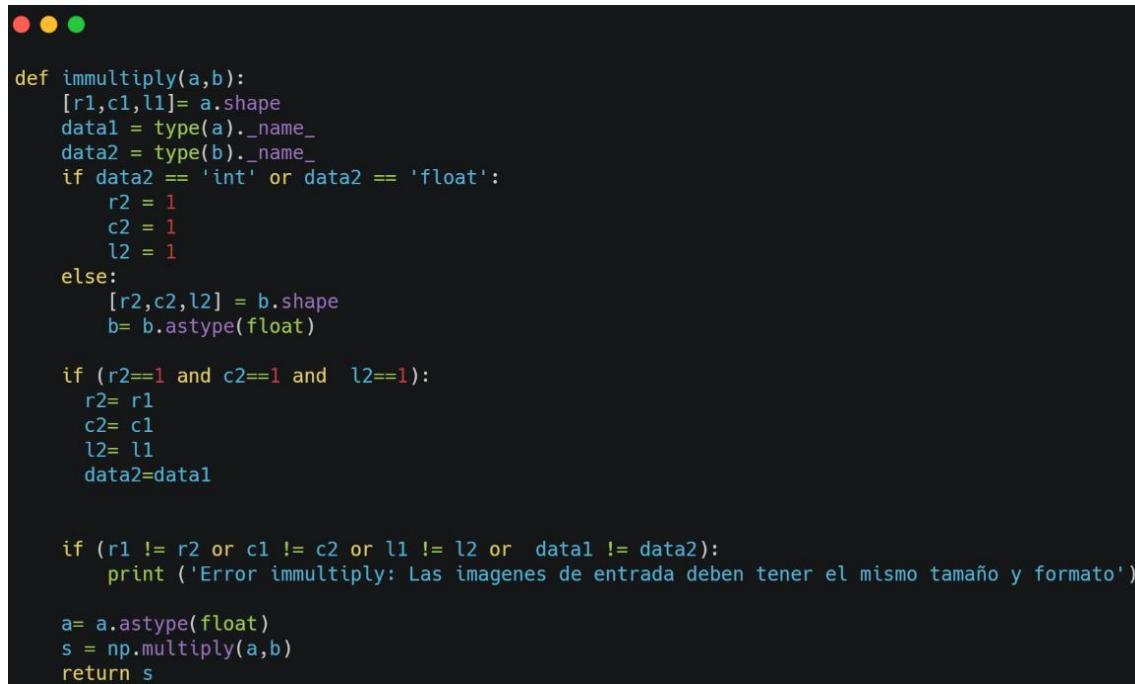
Fig. 14.1. Diagrama de la función.

```

1 //Nombre: immultiply
2 //Parámetros: Donde a y b son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato.
3 // Cuando la entrada b es un valor numérico (s), se realiza la multiplicación entre la imagen a y el valor numérico.
4 // Devuelve el producto en el elemento correspondiente de la matriz de salida.
5
6 Algoritmo immultiply
7
8 Funcion s <- immultiply (a,b)
9   [r1,c1,l1] <- tamaño (a);
10  [r2,c2,l2] <- tamaño (b);
11  datol <-clase (a);
12  dato2 <- clase (b);
13
14  Si (r2==1 & c2==1 & l2==1) //constante
15    r2 <- r1;
16    c2 <- c1;
17    l2 <- l1;
18    dato2=datol;
19  FinSi
20
21
22  Si (r1==r2 | c1==c2 | l1==l2 | datol==dato2)
23    Escribir ("Error immultiply: Las imágenes de entrada deben tener el mismo tamaño y formato")
24  FinSi
25
26
27  a <- double(a);
28  b <- double(b);
29  s <- a.*b;
30  s <- cast(s,datol);
31
32 FinFuncion
33
34 FinAlgoritmo
35
36 //NOTA: Funciones.
37 // double: Conversión a un número real.
38 //cast: Convierte variable S a diferente tipo de datos.

```

Fig. 14.2. Función en pseudolenguaje pse



```

def immultiply(a,b):
    [r1,c1,l1]= a.shape
    data1 = type(a).__name__
    data2 = type(b).__name__
    if data2 == 'int' or data2 == 'float':
        r2 = 1
        c2 = 1
        l2 = 1
    else:
        [r2,c2,l2] = b.shape
        b= b.astype(float)

    if (r2==1 and c2==1 and l2==1):
        r2= r1
        c2= c1
        l2= l1
        dato2=datol

    if (r1 != r2 or c1 != c2 or l1 != l2 or data1 != data2):
        print ('Error immultiply: Las imágenes de entrada deben tener el mismo tamaño y formato')

    a= a.astype(float)
    s = np.multiply(a,b)
    return s

```

Fig. 14.3 Algoritmo en python

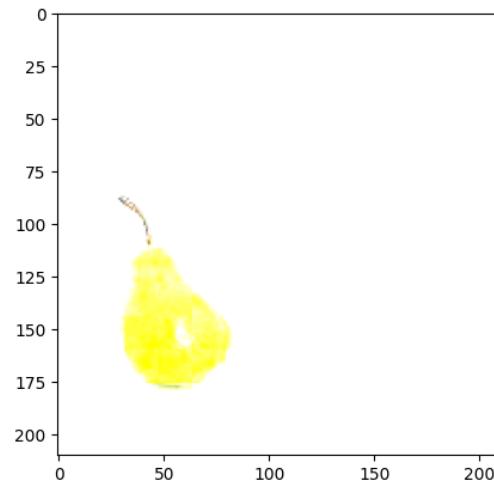
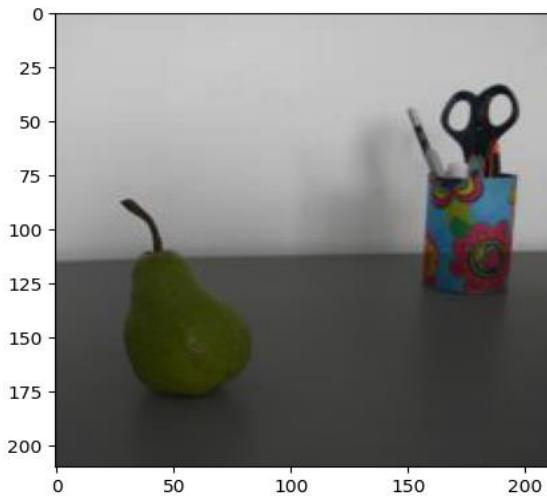


Fig. 14.4 Multiplicación de imagen y multiplicación de imagen por constante⁶.

15. Función imsubtract

15.1. Uso

imsubtract Resta dos imágenes o una constante de un imagen.

La sintaxis para su uso es:

- `imsubtract(a,b);`

Donde: “a” y “b” son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato. Cuando la entrada “b” se le asigna cualquier valor “S”, se realiza la resta entre la imagen “a” y el valor ingresado.

Retorna $s =$ La resta de una imagen de otra.

15.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imsubtract*:

⁶ Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

1. Se lee una imagen llamada “fondo.jpg” y se almacena en una variable “A” y se lee una imagen llamada “pera.jpg”, se almacena en una variable “B”; y ambas variables se dividen por 255 para que la matriz quede en un rango entre “0 y 1”:

- A = imread ('fondo.jpg')/255
- B = imread ('pera.jpg')/255

2. Se usa la instrucción *imsubtract* para restar las imágenes y se almacena en una variable “C”

- C = imsubtract (A,B)

3. Se muestra la resta de las dos imágenes con la instrucción *imshow*

- imshow(C)

15.3. Resultado

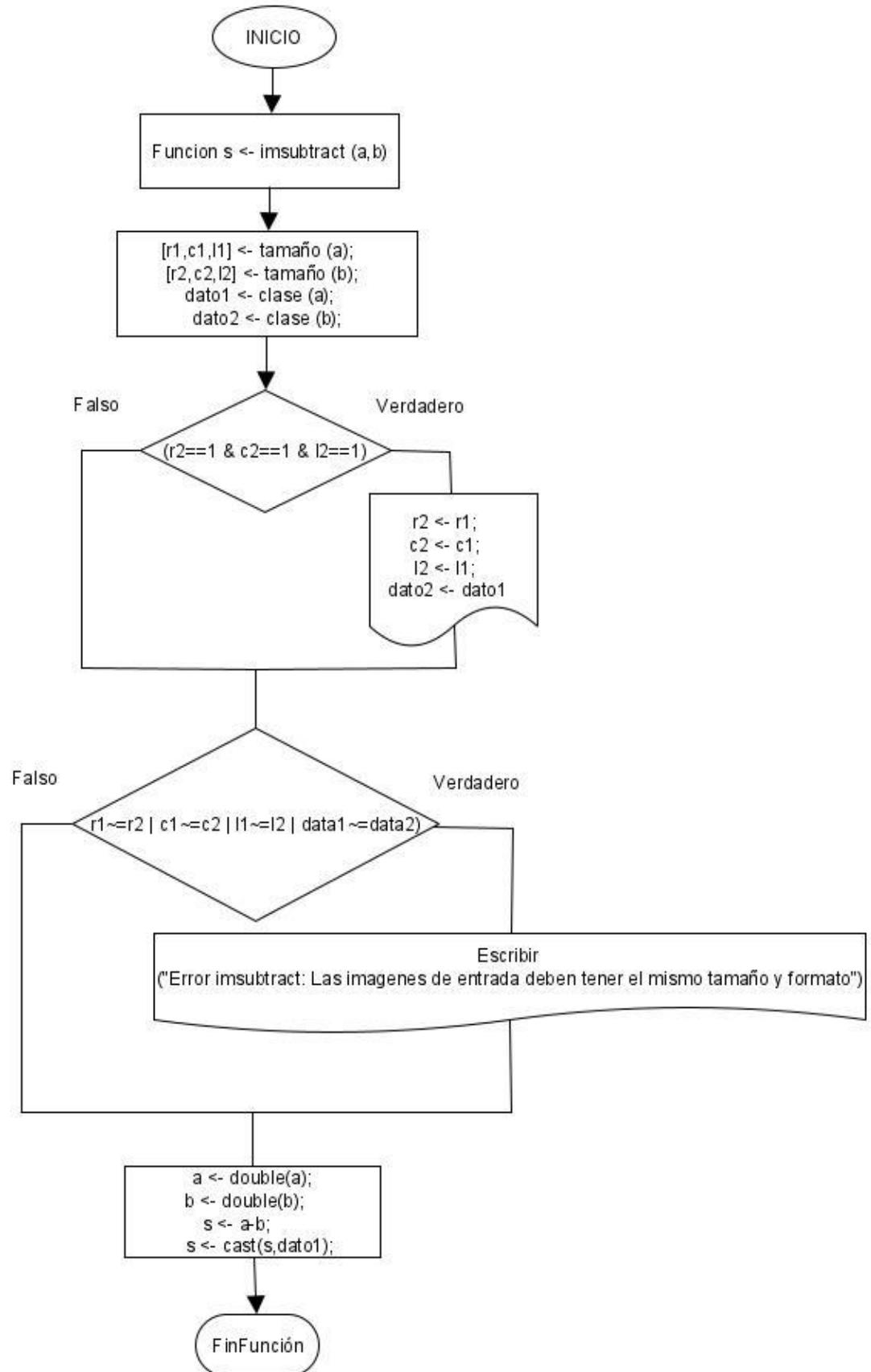


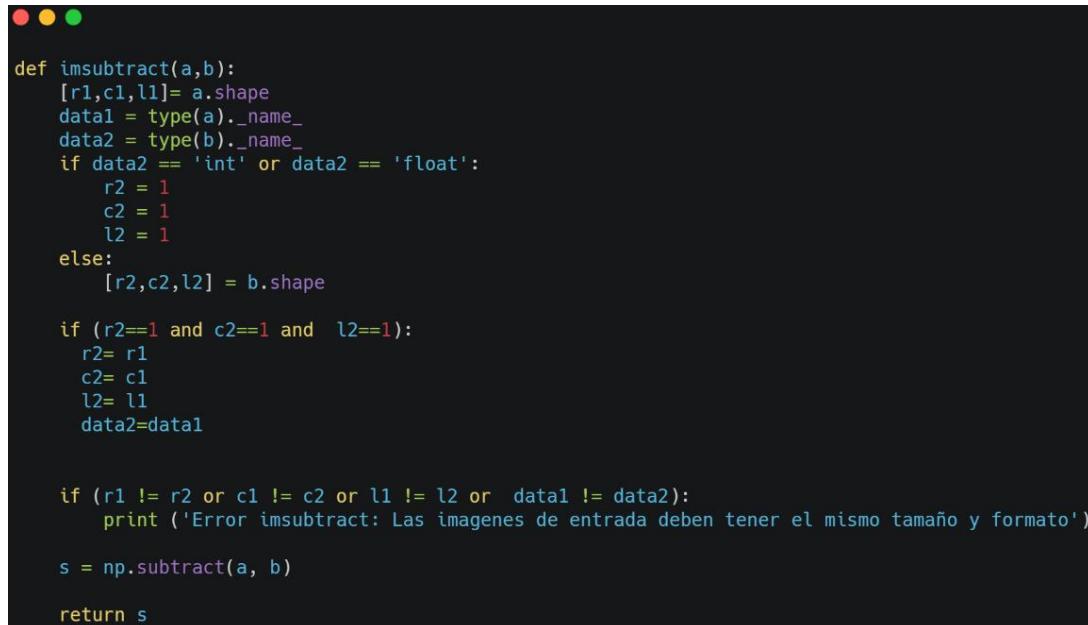
Fig. 15.1. Diagrama de flujo de la función.

```

1 //Nombre: imsubtract.
2 //Parámetros: Donde a y b son imágenes, ambas imágenes deben tener el mismo tamaño y el mismo tipo de dato.
3 // Cuando la entrada b es un valor numérico (s), se realiza la resta entre la imagen a y el valor numérico.
4 // Devuelve la diferencia en el elemento correspondiente de la matriz de salida.
5
6 Algoritmo imsubtract
7 Función s <- imsubtract (a,b)
8     [r1,c1,l1] <- tamaño (a);
9     [r2,c2,l2] <- tamaño (b);
10    datol <- clase (a);
11    dato2 <- clase (b);
12
13    Si (r2==1 & c2==1 & l2==1) //constante
14        r2 <- r1;
15        c2 <- c1;
16        l2 <- l1;
17        dato2 <- datol;
18    FinSi
19
20
21    Si (r1==r2 | c1==c2 | l1==l2 | data1==data2)
22        Escribir ("Error imsubtract: Las imágenes de entrada deben tener el mismo tamaño y formato")
23    FinSi
24
25
26    a <- double(a);
27    b <- double(b);
28    s <- a-b;
29    s <- cast(s,datol);
30
31 FinFunción
32
33 FinAlgoritmo
34
35 //NOTA:
36 // double: Conversión a un número real.
37 //cast: Convierte variable S a diferente tipo de datos.

```

Fig. 15.2. Función en pseudolenguaje pse



```

def imsubtract(a,b):
    [r1,c1,l1]= a.shape
    data1 = type(a).__name__
    data2 = type(b).__name__
    if data2 == 'int' or data2 == 'float':
        r2 = 1
        c2 = 1
        l2 = 1
    else:
        [r2,c2,l2] = b.shape

    if (r2==1 and c2==1 and l2==1):
        r2= r1
        c2= c1
        l2= l1
        data2=data1

    if (r1 != r2 or c1 != c2 or l1 != l2 or data1 != data2):
        print ('Error imsubtract: Las imágenes de entrada deben tener el mismo tamaño y formato')

    s = np.subtract(a, b)

    return s

```

Fig. 15.3. Algoritmo en Python



Fig. 15.4. Resta de dos imágenes en Python⁷

16. Función imadjust

16.1. Uso

imadjust Ajusta los valores de intensidad de una imagen.

La sintaxis para su uso es:

- `imadjust(r, E, S, n)`

Donde “r” es la imagen que se ajustará.

Retorna S= La imagen ajustada.

Los parámetros opcionales son:

- **Ex:** Tiene asignado un vector de dos valores entre 0 y 1 para ajustar los valores de entrada del histograma de la imagen “r” [Emin, Emax].
- **Sx:** Tiene asignado un vector de dos valores entre 0 y 1 para ajustar los valores de salida del histograma de la imagen “r” [Smin, Smax].

⁷ Imagen (a), (b), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

- **n**: Tiene asignado un vector de un valor numérico mayor que cero y menor que 10 para ajustar los valores de la curva del histograma de la imagen “r” y este se le conoce como Gamma.

16.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imadjust*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”.
 - `A = imread ('cartagena.jpg')`
2. Se cambia la imagen a escala de grises y se almacena en la variable “gris”.
 - `gris = rgb2gray (A)`
3. Se ajusta la imagen en escala de grises y se almacena en una variable “S”
 - `S = imadjust (gris,[0.2,0.8],[0,1],1)`
4. Se imprime el resultado usando la instrucción *imshow*:
 - `imshow (S)`

16.3. Resultado.

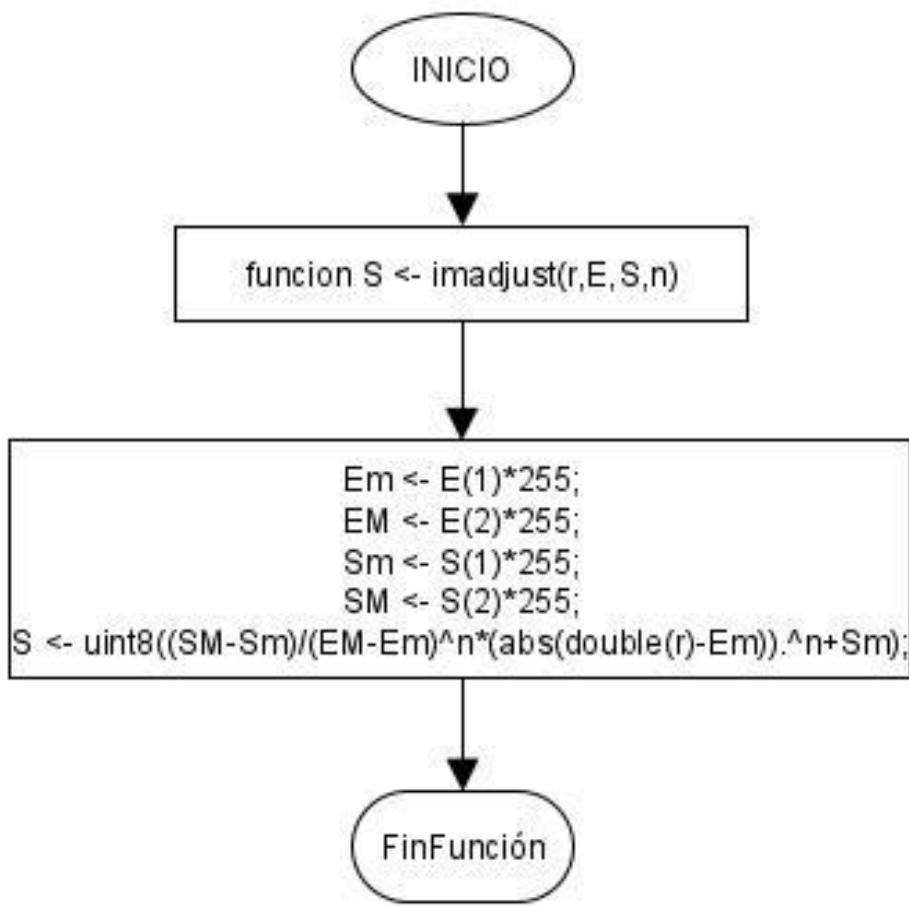


Fig. 16.1. Diagrama de flujo de la función

```

1 //Nombre: imadjust
2 //PARÁMETROS: E : Ajusta los valores de entrada del histograma de la imagen I, es un vector de 2 valores numéricos [Emin,Emax].
3 //           S : Ajusta los valores de salida del histograma de la imagen I, es un vector de 2 valores numéricos [Smin,Smax].
4 //           n : Ajusta los la curva del histograma de la imagen I, es un valor numérico.
5 // Devuelve el Ajuste del histograma a valores deseados con un gamma n dado.
6 Algoritmo imadjust
7   funcion S <- imadjust(r,E,S,n)
8     Em <- E(1)*255;
9     EM <- E(2)*255;
10    Sm <- S(1)*255;
11    SM <- S(2)*255;
12    S <- uint8((SM-Sm) / (EM-Em)^n*(abs(double(r)-Em)).^n+Sm);
13
14 FinAlgoritmo

```

Fig. 16.2. Función en Pseudolenguaje pse.

```
● ● ●  
def imadjust(r,E,Si,n):  
    Em= int(E[0]*255)  
    EM= int(E[1]*255)  
    Sm= int(Si[0]*255)  
    SM= int(Si[1]*255)  
    S= (((SM-Sm)/((EM-Em)*n))((abs(r-Em))**n))+ Sm  
    return S
```

Fig. 16.3. Algoritmo en Python

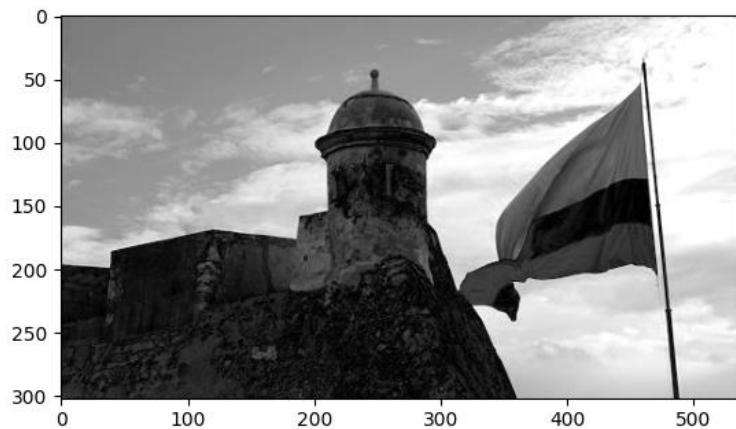


Fig. 16.4. Imagen ajustada en Python.

17. Función stretchlim

17.1. Uso

stretchlim Evalúa los límites superior e inferior del histograma de una imagen.

El límite inferior Emin se evalúa con el 0.01 % del histograma acumulado a la izquierda y el límite superior Emax se evalúa con el 0.99 %.

La sintaxis para su uso es:

- `stretchlim (B)`

Donde “B” es la imagen en escala de grises.

Retorna “E”: Límites del histograma de una imagen.

17.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *stretchlim*.

1. Se lee una imagen llamada 'cartagena.jpg' y se almacena en la variable “A”:
 - `A = imread ('cartagena.jpg')`
2. Se cambia la imagen a escala de grises y se almacena en la variable “gris”.
 - `gris = rgb2gray (A)`
3. Se hace uso de los valores de ajuste de la imagen en escala de grises “Emin” y “Emax” para usar en la función *imadjust*. y se almacena el resultado en la variable “S”
 - `S = imadjust (gris, stretchlim (gris), [0,1],1)`

17.3 Resultado:

```
[0.058823529411764705, 0.996078431372549]
```

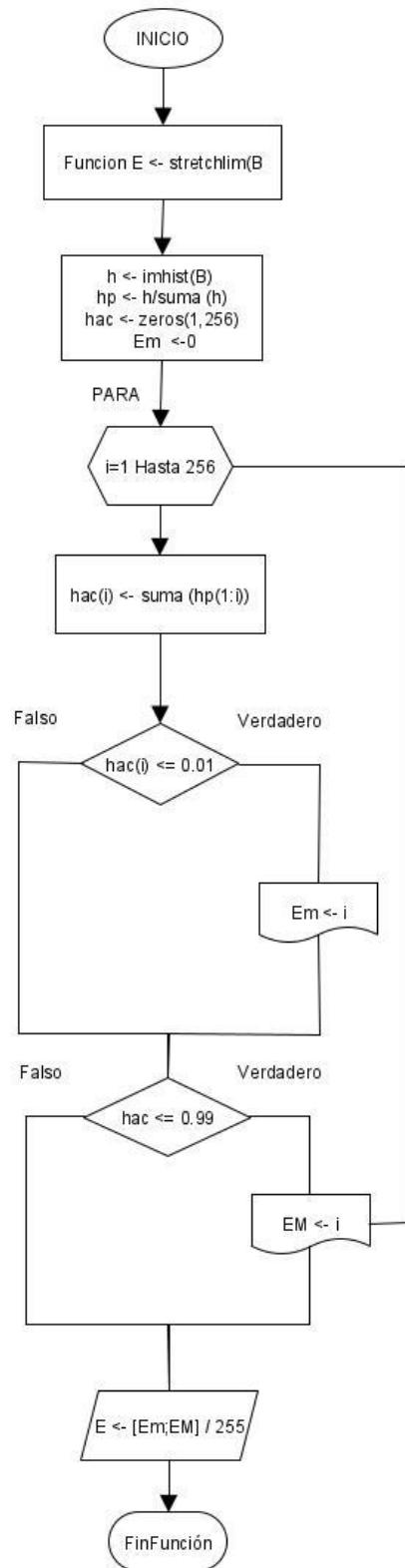


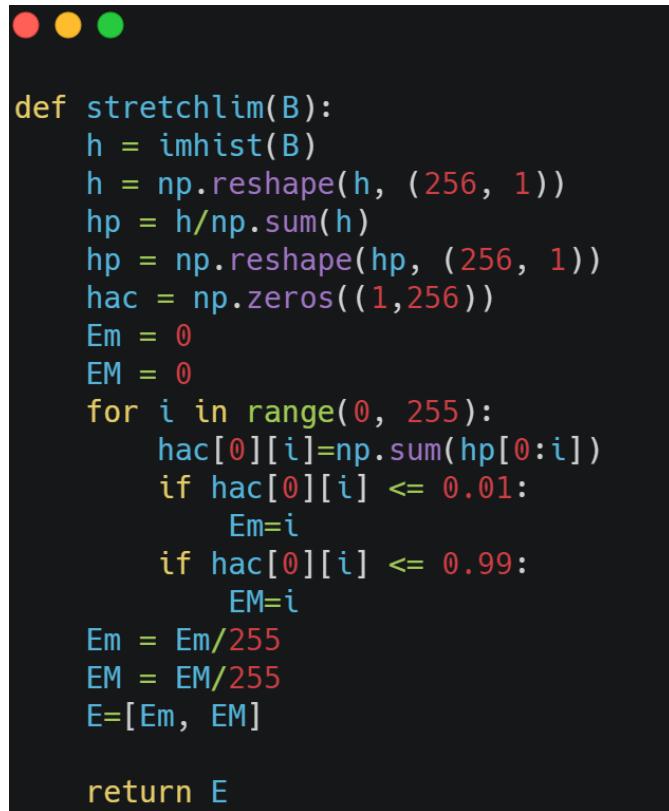
Fig. 17.1. Diagrama de flujo de la función.

```

1 //Nombre: stretchlim
2 // Parámetros: stretchlim: Encuentra los los límites de contraste del histograma de una imagen.
3 //             E: Es un vector de 2 posiciones que contiene los valores del contraste mínimo y máximo [Lmin,Lmax] normalizados.
4 //             B: Es la imagen en escala de grises o una de las capas de la imagen.
5 //Devuelve: Encuentra los los límites de contraste del histograma de una imagen, el límite inferior se calcula
6 //con el 1% del histograma y el límite superior se calcula a partir del 99% del histograma.
7
8 Algoritmo stretchlim
9
10 Funcion E <- stretchlim(B) //Calcula automaticamente los límites Em y EM.
11     h <- imhist(B)
12     hp <- h/suma (h)
13     hac <- zeros(1,256)
14     Em <-0
15     Para i=1 Hasta 256
16         hac(i) <- suma (hp(1:i))
17
18         Si hac(i) <= 0.01
19             Em <- i
20         FinSi
21
22
23         Si hac <= 0.99
24             EM <- i
25         FinSi
26
27     FinPara
28
29     E <- [Em;EM] / 255
30
31     FinFuncion
32 FinAlgoritmo
33
34 //NOTA: Funciones
35
36 //zeros: array de ceros.

```

Fig. 17.2. Función en pseudolenguaje pse



```

def stretchlim(B):
    h = imhist(B)
    h = np.reshape(h, (256, 1))
    hp = h/np.sum(h)
    hp = np.reshape(hp, (256, 1))
    hac = np.zeros((1,256))
    Em = 0
    EM = 0
    for i in range(0, 255):
        hac[0][i]=np.sum(hp[0:i])
        if hac[0][i] <= 0.01:
            Em=i
        if hac[0][i] <= 0.99:
            EM=i
    Em = Em/255
    EM = EM/255
    E=[Em, EM]

    return E

```

Fig. 17.3 Algoritmo en Python.

18. Función imnoise

18.1. Uso

imnoise Añade ruido a una imagen.

La sintaxis para su uso es:

- `imnoise (I, tipo, porcentaje)`

Donde “I” es una imagen, “tipo” es el tipo de ruido, “porcentaje” es el parámetro del ruido.

Retorna In: La imagen con ruido.

Uso de la variable “porcentaje”:

- **“salt & pepper”**: “porcentaje” es la densidad del ruido, es el parámetro numérico entre 0 y 1.
- **“gaussian”**: “porcentaje” es la varianza del ruido, es un parámetro numérico entre 0 y 1.
- **“speckle”**: “porcentaje” es la varianza del ruido, añade ruido multiplicativo utilizando la ecuación, donde se distribuye uniformemente el ruido con la media 0 y 0,05.

18.2 Ejemplo.

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imnoise*

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “A”:
 - `A = imread ('cartagena.jpg')`
2. Se cambia la imagen a escala de grises y se almacena en la variable gris:
 - `gris = rgb2gray (A)`
3. Se usa la instrucción *imnoise* para agregar ruido de tipo 'salt & pepper' y se almacena en una variable “C”.

- C = imnoise (gris, 'salt & pepper', 0.1)

4. Se usa la instrucción *imnoise* para agregar ruido de tipo 'gaussian' y se almacena en una variable "D".

- D = imnoise (gris, 'gaussian' ,0.09)

5. Se usa la instrucción *imnoise* para agregar ruido de tipo 'speckle' y se almacena en una variable "D".

- E = imnoise (gris, 'speckle' ,0.05)

6. Se usa la instrucción *imshow*, para mostrar las tres imágenes con ruido.

- imshow (C)
- imshow (D)
- imshow(E)

18.2. Resultado.

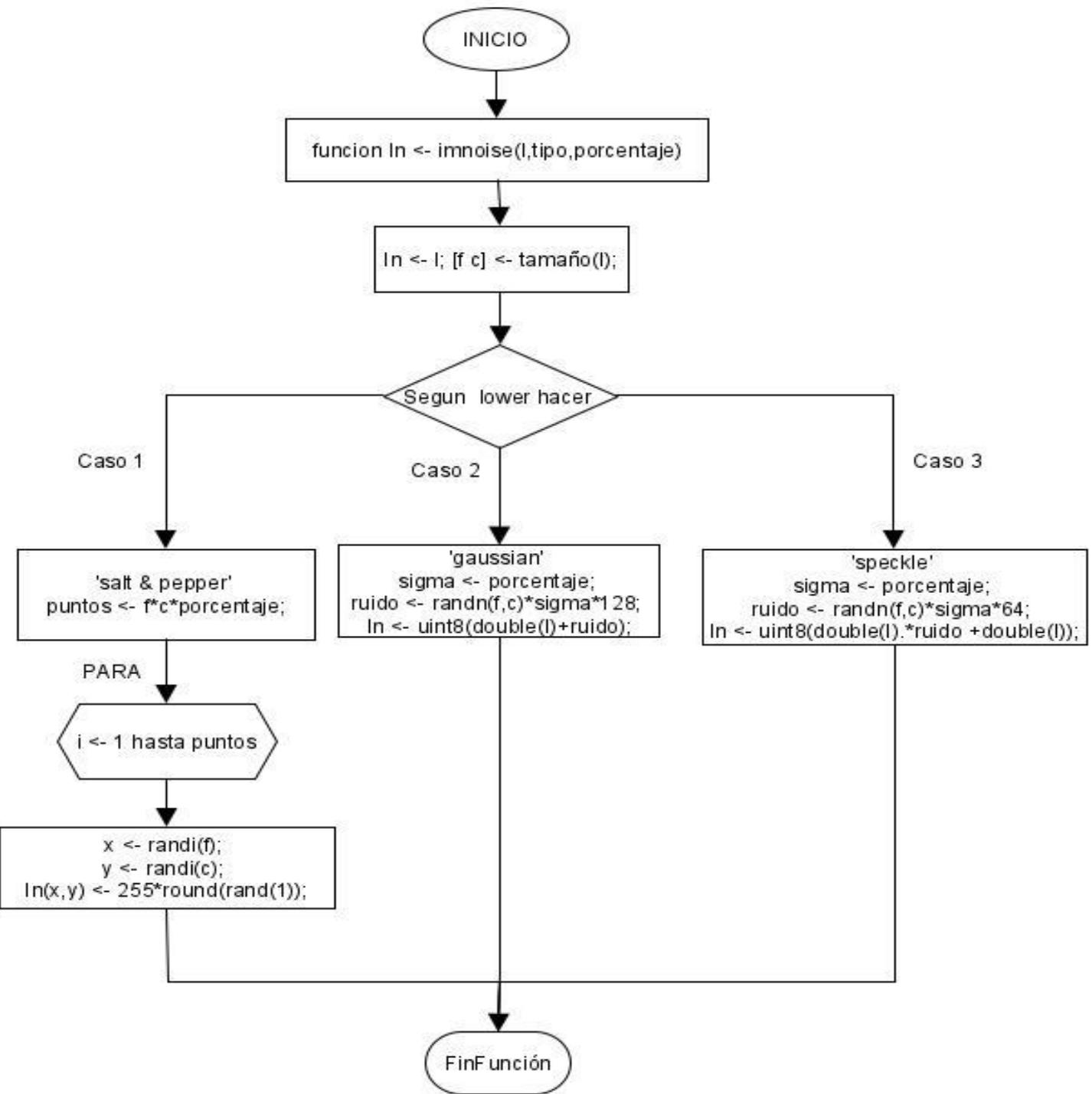


Fig. 18.1. Diagrama de flujo de la función

```

1 //Nombre: imnoise
2 //Parámetros: Donde tipo es el tipo de ruido,
3 //                Porcentaje es el parámetro del ruido
4 //                Ir es la imagen con ruido.
5 //Devuelve: Agrega un tipo de ruido determinado a la imagen.
6
7 Algoritmo imnoise
8 funcion In <- imnoise(I,tipo,porcentaje)
9     In <- I;
10    [f c] <- size(I);
11
12    Segun lower hacer
13        caso 1 : 'salt & pepper'
14            puntos <- f*c*porcentaje;
15
16        Para i <- 1 hasta puntos
17            x <- randi(f);
18            y <- randi(c);
19            In(x,y) <- 255*round(rand(1));
20        FinPara
21
22        caso 2 : 'gaussian'
23            sigma <- porcentaje;
24            ruido <- randn(f,c)*sigma*128;
25            In <- uint8(double(I)+ruido);
26
27        caso 2 : 'speckle'
28            sigma <- porcentaje;
29            ruido <- randn(f,c)*sigma*64;
30            In <- uint8(double(I).*ruido +double(I));
31        FinSegun
32
33 FinFuncion
34
35 FinAlgoritmo

```

Fig. 18.2. Función en pseudolenguaje pse

```

● ● ●

def imnoise(I, tipo, porcentaje):
    In = I
    [f, c] = I.shape

    tipologia = tipo.lower()

    if tipologia == 'salt&pepper':
        puntos = f*c*porcentaje
        i = 1
        for i in range(int(puntos)):
            x = np.random.randint(1,f)
            y = np.random.randint(1,c)
            In[x,y] = 255*np.random.rand(1)
            In = In.astype(int)

    if tipologia == 'gaussian':
        sigma = porcentaje
        ruido = np.random.randn(f,c)*sigma*128
        In = I+ruido

    if tipologia == 'speckle':
        sigma = porcentaje
        ruido = np.random.random((f,c))*sigma*0.5
        In = I*ruido+I

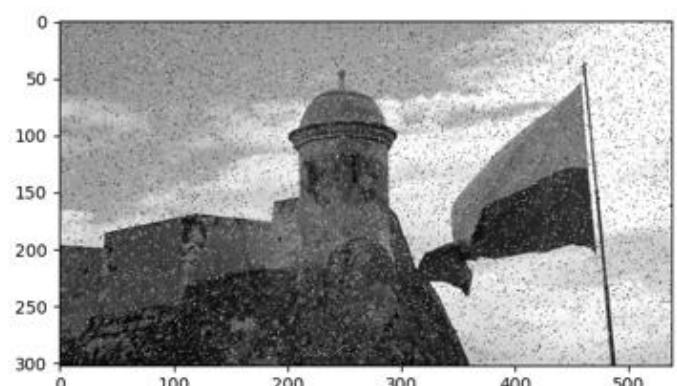
    return In

```

Fig. 18.3. Algoritmo en Python



(a) Ruido Salt&pepper



(b) Ruido Gaussian

Fig. 18.4. Imagen con ruido Salt & pepper , Gaussian y speckle en Python.

19. Función medfilt2

19.1. Uso

medfilt2 Realiza un filtrado promedio a una imagen en 2D.

La sintaxis para su uso es:

- `medfilt2(Gris,w)`

Donde “Gris” es la imagen en escala de grises, “w” es un vector de dos valores numéricos que define el tamaño de la ventana, es [3,3].

Retorna “T”: La imagen filtrada.

19.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *medfilt2*:

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en una variable “I”:
 - `I = imread ('cartagena.jpg')`
2. Se cambia la imagen a escala de grises y se almacena en una variable “B”:
 - `B = rgb2gray (I)`
3. Se usa la instrucción *imnoise* para agregar tipo de ruido 'salt & pepper' y se almacena en una variable “C”.
 - `C = imnoise (B,'salt & pepper',0.02)`
4. Se usa la instrucción *medfilt2* para filtrar el ruido de tipo 'salt & pepper' y se almacena en una variable “D”:
 - `D = medfilt2 (C,[3,3])`
5. Se muestra las dos imágenes con la instrucción *imshow*
 - `imshow (C)`
 - `imshow (D)`

19.3. Resultado

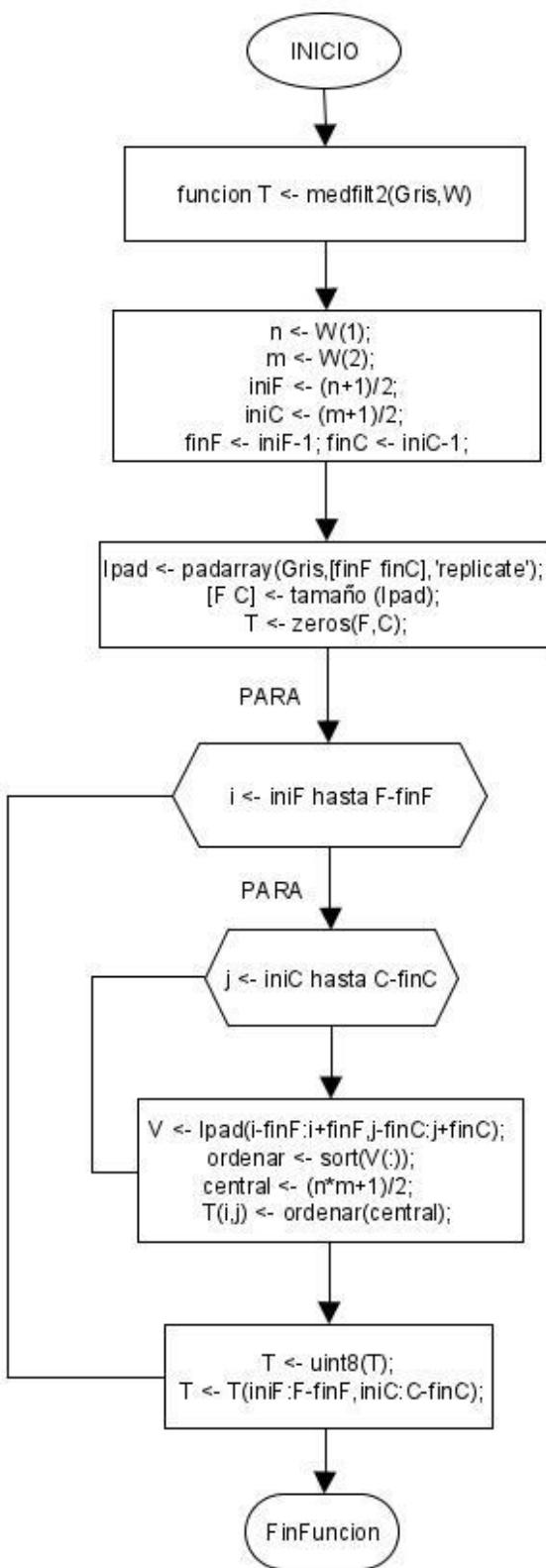


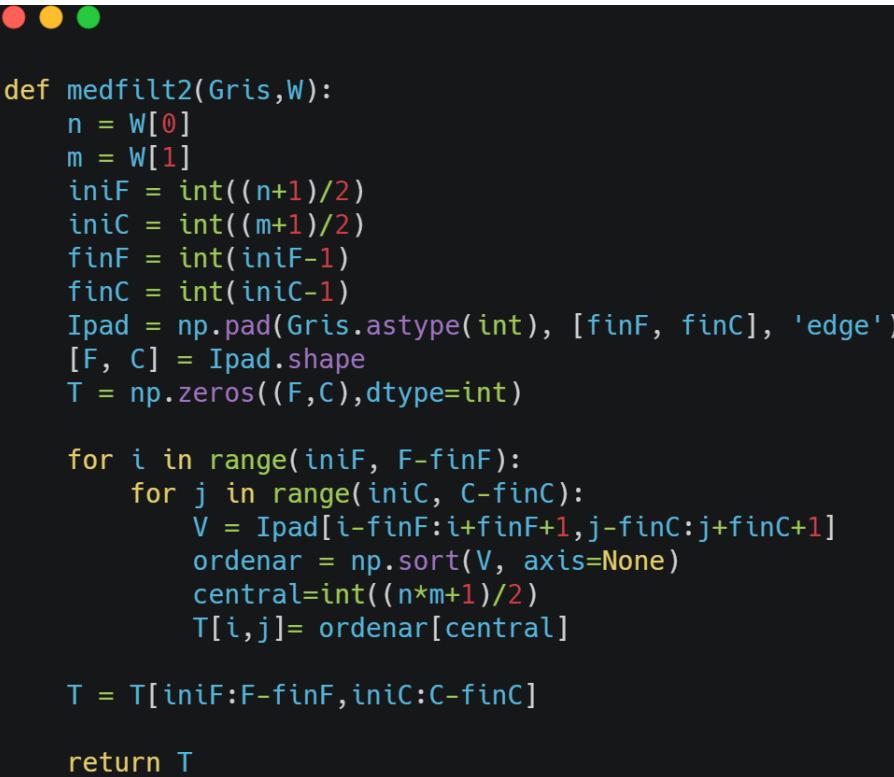
Fig. 19.1. Diagrama de flujo de la función.

```

1 //Nombre: medfilt2
2 //Parámetros: Gris imagen escala de grises
3 //          W es un vector de 2 valores numéricos que determina el tamaño de la ventana, por defecto es [3,3].
4 //          T es la imagen filtrada.
5 //Devuelve:
6
7
8 Algoritmo medfilt2
9     funcion T <- medfilt2(Gris,W)
10    n <- W(1);
11    m <- W(2);
12    iniF <- (n+1)/2;
13    iniC <- (m+1)/2;
14    finF <- iniF-1;
15    finC <- iniC-1;
16
17    Ipad <- padarray(Gris,[finF finC],'replicate');
18    [F C] <- tamaño (Ipad);
19    T <- zeros(F,C);
20
21    Para i <- iniF hasta F-finF
22        para j <- iniC hasta C-finC
23            V <- Ipad(i-finF:i+finF,j-finC:j+finC);
24            ordenar <- sort(V(:));
25            central <- (n*m+1)/2;
26            T(i,j) <- ordenar(central);
27        FinPara
28    FinPara
29
30    T <- uint8(T);
31    T <- T(iniF:F-finF,iniC:C-finC);
32
33 FinFuncion
34 FinAlgoritmo

```

Fig. 19.2. Función en pseudolenguaje pse



```

def medfilt2(Gris,W):
    n = W[0]
    m = W[1]
    iniF = int((n+1)/2)
    iniC = int((m+1)/2)
    finF = int(iniF-1)
    finC = int(iniC-1)
    Ipad = np.pad(Gris.astype(int), [finF, finC], 'edge')
    [F, C] = Ipad.shape
    T = np.zeros((F,C),dtype=int)

    for i in range(iniF, F-finF):
        for j in range(iniC, C-finC):
            V = Ipad[i-finF:i+finF+1,j-finC:j+finC+1]
            ordenar = np.sort(V, axis=None)
            central=int((n*m+1)/2)
            T[i,j]= ordenar[central]

    T = T[iniF:F-finF,iniC:C-finC]

    return T

```

Fig. 19.3. Algoritmo en Python

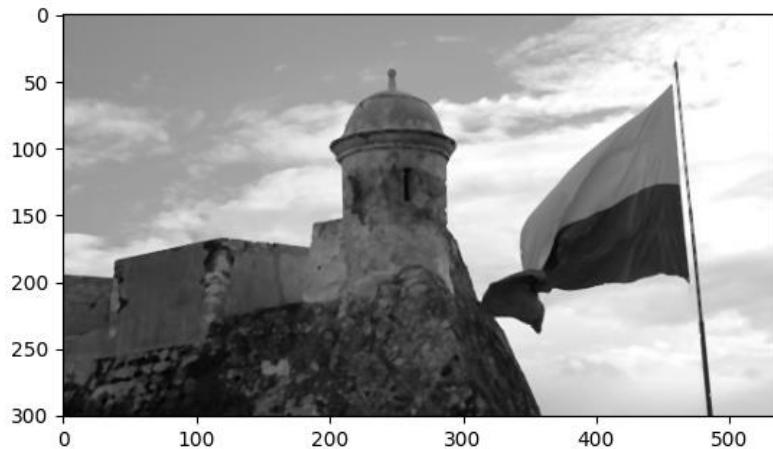


Fig. 19.4 Imagen filtrada en Python.

20. Función ordfilt2

20.1. Uso

ordfilt2 Realiza un filtrado estadístico a una imagen 2D.

La sintaxis para su uso es:

- *ordfilt2 (Gris,termino,K)*

Donde “Gris” es la imagen en escala de grises, “termino” es el elemento que se reemplaza de la ventana el cual es un parámetro numérico entero, “K” es una matriz “double”.

Retorna T: La imagen filtrada.

20.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *ordfilt2*

1. Se lee la imagen 'cartagena.jpg' y se almacena en la variable "I".

- `I = imread ('cartagena.jpg')`

2. Se cambia la imagen a escala de grises y se almacena en la variable "Ic".

- `Ic = rgb2gray (I)`

3. Se usa la instrucción `imnoise` para agregar tipo de ruido "salt & pepper" a la imagen "Ic" y se almacena en la variable "Ir"

- `Ir = imnoise (Ic,'salt & pepper',0.02)`

4. Se realiza el filtrado mínimo 2D y se almacena en una variable "Imed".

- `Imed = ordfilt2 (Ir,1,np.ones(3,3), dtype=int))`

5. Se muestra la imagen con filtrado estadístico con la instrucción `imshow`.

- `imshow (Imed)`

20.3. Resultado

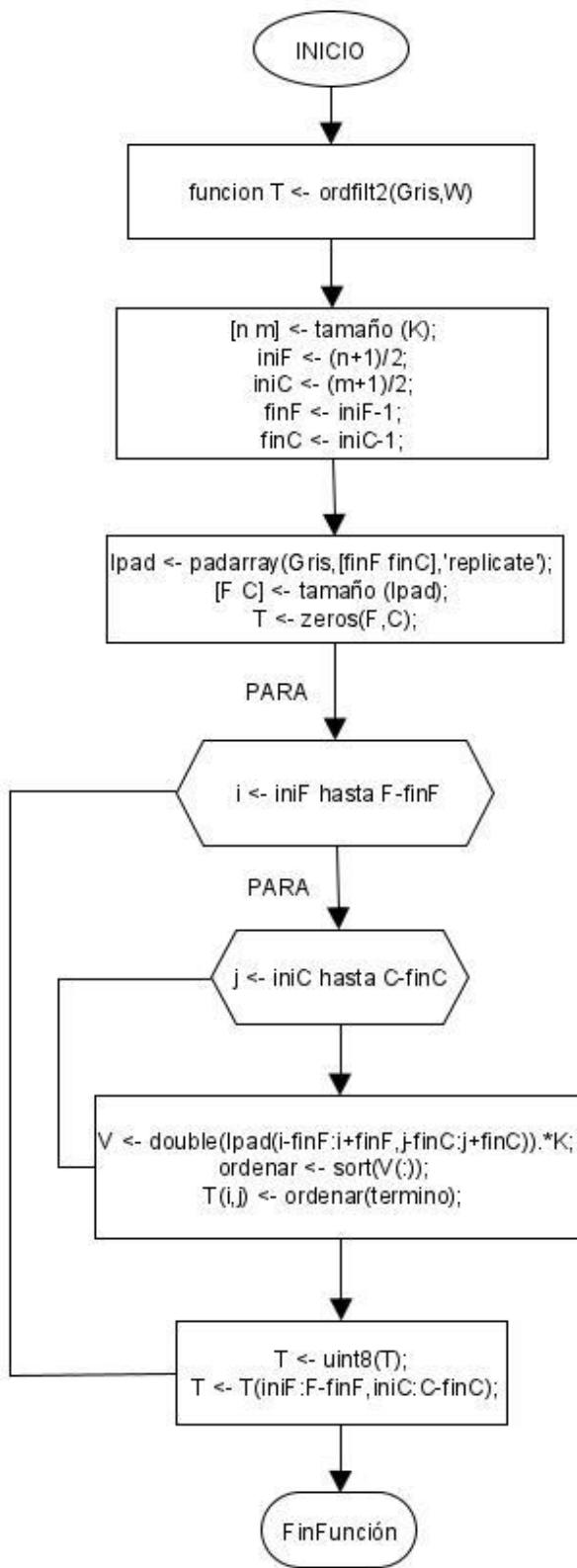


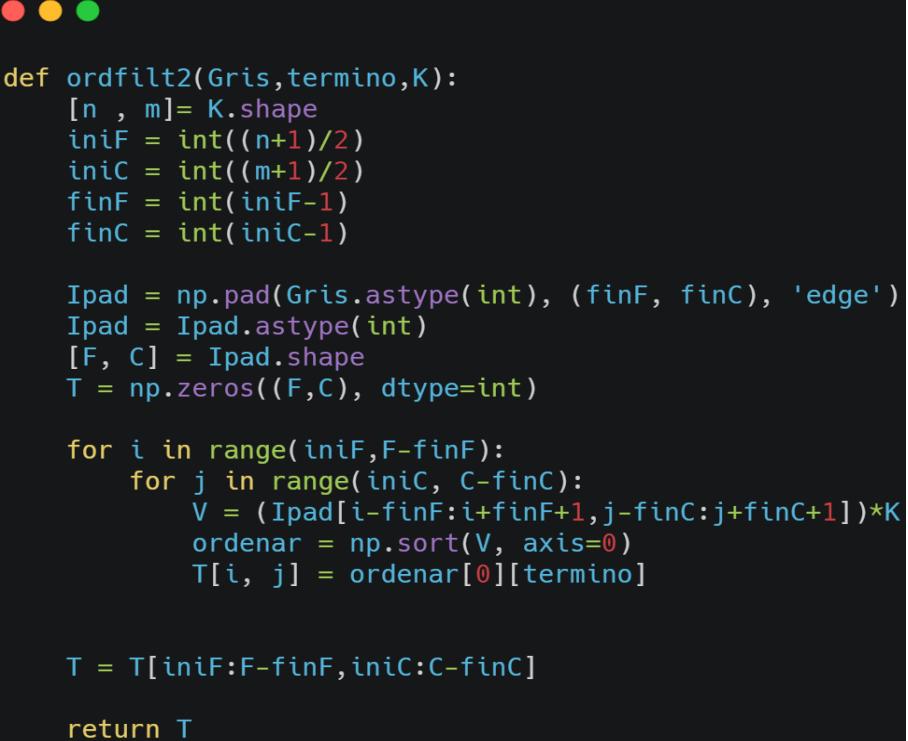
Fig. 20.1. Diagrama de flujo de la función.

```

1 //Nombre: ordfilt2
2 //Parámetros: Gris imagen escala de grises
3 //              termino especificado como una matriz numérica o lógica, que contiene 1s y 0s
4 //Devuelve: Filtrado de estadísticas de óxos 2-D
5
6 Algoritmo ordfilt2
7     función T <- ordfilt2(Gris,termino,K)
8         [n m] <- tamaño (K);
9         iniF <- (n+1)/2;
10        iniC <- (m+1)/2;
11        finF <- iniF-1;
12        finC <- iniC-1;
13
14        Ipad <- padarray(Gris,[finF finC],'replicate');
15        [F C] <- tamaño (Ipad);
16        T <- zeros(F,C);
17
18        Para i=iniF hasta F-finF
19            para j <- iniC hasta C-finC
20                V <- double(Ipad(i-finF:i+finF,j-finC:j+finC)).*K;
21                ordenar <- sort(V(:));
22                T(i,j) <- ordenar(termino);
23            FinPara
24        FinPara
25
26        T <- uint8(T);
27        T <- T(iniF:F-finF,iniC:C-finC);
28    FinFunción
29 FinAlgoritmo

```

Fig. 20.2. Función en pseudolenguaje pse



```

def ordfilt2(Gris,termino,K):
    [n , m]= K.shape
    iniF = int((n+1)/2)
    iniC = int((m+1)/2)
    finF = int(iniF-1)
    finC = int(iniC-1)

    Ipad = np.pad(Gris.astype(int), (finF, finC), 'edge' )
    Ipad = Ipad.astype(int)
    [F, C] = Ipad.shape
    T = np.zeros((F,C), dtype=int)

    for i in range(iniF,F-finF):
        for j in range(iniC, C-finC):
            V = (Ipad[i-finF:i+finF+1,j-finC:j+finC+1])*K
            ordenar = np.sort(V, axis=0)
            T[i, j] = ordenar[0][termino]

    T = T[iniF:F-finF,iniC:C-finC]

    return T

```

Fig. 20.3 Algoritmo en Python

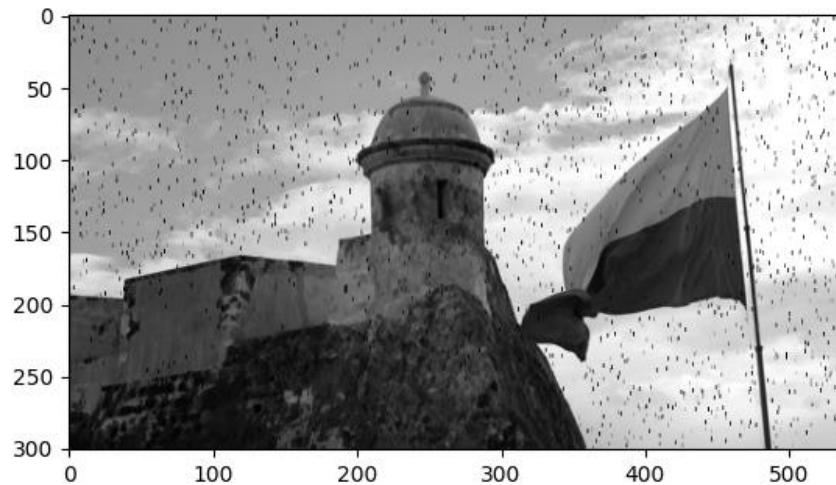


Fig. 20.4 Imagen con filtrado estadístico.

21. Función fspecial

21.1. Uso

fspecial Crea un kernel (máscara) de correlación para filtrar una imagen 2D.

La sintaxis para su uso es:

- `fspecial (kwargs)`

Donde “`kwargs`” es el kernel.

Retorna “`k`”: Filtrado de una imagen.

Uso de la variable “`n`”:

- En el filtro “average”, la máscara debe ser de “`nxn`” y ser una matriz de unos multiplicada por el recíproco del cuadrado de “`n`”.
- En el filtro “laplaciano”, “`n`” especifica el tipo de “laplaciano” a evaluar (variable alpha).
- En el filtro “gaussiano”, “`n`” es el parámetro que corresponde a la desviación estándar y este filtro toma el tamaño del kernel con este valor.

El tipo kwargs (kernel) debe especificarse así:

- '**average**': Máscara para filtro por promedio (blur), este valor por defecto es un vector de [3,3].
 -->kernel = fspecial ('average', tamaño)
- '**laplacian**': Filtro “laplaciano”, pasa alto derivativo (detector de bordes y detalles), alpha es un valor entre 0 y 1, este valor están dados por defecto de 0.2.
 -->kernel = fspecial ('laplacian', alpha)
- '**gaussian**': Filtro gaussiano pasa bajo, su tamaño es un escalar y sigma un valor entre 0 y 1, estos valores están dados por defecto de [3,3] y sigma 0.5.
 -->kernel = fspecial ('gaussian', tamaño, sigma)

FILTROS PARA ALTOS DERIVATIVOS DIRECCIONALES:

- '**sobel**': Detecta líneas horizontales.
 -->kernel = fspecial ('sobel')
- '**prewitt**': Filtro direccional.
 -->kernel = fspecial ('prewitt')

21.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *fspecial*:

1. Se crea un kernel de filtro promedio en “K2” y así para cada uno de los tipos de kernel.

- K2 = fspecial (tipo= 'average')

21.3. Resultado

K2 =

0,1111 0,1111 0,1111
0,1111 0,1111 0,1111

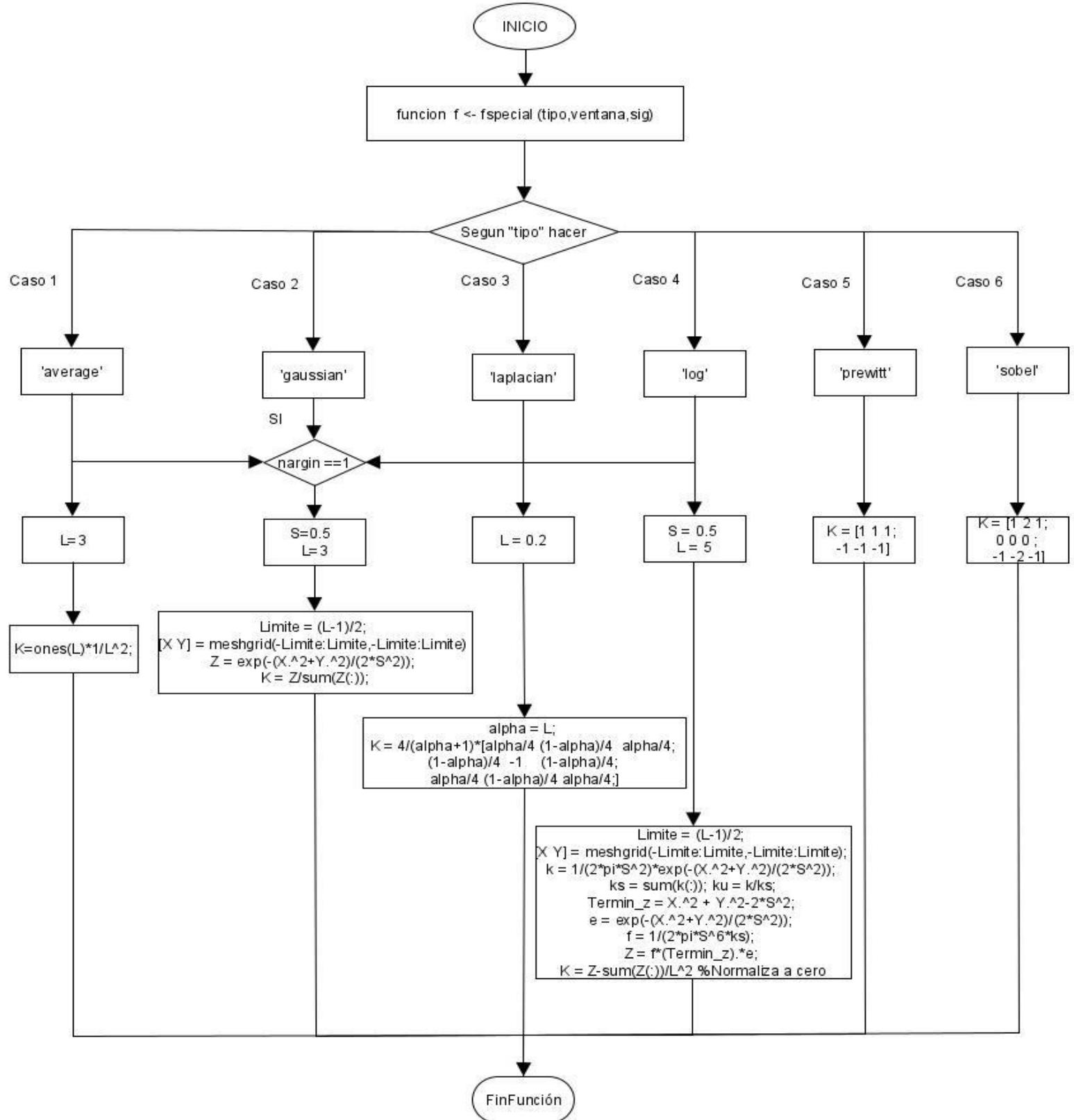


Fig. 21.1. Diagrama de flujo de la función

```

1 //Nombre: fspecial
2 //Parámetros: tipo: Es el kernel
3 //          Ventana Es la ventana.
4 //          sig: Desviación estándar.
5 Algoritmo fspecial
6     función f <- fspecial (tipo,ventana,sig)
7         Segun tipo hacer
8             caso 1: 'average'
9                 Si nargin ==1
10                    L=3
11                    FinSi
12
13                 K=ones(L)*1/L^2;
14
15                 caso 2: 'gaussian'
16                     si nargin ==1
17                         S=0.5
18                         L=3
19                     FinSi
20
21                 Limite = (L-1)/2;
22                 [X Y] = meshgrid(-Limite:Limite,-Limite:Limite)
23                 Z = exp(-(X.^2+Y.^2)/(2*S^2));
24                 K = Z/sum(Z(:));
25
26                 caso 3: 'laplacian'
27                     si nargin ==1
28                         L = 0.2
29                     FinSi
30
31                 alpha = L;
32                 K = 4/(alpha+1)*[alpha/4 (1-alpha)/4 alpha/4;
33                               (1-alpha)/4 -1 (1-alpha)/4;
34                               alpha/4 (1-alpha)/4 alpha/4];
35
36
37                 caso 4: 'log'
38                     Si nargin ==1
39                         S = 0.5
40                         L = 5
41                     FinSi
42
43                     Limite = (L-1)/2;
44                     [X Y] = meshgrid(-Limite:Limite,-Limite:Limite);
45                     k = 1/(2*pi*S^2)*exp(-(X.^2+Y.^2)/(2*S^2));
46                     ks = sum(k(:));
47                     ku = k/ks;
48                     Termin_z = X.^2 + Y.^2-2*S.^2;
49                     e = exp(-(X.^2+Y.^2)/(2*S^2));
50                     f = 1/(2*pi*S^6*ks);
51                     Z = f*(Termin_z).*e;
52                     K = Z-sum(Z(:))/L^2 %Normaliza a cero
53
54                 caso 5: 'prewitt'
55                     K = [1 1 1;
56                           0 0 0 ;
57                           -1 -1 -1];
58
59                 caso 6: 'sobel'
60                     K = [1 2 1;
61                           0 0 0 ;
62                           -1 -2 -1];
63             FinSegun
64         Finfuncion
65     FinAlgoritmo
66
67 //NOTA: Funciones.
68 //nargin: Determina el número de argumentos de entrada.
69 //imrotate: Rotar imagen.
70 //average: Máscara para filtro por promedio (blur), tamaño es un vector [N,N], por defecto es [3,3].
71 //gaussian: filtro gaussiano pasa bajo, tamaño es un escalar y sigma un valor entre 0 y 1, por defecto tamaño es de [3,3] y sigma 0.5.
72 //laplacian: Filtro laplaciano, pasa alto derivativo (detector de bordes y detalles), alpha es un valor entre 0 y 1, por defecto es 0.2.
73 //meshgrid: Tamaño de la cuadricula.
74 //prewitt: Filtro horizontal de enfatizar bordes Prewitt
75 //sobel: filtro direccional que detecta (muestra) líneas horizontales.
76 //motion: Kernel de difuminación, longitud es un escalar y angulo un valor en grados.

```

Fig. 21.2. Función en pseudolenguaje pse.

```
def fspecial(**kwargs):
    tipologia = kwargs['tipo']

    if tipologia == 'average':
        if len(kwargs) == 1:
            L = 3
        K = np.ones((L,L))/L**2

    if tipologia == 'gaussian':
        if len(kwargs) == 1:
            S = 0.5
            L = 3
        Limite = (L-1)/2
        [X, Y] = np.meshgrid(-Limite,Limite)
        Z = math.exp(-(np.dot(X,2)+np.dot(Y,2))/(2*(S**2)))
        K = Z/sum(Z)

    if tipologia == 'laplacian':
        if len(kwargs) == 1:
            L = 0.2
            alpha = L

        K=4/(alpha+1)*[[alpha/4, (1-alpha)/4, alpha/4],
                      [(1-alpha)/4, -1, (1-alpha)/4],
                      [alpha/4, (1-alpha)/4, alpha/4]]

    if tipologia == 'log':
        if len(kwargs):
            S = 0.5
            L = 5
        Limite = (L-1)/2
        [X, Y] = np.meshgrid(-Limite,Limite)
        k = 1/(2*math.pi*S**2)math.exp(-(np.dot(X,2)+np.dot(Y,2))/(2*(S**2)))
        ks = sum(k)
        Termin_z = np.dot(X,2) + np.dot(Y,2) - 2*(S**2)
        e = math.exp(-(np.dot(X,2) + np.dot(Y,2))/(2*(S**2)))
        f = 1/(2*math.pi*(S**6)*ks)
        Z = np.dot(f*(Termin_z), e)
        K = Z-sum(Z)/L^2

    if tipologia == 'prewitt':
        K = [[1, 1, 1],
              [0, 0, 0],
              [-1, -1, -1]] 

    if tipologia == 'sobel':
        K = [[1, 2, 1],
              [0, 0, 0],
              [-1, -2, -1]] 

    return K
```

Fig. 21.3 Algoritmo en Python.

22. Función strel

22.1. Uso

strel Crea un elemento estructurador morfológico.

La sintaxis para su uso es:

- `strel ('tipo', forma)`

Donde “tipo” es el tipo de elemento estructurador y “forma” es el parámetro para configurar el elemento.

Retorna `f`: Matriz del elemento estructurador.

Uso de la variable “forma”:

- '**disk**', '**lineV**' y '**lineH- '**square**' y '**cross[f,c]) o un valor numérico que define el tamaño de la matriz.****

22.2. Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *strel*:

1. Se crea un elemento estructurador tipo 'cross' y 'lineV'.

- `EE = strel ('cross',5)`
- `EE1 = strel ('lineV',5)`

22.3. Resultado

```
0 0 1 0 0  
0 0 1 0 0  
0 0 1 0 0  
0 0 1 0 0  
0 0 1 0 0
```

Elemento Estructurante tipo 'lineV'

0 0 1 0 0

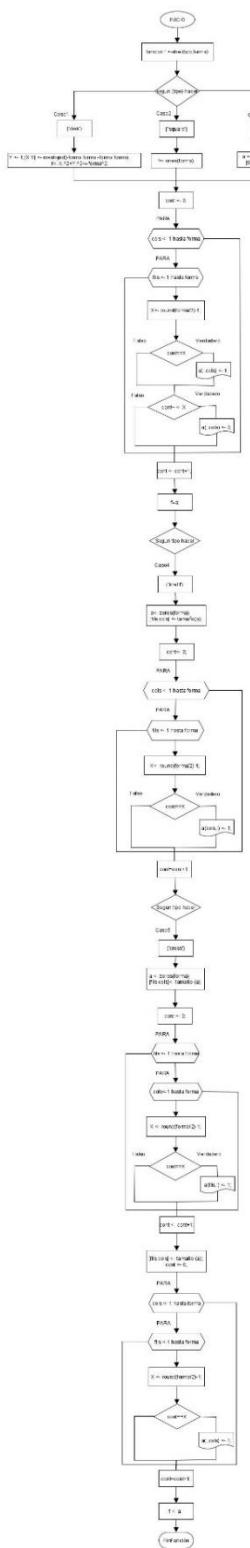
0 0 1 0 0

1 1 1 1 1

0 0 1 0 0

0 0 1 0 0

Elemento Estructurante tipo 'cross'



```
1 //Nombre: Strel
2 //Parámetros: Donde "tipo" es el tipo de elemento estructurante y
3 //           "forma" es el parámetro de configuración del elemento.
4
5 Algoritmo strel
6
7 función f <-strel(tipo,forma)
8
9     segun (tipo) hacer
10
11         caso 1: {'disk'}
12             Y <- 1;[X Y] <- meshgrid(-forma:forma,-forma:forma);
13             f<- X.^2+Y.^2<=forma^2;
14
15         caso 2: {'square'}
16             f<- ones(forma);
17
18         caso 3: {'lineV'}
19             a <- zeros(forma);
20             [fils cols] <- tamaño (a);
21             cont <- 0;
22
23             Para cols <- 1 hasta forma
24                 Para fils <- 1 hasta forma
25                     X<- round(forma/2)-1;
26
27                     si cont==X
28                         a(:,cols) <- 1;
29                         FinSi
30
31                     si cont~ <- X
32                         a(:,cols) <- 0;
33                         FinSi
34                     FinPara
35
36             cont <- cont+1;
37             FinPara
```

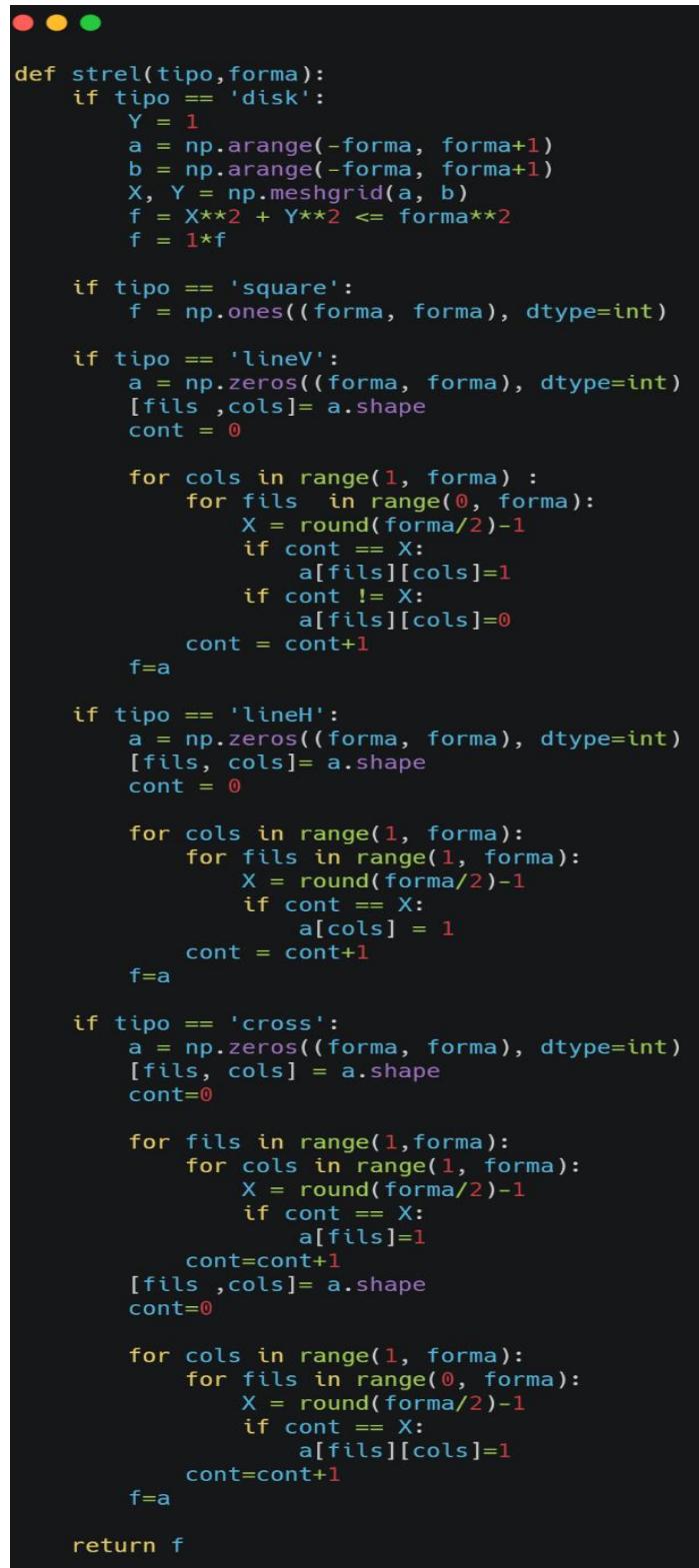
```

39         f=a;
40
41     caso 4: {'lineH'}
42         a<- zeros(forma);
43         [fils cols] <- tamaño(a);
44         cont<- 0;
45
46     para cols <- 1 hasta forma
47         para filas <- 1 hasta forma
48             X<- round(forma/2)-1;
49
50             si cont==X
51                 a(cols,:)<- 1;
52                 FinSi
53             FinPara
54
55             cont=cont+1;
56         FinPara
57
58     f<-a;
59
60     caso 5 {'cross'}
61         a <- zeros(forma);
62         [fils cols]<- tamaño (a);
63         cont <- 0;
64
65     Para filas <- 1 hasta forma
66         para cols<-1 hasta forma
67             X <- round(forma/2)-1;
68
69             si cont==X
70                 a(filas,:)<- 1;
71                 FinSi
72             FinPara
73
74             cont <- cont+1;
75         FinPara
76         [fils cols] <- tamaño (a);
77         cont <- 0;
78
79
80     para cols <- 1 hasta forma
81         para filas <-1 hasta forma
82             X <- round(forma/2)-1;
83
84             si cont==X
85                 a(:,cols)<- 1;
86                 FinSi
87
88             FinPara
89
90             cont=cont+1;
91         FinPara
92
93         f <- a;
94     FinSegun
95
96 FinAlgoritmo

```

Fig. 22.2. Función en pseudolenguaje pse

```


def strel(tipo, forma):
    if tipo == 'disk':
        Y = 1
        a = np.arange(-forma, forma+1)
        b = np.arange(-forma, forma+1)
        X, Y = np.meshgrid(a, b)
        f = X**2 + Y**2 <= forma**2
        f = 1*f

    if tipo == 'square':
        f = np.ones((forma, forma), dtype=int)

    if tipo == 'lineV':
        a = np.zeros((forma, forma), dtype=int)
        [fils, cols] = a.shape
        cont = 0

        for cols in range(1, forma):
            for fils in range(0, forma):
                X = round(forma/2)-1
                if cont == X:
                    a[fils][cols] = 1
                if cont != X:
                    a[fils][cols] = 0
                cont = cont+1
        f=a

    if tipo == 'lineH':
        a = np.zeros((forma, forma), dtype=int)
        [fils, cols] = a.shape
        cont = 0

        for cols in range(1, forma):
            for fils in range(1, forma):
                X = round(forma/2)-1
                if cont == X:
                    a[cols] = 1
                cont = cont+1
        f=a

    if tipo == 'cross':
        a = np.zeros((forma, forma), dtype=int)
        [fils, cols] = a.shape
        cont=0

        for fils in range(1, forma):
            for cols in range(1, forma):
                X = round(forma/2)-1
                if cont == X:
                    a[fils] = 1
                cont=cont+1
        [fils, cols] = a.shape
        cont=0

        for cols in range(1, forma):
            for fils in range(0, forma):
                X = round(forma/2)-1
                if cont == X:
                    a[fils][cols] = 1
                cont=cont+1
        f=a

    return f

```

Fig. 22.3. Algoritmo en Python.

23. Función graythresh

23.1. Uso

graythresh Evalúa el umbral de una imagen mediante el método de Otsu.

La sintaxis para su uso es:

- `graythresh (I)`

Retorna T:

Para convertir una imagen de intensidad “I” a una imagen binaria, se hace uso de la función *im2bw* y este evalúa el umbral global “T”.

“T” es un valor de intensidad normalizado que se encuentra en el intervalo de [0, 1].

La función *graythresh* hace uso del método de Otsu que elige el umbral para minimizar la varianza intraclass de los píxeles en blanco y negro.

23.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *graythresh*:

1. Se lee una imagen llamada “manzana.jpg” y se almacena en una variable “A”:
 - `A = imread ('manzana.jpg')`
2. Se guarda la componente azul de la imagen leída y se almacena en una variable “B”.
 - `B = A(:,:,2)`
3. Se usa la instrucción *graythresh* para evaluar el umbral y se almacena en una variable “T”.
 - `T = graythresh(B)`

23.3. Resultado

ans = 0.4824

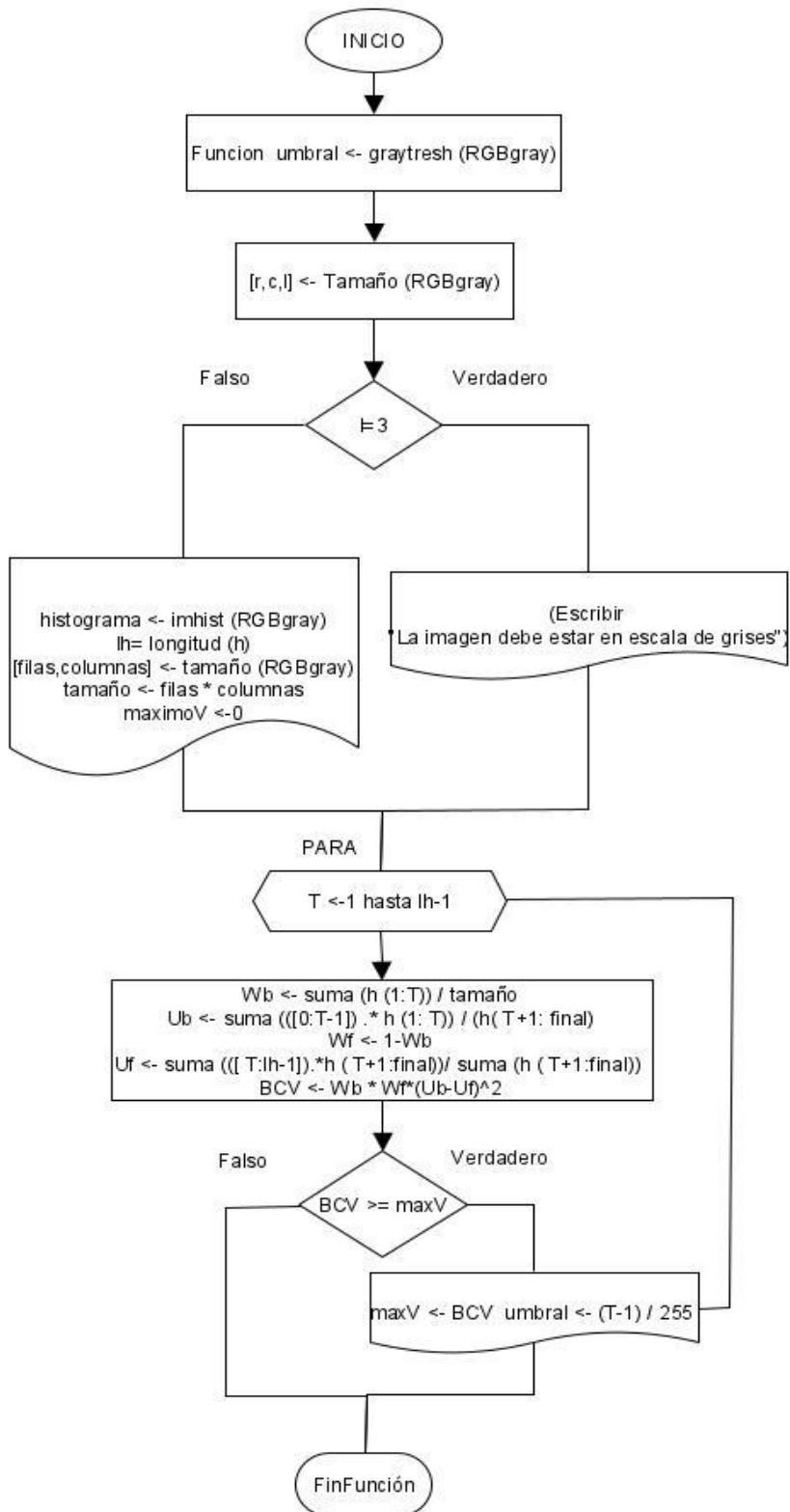


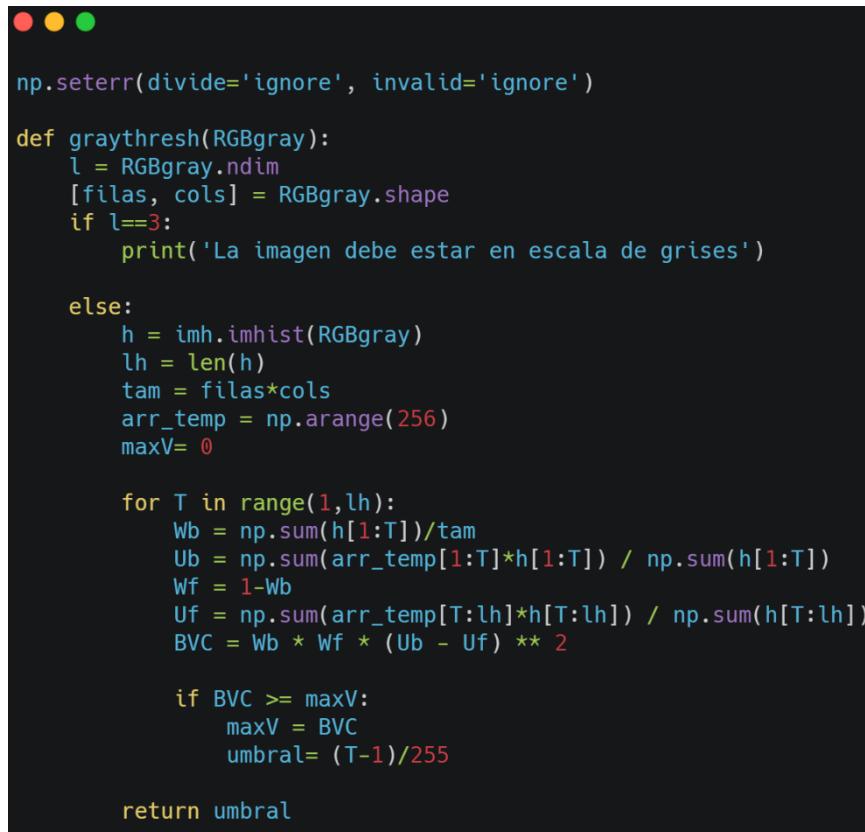
Fig. 23.1. Diagrama de flujo de la función

```

1 //Nombre:graytresh
2 // Parámetros: RGBgray: Es la imagen en escala de grises
3 //             umbral: Es el umbral calculado.
4
5 Algoritmo graytresh
6
7 Funcion umbral <- graytresh (RGBgray)
8     [r,c,l] <- Tamaño (RGBgray); // tamaño de la imagen:retorna su tamaño en filas,columnas y número de capas.
9     si l=3
10         (Escribir "La imagen debe estar en escala de grises")
11         SiNo
12             histograma <- imhist (RGBgray)
13             lh= longitud (h)                                //longitud del histograma
14             [filas,columnas] <- tamaño (RGBgray)      //Tamaño de la imagen
15             tamaño <- filas * columnas
16             maximoV <-0
17
18
19             para T <-1 hasta lh-1
20                 Wb <- suma (h (1:T)) / tamaño
21                 Ub <- suma ([0:T-1]) .* h (1: T) / (h( T+1: final)
22                 Wf <- 1-Wb
23                 Uf <- suma (([ T:lh-1]).*h ( T+1:final))/ suma (h ( T+1:final))
24                 BCV <- Wb * Wf* (Ub-Uf)^2
25
26                 Si BCV >= maxV
27                     maxV <- BCV
28                     umbral <- (T-1) / 255
29                 FinSi
30             FinPara
31         FinSi
32     FinFuncion
33     FinAlgoritmo

```

Fig. 23.2. Función en pseudolenguaje pse



```

np.seterr(divide='ignore', invalid='ignore')

def graytresh(RGBgray):
    l = RGBgray.ndim
    [filas, cols] = RGBgray.shape
    if l==3:
        print('La imagen debe estar en escala de grises')

    else:
        h = imh.imhist(RGBgray)
        lh = len(h)
        tam = filas*cols
        arr_temp = np.arange(256)
        maxV= 0

        for T in range(1,lh):
            Wb = np.sum(h[1:T])/tam
            Ub = np.sum(arr_temp[1:T]*h[1:T]) / np.sum(h[1:T])
            Wf = 1-Wb
            Uf = np.sum(arr_temp[T:lh]*h[T:lh]) / np.sum(h[T:lh])
            BVC = Wb * Wf * (Ub - Uf) ** 2

            if BVC >= maxV:
                maxV = BVC
                umbral= (T-1)/255

    return umbral

```

Fig. 23.3 Algoritmo en Python

24. Función im2bw

24.1. Uso

im2bw Cambia una imagen a una imagen binaria, basado en el umbral “T”.

La sintaxis para su uso es:

- `im2bw(b,T)`

Donde: “b” es la imagen y “T” es el umbral.

A partir de una imagen “l” en escala de grises crea una imagen binarizada “B”.

Retorna “bin”: La imagen binarizada

24.2. Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *im2bw*:

1. Se lee una imagen llamada “manzana.jpg” y se almacena en una variable “A”:

- `A = imread ('manzana.jpg')`

2. Se guarda la componente azul de la imagen leída y se almacena en una variable “B”

- `B = A(:,:,3)`

3. Se usa la instrucción *graythresh* para evaluar el umbral y se almacena en una variable “T”.

- `T = graythresh (B)`

4. Se usa la instrucción *im2bw* para binarizar la imagen y se almacena en una variable “binaria”.

`binaria = im2bw (b,T)`

5. Se muestra la imagen binarizada con la instrucción *imshow*:

- `imshow (binaria)`

24.3. Resultado

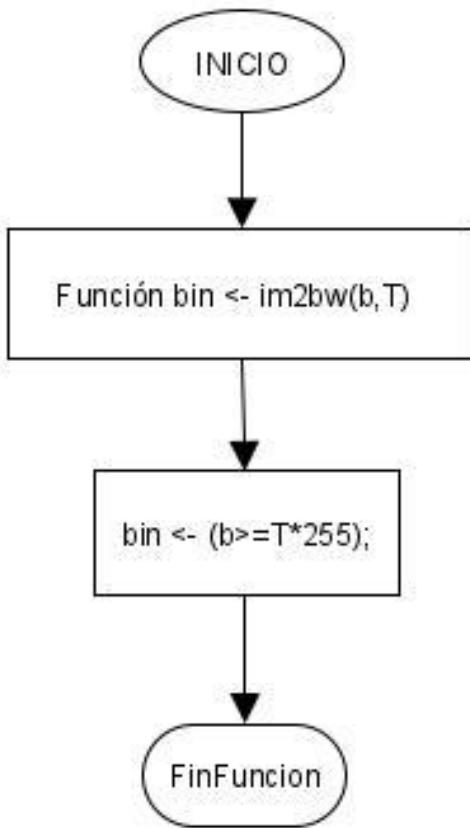


Fig. 24.1. Diagrama de flujo de la función

```
1 //Nombre: im2bw
2 // Parámetros: T es el umbral determinado entre 0 y 1
3 //           bin es la imagen binarizada.
4 //Devuelve conversión de una imagen 2D a Binaria usando el umbral T
5
6 Algoritmo im2bw
7
8     funcion bin <- im2bw(b,T)
9         bin <- (b>=T*255);
10        finFuncion
11 FinAlgoritmo
```

Fig. 24.2 Función en pseudolenguaje pse

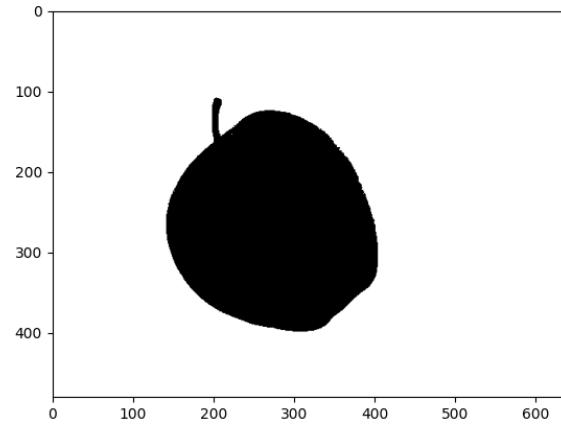


```
def im2bw(b,T):
    bin = (b >= T*255)
    return bin
```

Fig. 24.3 Algoritmo en Python



IA (Imagen en RGB)



B (Imagen binarizada)

Fig. 24.4 Imagen binarizada⁸

⁸ Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

25. Función histeq

25.1. Uso

histeq Ecualiza el histograma para mejorar el contraste de una imagen.

La sintaxis para su uso es:

- `histeq(img)`

Donde “img” es la imagen en escala de grises.

Retorna S: La imagen ecualizada.

25.2. Ejemplo

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *histeq*.

1. Se lee una imagen llamada “cartagena.jpg” y se almacena en la variable “A”.
 - `A = imread ('cartagena.jpg')`
2. Se cambia la imagen a escala de grises y se almacena en la variable “gris”.
 - `gris = rgb2gray (A)`
3. Se muestra el histograma de la imagen “gris” con la instrucción *imhist*.
 - `imhist (gris)`
4. Se realiza la ecualización de la imagen en escala de grises.
 - `S = histeq (gris)`
 - `imhist (S)`

25.3. Resultado:

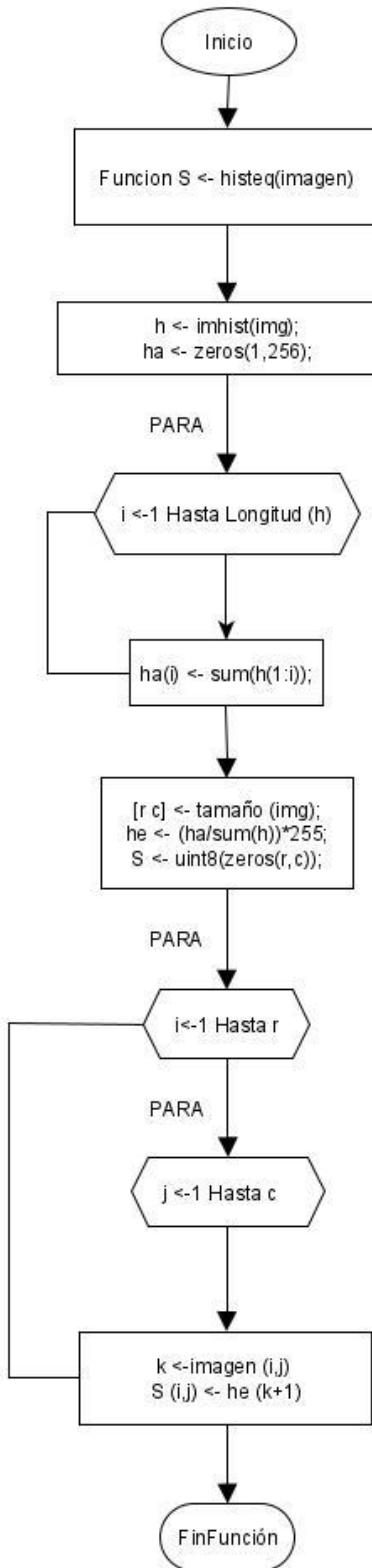


Fig. 25.1. Diagrama de flujo de la función.

```

1 //Nombre: histeq
2 // Parámetros:Imagen: Es la imagen en escala de grises
3 //           S: Es la imagen ecualizada.
4
5 Algoritmo histeq
6
7 Funcion S <- histeq(imagen)
8     h <- imhist(img);
9     ha <- zeros(1,256);
10
11    Para i <- 1 hasta longitud (h)
12        ha(i) <- sum(h(1:i));
13    FinPara
14
15    [r c] <- tamaño (img);
16    he <- (ha/sum(h))*255;
17    S <- uint8(zeros(r,c));
18
19    Para i<-1 Hasta r
20        Para j <-1 Hasta c
21            k <-Imagen (i,j)
22            S (i,j) <- he (k+1)
23        FinPara
24    FinPara
25 finFuncion
26
27 FinAlgoritmo

```

Fig. 25.2. Función en pseudolenguaje pse.



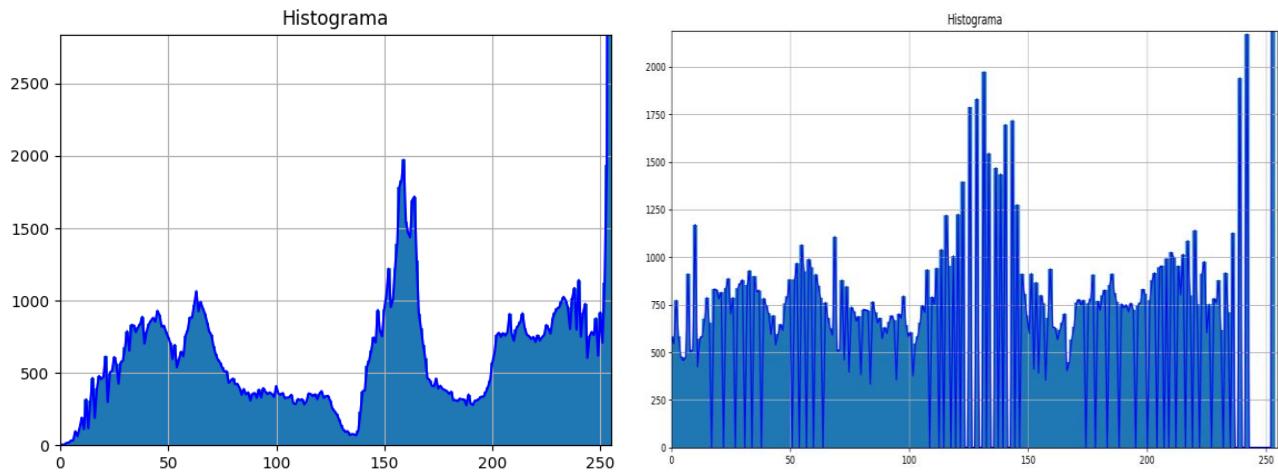
```

def histeq(img):
    h= imhist(img)
    ha = np.array([sum(h[:i+1]) for i in range(len(h))])
    he = np.uint8(255 * ha)
    r,c= img.shape[:2]
    S= np.zeros_like(img)

    for i in range(0, r):
        for j in range(0, c):
            S[i, j] = he[int(img[i, j])]
    return S

```

Fig. 25.3. Algoritmo en Python.



(a)Histograma de la imagen en escala de grises.

(b)Histograma de la imagen ecualizada en escala de grises.

Fig. 25.4 Ecualización en Python.

26. Función mat2gray

26.1 Uso

`mat2gray` Cambia una matriz en escala de grises.

La sintaxis para su uso es:

- `mat2gray(I, **lim)`

Donde: “I” es la matriz numérica y “lim” son valores límite.

26.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `mat2gray`.

1. Se crea una matriz numérica y se almacena en una variable “A”.
 - $A = [4,7,8;7,9,0;5,6,7]$
1. Se cambia la imagen a escala de grises y se almacena en una variable “gris”.

- gris=mat2gray(A)
2. Se muestra la imagen con la instrucción “imshow”.
- imshow(gris)

26.3 Resultado

```
[ [0.44444444 0.77777778 0.88888889]
  [0.77777778 1.          0.          ]
  [0.55555556 0.66666667 0.77777778] ]
```

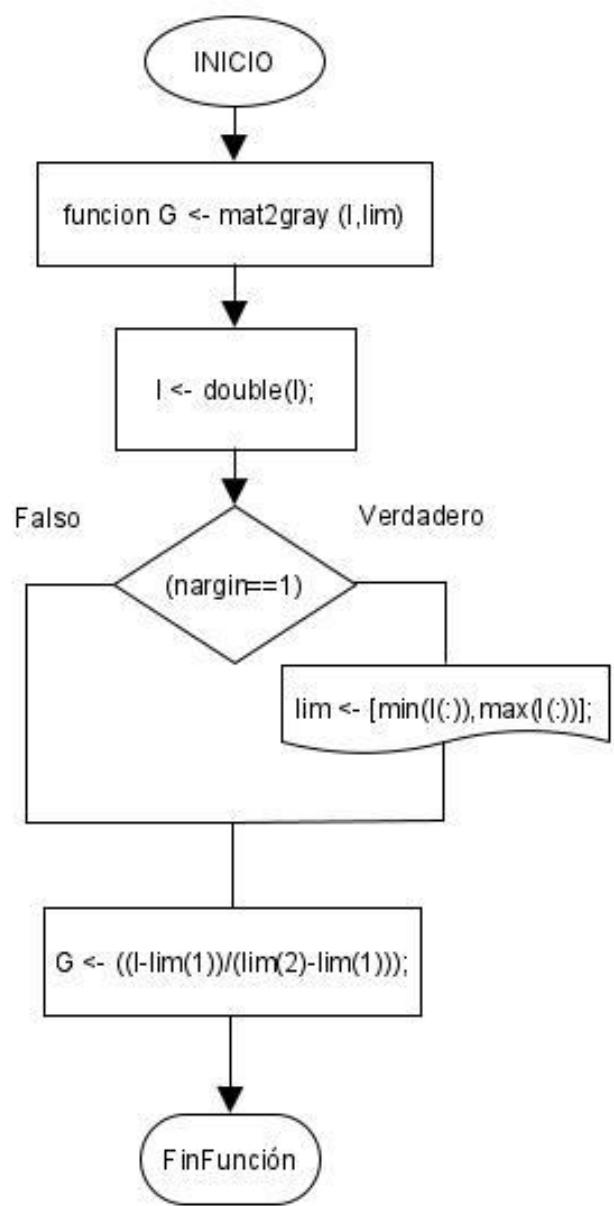


Fig. 26.1. Diagrama de flujo de la función.

```

1 //Nombre: mat2gray
2 //Parámetros: Donde I es la matriz numérica
3 //              G es la imagen en formato RGB.
4 //Devuelve los valores de 'y' en los valores límite.
5
6 Algoritmo mat2gray
7
8     función G <- mat2gray (I,lim)
9         I <- double(I);
10
11         Si (nargin==1)
12             lim <- [min(I(:)),max(I(:))];
13         FinSi
14
15         G <- ((I-lim(1))/(lim(2)-lim(1)));
16
17     FinFunción
18
19 FinAlgoritmo
20
21 //NOTA:
22 //nargin: Determina el número de argumentos de entrada.
23 // double: Conversión a un número real.

```

Fig. 26.2. Función en pseudolenguaje pse.



```

def mat2gray(I, **lim):
    if len(lim) == 0:
        lim = [min(I.flatten(order = 'C')), max(I.flatten(order = 'C'))]
    else:
        lim = lim['lim']
    G = ((I-lim[0])/(lim[1]-lim[0]))
    return G

```

Fig. 26.3 Algoritmo en Python.

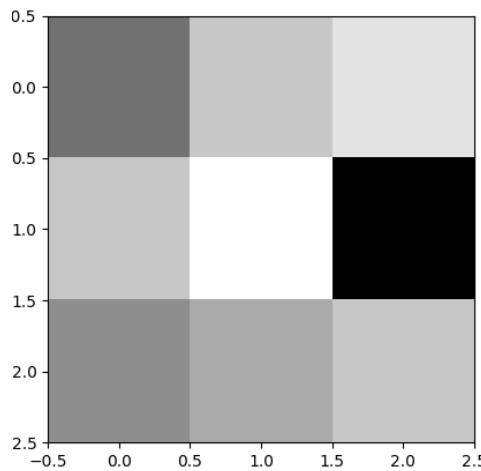


Fig. 26.4. Imagen creada mat2gray en Python.

27. Función imwrite.

27.1 Uso

Imwrite Escribe los datos de una matriz en un archivo determinado.

La sintaxis para su uso es:

- `imwrite(A, filename, **kwargs)`

Donde: “A” es la imagen.

27.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imwrite*:

1. Se crea una imagen en escala de grises, se almacena en una variable “a”, se muestra con la instrucción “imshow” y se almacena en un archivo de imagen.

- `a = np.uint8(255*np.random.rand(64, 64))`

- `imshow(a)`
- `imwrite(a, 'imwrite.jpg', cmapa = 'gray')`

2. Se lee el archivo de imagen y se muestra con la instrucción “imshow”.

- `b = plt.imread('imwrite.jpg')`
- `plt.imshow(b)`
- `plt.show()"`

27.3 Resultado:

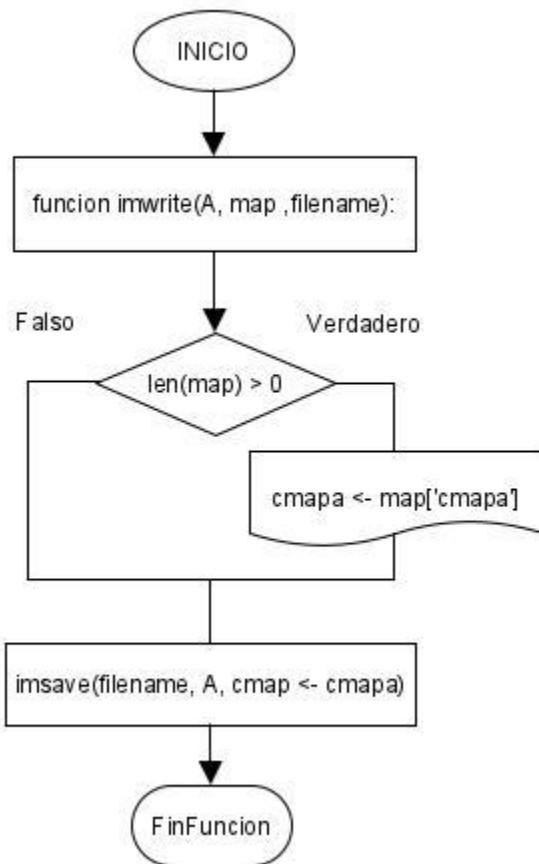


Fig. 27.1. Diagrama de flujo de la función.

```

1 //Nombre: imwrite
2 //Parámetros: Donde filename nombre del archivo a escribir.
3 //           "A" es la imagen que contiene los datos.
4 //           "map" contiene mapa de colores.
5
6 Algoritmo imwrite
7
8 función imwrite(A, map ,filename):
9     Si len(map) > 0
10        cmapa <- map['cmapa']
11        imsave(filename, A, cmap = cmapa)
12
13 FinFuncion
14
15 FinAlgoritmo

```

Fig. 27.2. Función en pseudolenguaje pse.

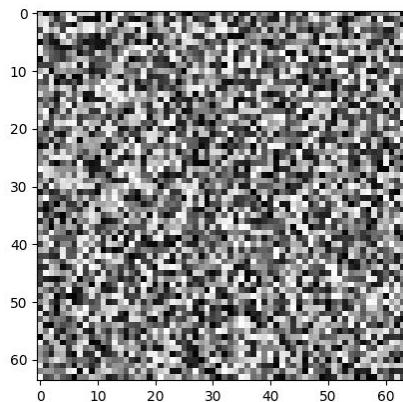


```

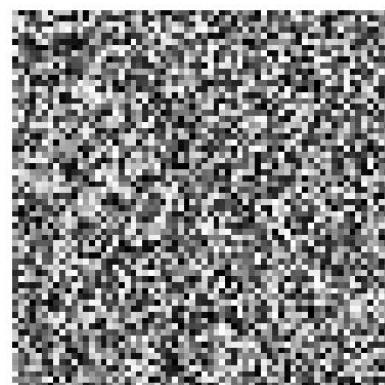
def imwrite(A, filename, **kwargs):
    if len(kwargs) > 0:
        cmapa = kwargs['cmapa']
    plt.imsave(filename, A, cmap = cmapa)

```

Fig. 27.3. Algoritmo en Python.



(a) Imagen guardada.



(b)Imagen leída.

Fig. 27.4. Imágenes creadas imwrite Python.

28. Función imfilter.

28.1 Uso:

imfilter Realiza el filtrado de correlación 2D de la imagen.

La sintaxis para su uso es:

- `imfilter(Gris,K)`

Donde: "Gris" es la imagen en escala de grises y "K" es el tipo de correlación.

El parámetro "K" tiene las siguientes entradas tipo string:

- 'full' - Realiza la correlación completa.
- 'same' - Retorna la parte central del resultado que es del mismo tamaño de I (defecto).
- 'valid' - Retorna una porción de la correlación computada sin zero-padd.

28.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imfilter*:

1. Se lee una imagen llamada "cartagena.jpg", se almacena en una variable "I".

- `I = imread('cartagena.JPG')`

2. Se cambia a escala de grises y se almacena en una variable "Ic".

- `Ic = rgb2gray(I)`

3. Se añade ruido gaussiano a la imagen "Ic", se almacena en una variable "Ir" y se muestra con la instrucción *imshow*.

- `Ir = imnoise(Ic,'gaussian',0.1)`
- `imshow(Ir)`

4. Se crea un kernel de filtro promedio en K2.

- `K2 = fspecial('average')`

5. Se realiza la correlación entre la imagen “Ic” y “K2” y se muestra con la instrucción “imshow”.

- `imf2 = my_imfilter(Ir, K2)`
- `imshow(imf2)''`

28.3 Resultado:

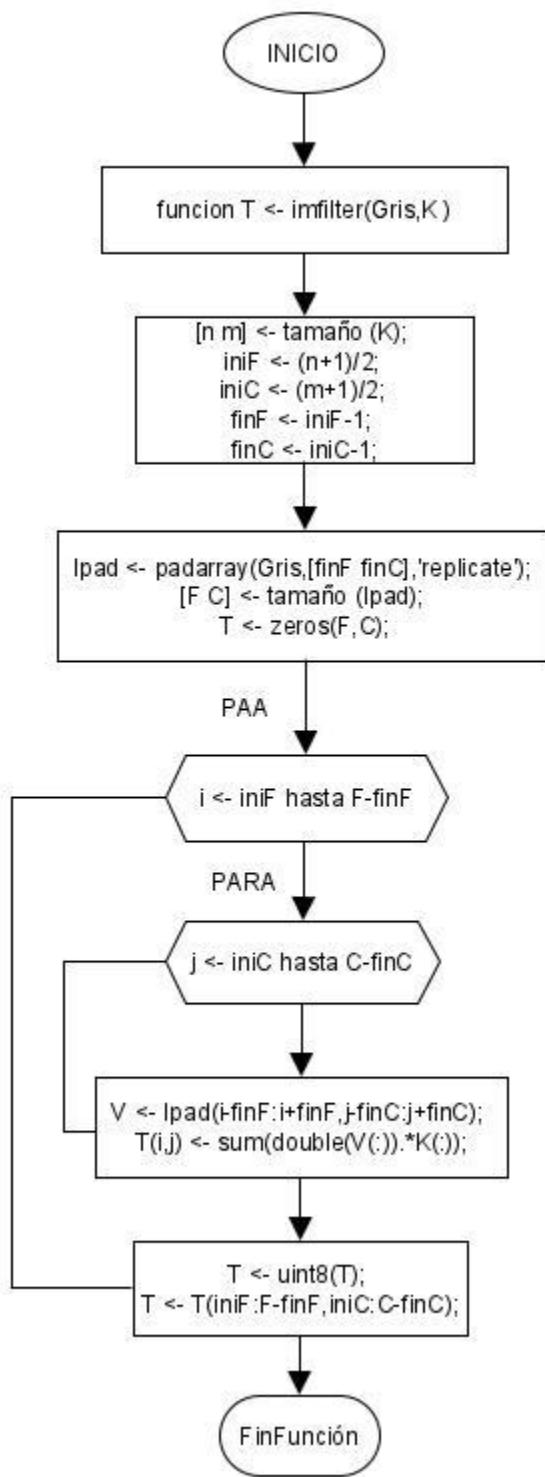


Fig. 28.1. Diagrama de flujo de la función.

```

1 //Nombre: imfilter
2 //Parámetros: Gris imagen escala de grises
3 //          K es la imagen a filtrar
4 //Devuelve: Filtrado N-D de imágenes multidimensionales
5
6 Algoritmo imfilter
7 funcion T <- imfilter(Gris,K )
8     [n m] <- tamaño (K);
9     iniF <- (n+1)/2;
10    iniC <- (m+1)/2;
11    finF <- iniF-1;
12    finC <- iniC-1;
13
14    Ipad <- padarray(Gris,[finF finC],'replicate');
15    [F C] <- tamaño (Ipad);
16    T <- zeros(F,C);
17
18    Para i <- iniF hasta F-finF
19        para j <- iniC hasta C-finC
20            V <- Ipad(i-finF:i+finF,j-finC:j+finC);
21            T(i,j) <- sum(double(V(:)).*K(:));
22        FinPara
23    FinPara
24
25    T <- uint8(T);
26    T <- T(iniF:F-finF,iniC:C-finC);
27 FinAlgoritmo
28
29 //NOTA:
30 //double: Conversión a un número real.

```

Fig. 28.2. Función en pseudolenguaje pse.

```

● ● ●

def imfilter(Gris,K):
    [n , m] = K.shape
    iniF = int((n+1)/2)
    iniC = int((m+1)/2)
    finF = int(iniF-1)
    finC = int(iniC-1)

    Ipad = np.pad(Gris.astype(int), (finF, finC), 'edge')
    Ipad = Ipad.astype(int)
    [F, C] = Ipad.shape
    T = np.zeros((F,C), dtype=int)

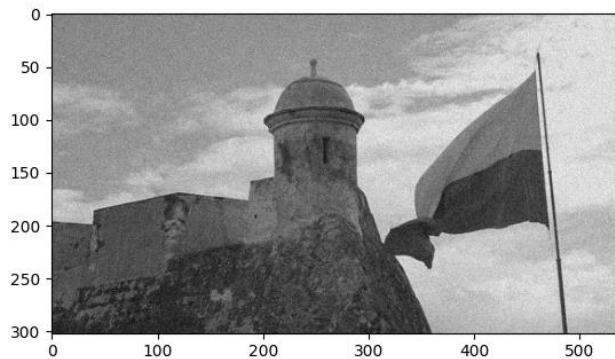
    for i in range(iniF, F-finF):
        for j in range(iniC, C-finC):
            V = Ipad[i-finF:i+finF+1,j-finC:j+finC+1]
            T[i,j] = np.sum(V.flatten(order = 'C')*K.flatten(order = 'C'))

    T = T[iniF:F-finF,iniC:C-finC]

    return T

```

Fig. 28.3. Algoritmo en Python.



(a)Imagen con ruido Gaussiano.



(b))Correlación de la imagen y el kernel.

Fig. 28.4. Filtrado de correlación en Python.

29. Función Hough.

29.1 Uso

`houhg` Evalúa la transformada de Houhg en una imagen binaria.

La sintaxis para su uso es:

- `hough(I)`

Donde: “I” es la imagen binarizada.

Los parámetros opcionales son:

'Rhomap': Modifica la resolución de Rho, valor puede ser de cualquier tipo numérico.

- `[P,Theta,Rho]=hough(Ibin,'RhoResolution',valor);`

'Thetamap': Modifica la resolución de Theta, valor puede ser de cualquier tipo numérico.

- `[P,Theta,Rho]=hough(Ibin,'RhoResolution',valor)`

'Th': Determina el rango de valores de theta en un vector de 2 posiciones [Tmin,Tmax].

- `[P,Th,Rho]=hough(Ibin,'Th',[Tmin,Tmax])`

29.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *Hough*:

1. Se lee una imagen llamada "rectas.jpg" y se almacena en la variable "A".
 - `A = imread('rectas.jpg')`
2. Se cambia la imagen en escala de grises y se almacena en una variable "Igris".
 - `Igris = rgb2gray(A)`
3. Se binariza la imagen, se almacena en la variable "Ibin" y se muestra usando la instrucción "imshow".
 - `Ibin = im2bw(Igris, 0.8)`
 - `Ibin = ~Ibin*1`
 - `imshow(Ibin)`
4. Se evalúa la transformada de Hough de la imagen "Ibin" y se transforma para ser muestrada usando la instrucción *imshow*.
 - `Th, Theta, Rho = hough(Ibin)`
 - `Hm1 = mat.mat2gray(Th)`
 - `imshow(Hm1)`
 - `Hm2 = imadjust(Hm1, [0,0.0824], [0, 1], 1)`
 - `imshow(Hm2)`

29.3 Resultado:

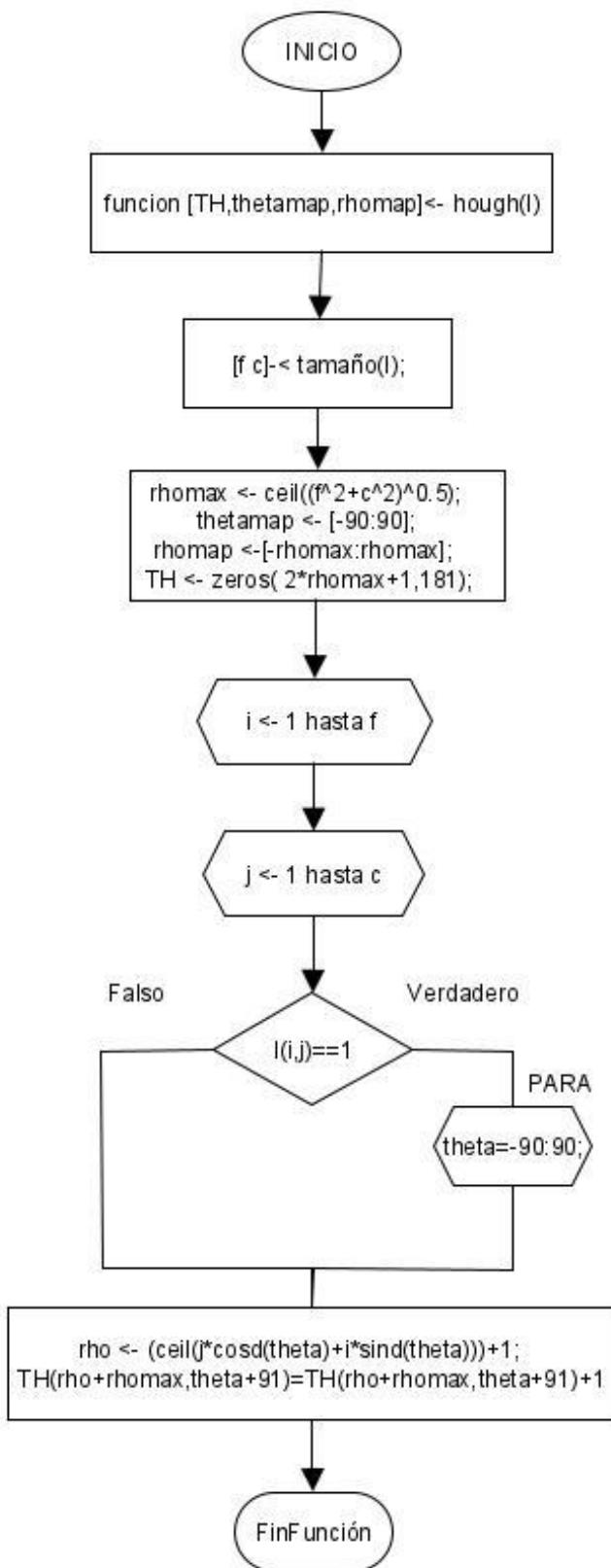


Fig. 29.1. Diagrama de flujo de la función.

```

1 //Nombre: hough.
2 //Parámetros:Donde I es la imagen binarizada
3 //           TH es la matriz acumuladora de Hough.
4 //           Theta es el vector de ángulo en grados
5 //           Rho es el vector de rho.
6
7 Algoritmo hough
8 funcion [TH,thetemap,rhomap]<- hough(I)
9     [f c]<- tamaño(I);
10    rhomax <- ceil((f^2+c^2)^0.5);
11    thetemap <- [-90:90];
12    rhomap <-[rhomax:rhomax];
13    TH <- zeros( 2*rhomax+1,181);
14    for i <- 1 hasta f
15        for j <- 1 hasta c
16            if I(i,j)==1
17                for theta=-90:90;
18                    rho <- (ceil(j*cosd(theta))+i*sind(theta)))+1;
19                    TH(rho+rhomax,theta+91)=TH(rho+rhomax,theta+91)+1;
20                end
21            end
22        end
23    end
24
25 FinAlgoritmo
26
27 //NOTA:
28 //RhoResolution: Modifica la resolución de Rho, valor puede ser de cualquier tipo numérico.
29 //ThetaResolution: Modifica la resolución de Theta, valor puede ser de cualquier tipo numérico.
30 //Theta: Determina el rango de valores de theta en un vector de 2 posiciones [Tmin,Tmax].
31 //Varargin: Lista de argumentos de entrada de longitud variable.

```

Fig. 29.2. Función en pseudolenguaje pse.



```

def hough(I):
    f,c = I.shape
    rhomax = int(np.ceil(pow(pow(f,2)+pow(c, 2), 0.5)))
    thetemap = np.arange(-90, 91).reshape(1, 181)
    rhomap = np.arange(-rhomax, rhomax+1, dtype=int).reshape(1, 667)
    Th = np.zeros((int(2*rhomax+1), 181))
    for i in range(0, f):
        for j in range(0, c):
            if I[i, j] == 1:
                for theta in range(-90, 91):
                    rho = int(np.ceil((j*np.cos(np.radians(theta)) + (i*np.sin(np.radians(theta)))))+1
                    Th[rho+rhomax, theta+90] = Th[rho+rhomax, theta+90] + 1
    return [Th, thetemap, rhomap]

```

Fig. 29.3. Algoritmo en Python.

30. Función imerode.

30.1 Uso

Imerode Erosiona una imagen.

La sintaxis para su uso es:

- `imerode(Ibin,SE)`

Donde: “Ibin” es la imagen binaria y “SE” elemento estructurador que corresponde a una matriz.

30.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imerode*:

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”.
 - `A=imread('herramientas.jpg');`
2. Se cambia la imagen a escala de grises y se almacena en una variable “B”.
 - `B=rgb2gray(A)`
3. Se usa la instrucción *graythresh* para encontrar el umbral T de la imagen y se almacena en una variable “T”.
 - `T = graythresh(B)`
4. Se usa la instrucción *im2bw* para binarizar la imagen “B” y se almacena en una variable “C”.
 - `C = im2bw(B,T)`
5. Se usa la instrucción *strel* para crear un elemento estructurante de tipo ‘disk’ de tamaño 3x3 y se almacena en una variable “SE”.

- $\text{SE} = \text{strel('disk',3)}$

6. Se usa la instrucción *imerode* para erosionar la imagen “C” y almacena en una variable “D”.

- $\text{D} = \text{imerode}(\text{C}, \text{SE})$

7. Se muestran las imágenes binarizadas y erosionadas con la instrucción *imshow*.

- $\text{imshow}(\text{C})$
- $\text{imshow}(\text{D})$

30.3 Resultado:

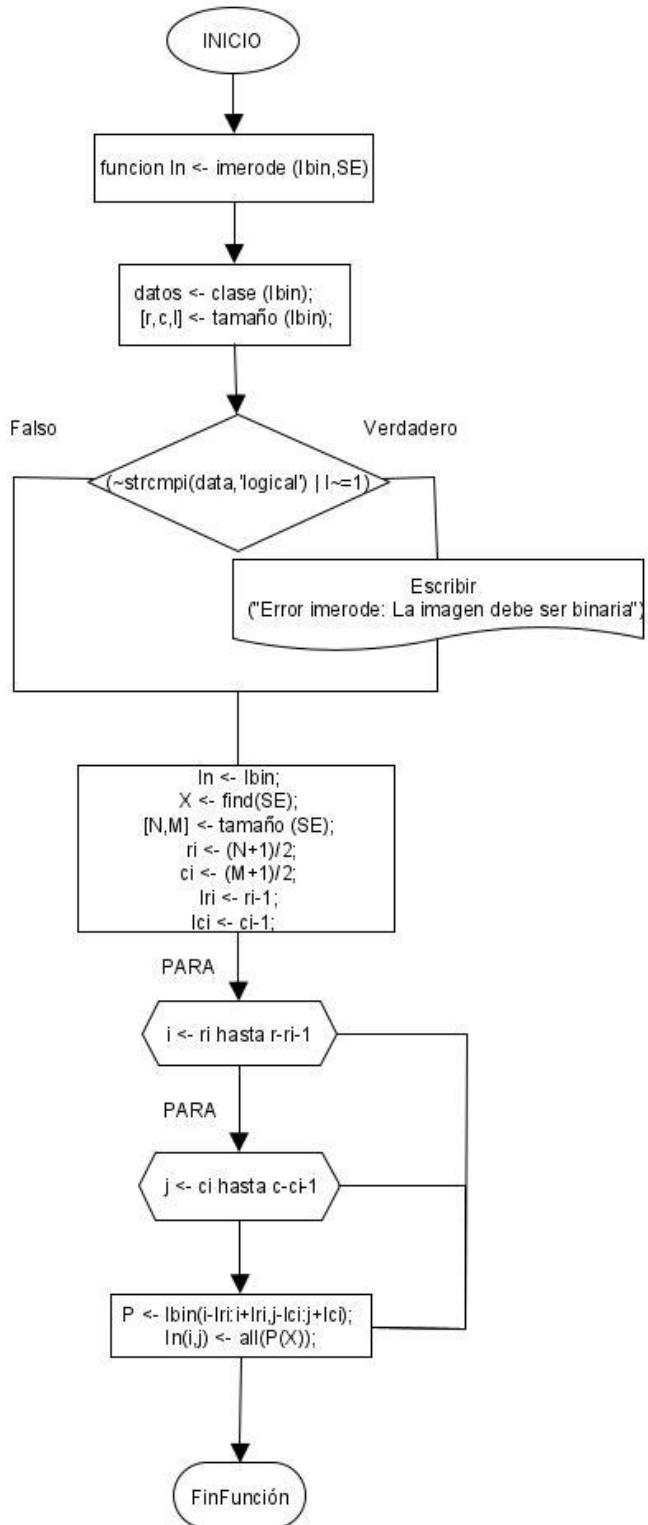


Fig. 30.1. Diagrama de flujo de la función.

```

1 //Nombre: imerode
2 //Parámetros: Donde Ibin es la imagen binaria,
3 //           SE es el elemento escructurante que corresponde a una matriz.
4 //           In es la imagen binaria con el cerrado morfologico.
5 //Devuelve la imagen erosionada.
6
7 Algoritmo imerode
8     función In <- imerode (Ibin,SE)
9         datos <- clase (Ibin);
10        [r,c,l] <- tamaño (Ibin);
11
12        Si (~strcmpi(data,'logical') | l~=1)
13            Escribir ("Error imerode: La imagen debe ser binaria")
14        FinSi
15
16        In <- Ibin;
17        X <- find(SE);
18        [N,M] <- tamaño (SE);
19        ri <- (N+1)/2;
20        ci <- (M+1)/2;
21        lri <- ri-1;
22        lci <- ci-1;
23
24        Para i <- ri hasta r-ri-1
25            Para j <- ci hasta c-ci-1
26                P <- Ibin(i-lri:i+lri,j-lci:j+lci);
27                In(i,j) <- all(P(X));
28            FinPara
29        FinPara
30    FinFunción
31 FinAlgoritmo
32
33 //NOTA:
34 //Find: Buscar indices y valores de elementos no nulos.

```

Fig. 30.2. Función en pseudolenguaje pse.

```


def imerode(Ibin,SE):
    Ibin = 1*Ibin
    filt = np.ones((7, 7))
    r, c = Ibin.shape
    f, co = SE.shape
    R = r+f-1
    C = c+co-1
    In = np.zeros((R, C))

    for i in range(r):
        for j in range(c):
            In[i+1, j+1] = Ibin[i, j]

    for i in range(r):
        for j in range(c):
            k = In[i:i+f,j:j+co]
            resultado = (k == filt)
            final = np.all(resultado == True)
            if final:
                Ibin[i,j] = 1
            else:
                Ibin[i,j] = 0
    return Ibin

```

Fig. 30.3. Algoritmo en Python.

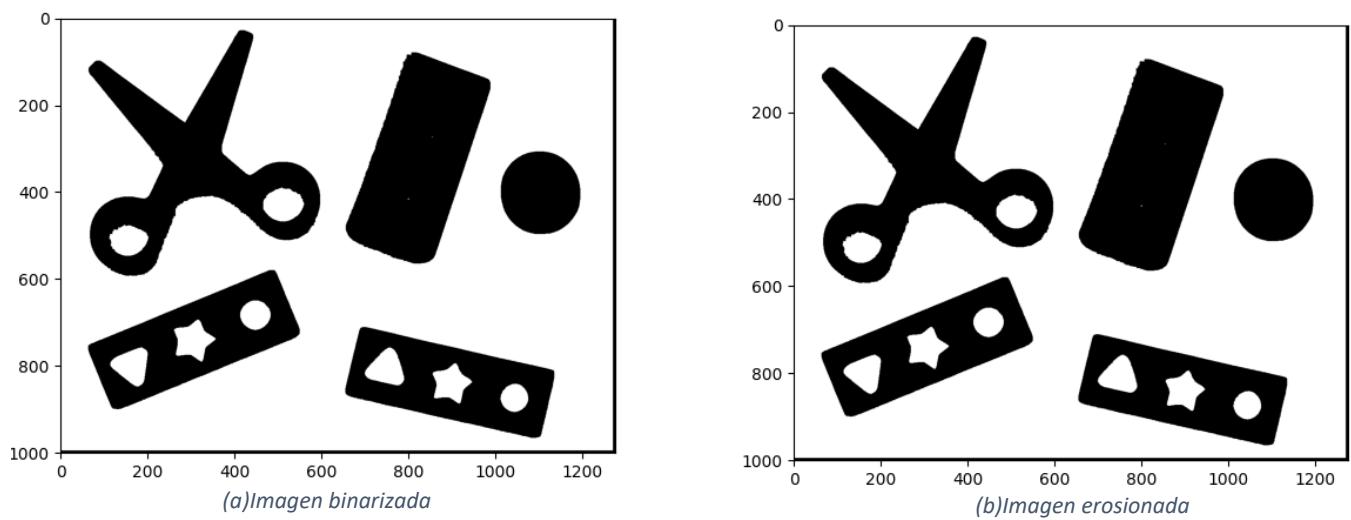


Fig. 30.4. Imágenes creadas imerode en Python.

31. Función imdilate.

31.1 Uso.

Uso: Dilata una imagen binaria o en escala de grises.

La sintaxis para su uso es:

- `imdilate (Ibin,SE)`

Donde: "Ibin" es la imagen binaria o en escala de grises y "SE" elemento estructurador que corresponde a una matriz.

31.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imdilate*:

1. Se lee una imagen llamada "herramientas.jpg" y se almacena en una variable "A".
 - `A=imread('herramientas.jpg');`
2. Se cambia la imagen a escala de grises y se almacena en una variable "B".
 - `B=rgb2gray(A)`
3. Se usa la instrucción *graythresh* para encontrar el umbral T de la imagen y se almacena en una variable "T".
 - `T = graythresh(B)`
4. Se usa la instrucción *im2bw* para binarizar la imagen "B" y se almacena en una variable "C".
 - `C = im2bw(B,T)`
5. Se usa la instrucción *strel* para crear un elemento estructurador de tipo 'disk' de tamaño 3x3 y se almacena en una variable "SE".

- `SE = strel('disk',3)`

6. Se usa la instrucción *imdilate* para dilatar la imagen “C” y almacena en una variable “D”.

- `D = imdilate(C, SE)`

7. Se muestran la imagen dilatada con la instrucción *imshow*.

- `imshow(D)`

31.3 Resultado:

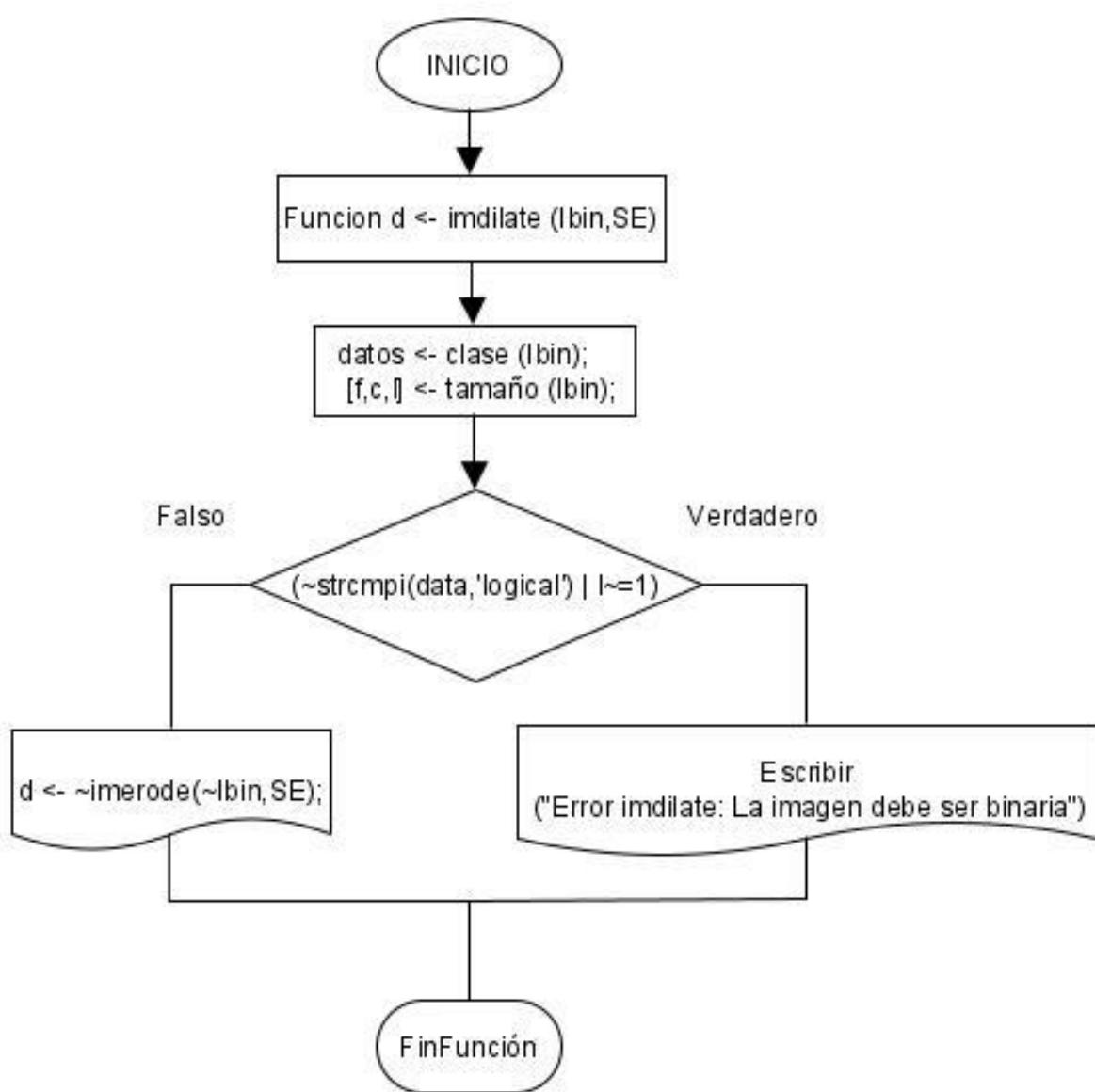


Fig. 31.1. Diagrama de flujo de la función.

```

1 //Nombre: imdilate
2 //Parámetros: Donde Ibin es la imagen binaria.
3 //           SE es el elemento estructurante que corresponde a una matriz.
4 //           d es la imagen binaria con el cerrado morfológico.
5 // Devuelve la imagen dilatada,
6
7 Algoritmo imdilate
8 Funcion d <- imdilate (Ibin,SE)
9     datos <- clase (Ibin);
10    [f,c,l] <- tamaño (Ibin);
11
12    Si (~strcmpi(data,'logical') | l~=1)
13        Escribir ("Error imdilate: La imagen debe ser binaria")
14    SiNo
15        d <- ~imerode(~Ibin,SE);
16    FinSi
17 FinFuncion
18
19 FinAlgoritmo
20
21 //NOTA:
22
23 // imrode: Erosiona la imagen.

```

Fig. 31.2. Función en pseudolenguaje pse.



```

def imdilate(Ibin,SE):
    l = Ibin.ndim
    if l > 2:
        print("imdilate: La imagen debe ser binaria")
    else:
        d = ~ime.imerode(~Ibin,SE)
    return d

```

Fig. 31.3 Algoritmo en Python.

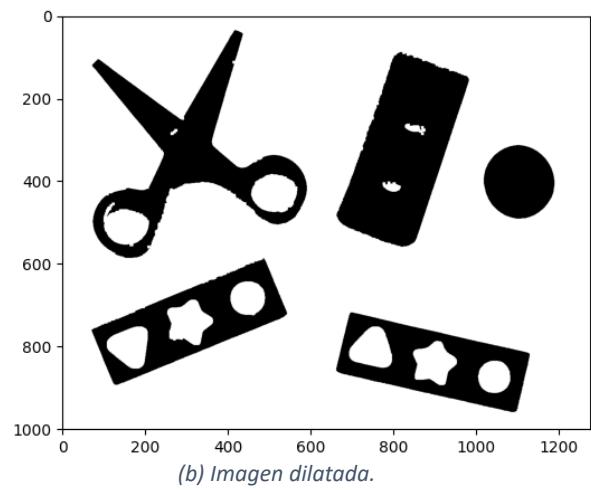
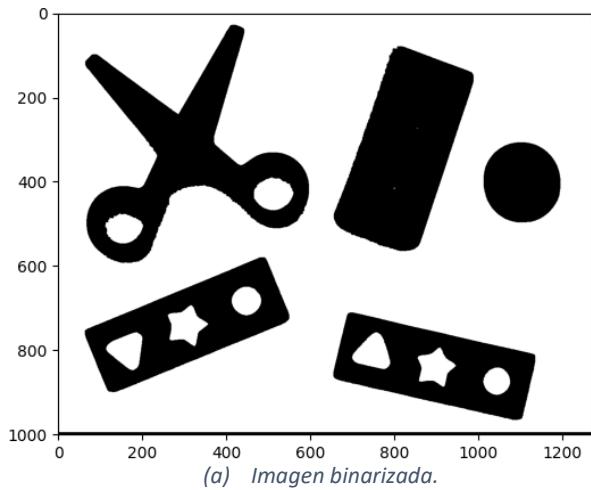


Fig.31.4. Imágenes generadas *imdilate* en Python.

32. Función imclose.

32.1 Uso

Imclose Realiza cerrado morfológico de una imagen.

La sintaxis para su uso es:

- `imclose(Ibin,SE)`

Donde: “Ibin” es la imagen y “SE” es elemento estructurador que corresponde a una matriz.

32.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *imclose*:

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”.

- A=imread('herramientas.jpg')
2. Se cambia la imagen a escala de grises y se almacena en una variable "B".
- B=rgb2gray(A);
3. Se usa la instrucción *graythresh* para encontrar el umbral T de la imagen y se almacena en una variable "T".
- T=graythresh(B)
4. Se usa la instrucción *im2bw* para binarizar la imagen y se almacena en una variable "C".
- C=im2bw(B,T);
5. Se usa la instrucción *strel* para crear un elemento estructurador de tipo 'disk' de tamaño 4X4 y se almacena en una variable "SE".
- SE=strel('disk',4)
6. Se usa la instrucción *imclose* para cerrar la imagen y se almacena en una variable "D".
- D=imclose(C,SE);
7. Se muestran las imágenes binarizadas y cerradas con la instrucción *imshow*.
- imshow(C)
 - imshow(D)

32.3 Resultado:

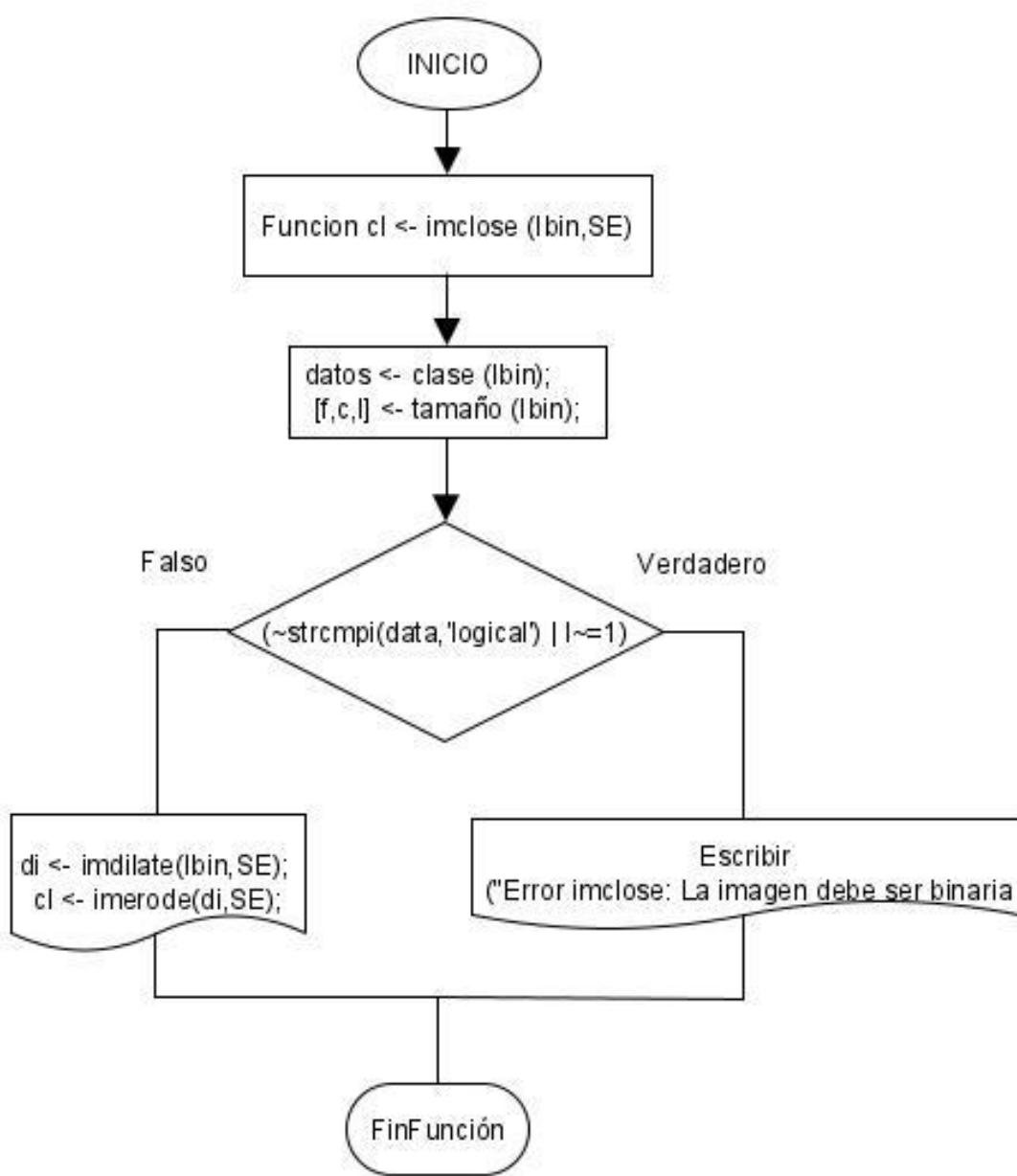


Fig. 32.1. Diagrama de flujo de la función.

```

1 //Nombre: imclose
2 //Parámetros: Donde Ibin es la imagen binaria,
3 //              SE es el elemento estructurante que corresponde a una matriz.
4 //              cl es la imagen binaria con el cerrado morfológico.
5 //Devuelve: La operación morfológica de cierre, primero dilatación seguida de una erosión.
6
7 Algoritmo imclose
8 Funcion cl <- imclose (Ibin,SE)
9     datos <- clase (Ibin);
10    [f,c,l] <- tamaño (Ibin);
11
12    Si (~strcmpi(data,'logical') | l~=1)
13        Escribir ("Error imclose: La imagen debe ser binaria")
14    SiNo
15        di <- imdilate(Ibin,SE);
16        cl <- imerode(di,SE);
17    FinSi
18 FinFuncion
19
20 FinAlgoritmo
21
22 //NOTA:
23
24 // imdilate: Dilatar imagen.
25 // imrode: Erosiona la imagen.

```

Fig. 32.2. Función en pseudolenguaje pse.



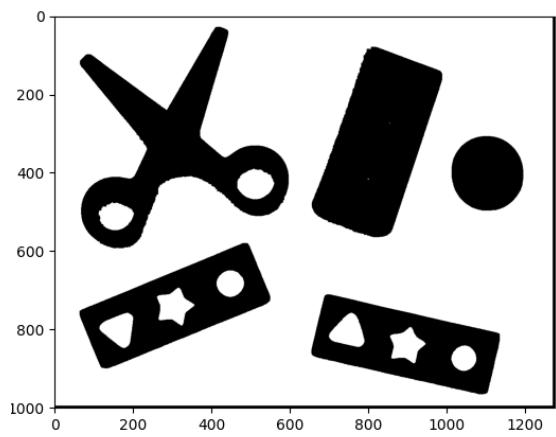
```

def imclose(Ibin,SE):
    l = Ibin.ndim
    di = imdilate(Ibin, SE)
    di = di*-1
    cl = imerode(di, SE)

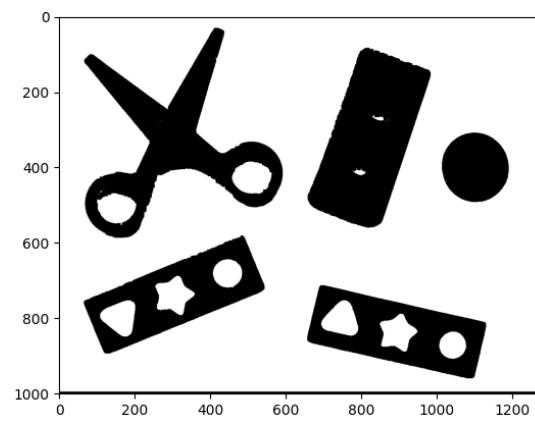
    return cl

```

Fig. 32.3. Algoritmo en Python.



a) Imagen binarizada



(b)Imagen con cerrado morfológico.

Fig. 32.4. Imágenes generadas imclose en Python.

33. Función fft2.

33.1 Uso

fft2 Evalúa la transformada rápida de Fourier (FFT) en 2D.

La sintaxis para su uso es:

- `fft2(I)`

Donde: "I" es la imagen.

Devuelve DFT (la transformada discreta de Fourier en 2D) de la imagen, la cual es evaluada con el algoritmo de la FFT.

33.2 Resultado:

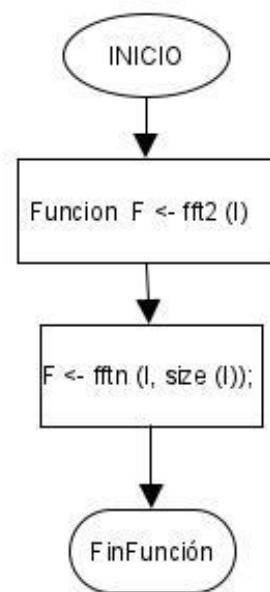


Fig. 33.1. Diagrama de flujo de la función.

```

1 //Nombre: fft2
2 //Parámetros: I Es la imagen
3 //Retorna: Transformada de Fourier 2D de la imagen I
4
5 Algoritmo fft2
6   funcion F <- fft2(I)
7     F <- ifftn(I,tamaño(I));
8   FinFuncion
9   Finalgoritmo

```

Fig. 33.2. Función en pseudolenguaje pse.

```

def fft2(I):
    x, y = I.shape
    F = np.fft.ifftn(I, (x,y))
    return F

```

Fig. 33.3. Algoritmo en Python.

34. Función fftshift

34.1 Uso

fftshift Desplaza la matriz o el componente de frecuencia cero en una dirección positiva o al centro del espectro.

La sintaxis para su uso es:

- `ffshift(fw)`

Donde: “fw” es una matriz.

Recompone la salida de `fft2` desplazando los componentes de frecuencia cero al centro del espectro de la matriz.

34.2 Resultado:

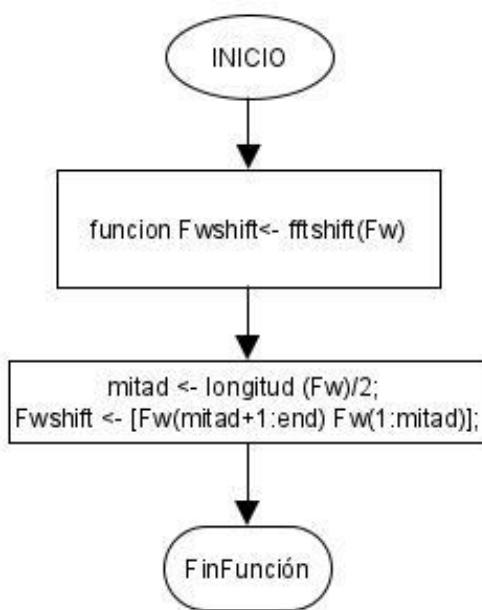


Fig. 34.1. Diagrama de flujo de la función.

```

1 //Nombre: fftshift
2 //Parámetros: FW es una matriz
3 //Devuelve una transformada de Fourier X desplazando el componente de frecuencia cero al centro de la matriz.
4
5 Algoritmo fftshift
6
7     función Fwshift<-fftshift(Fw)
8         mitad <- longitud (Fw)/2;
9         Fwshift <- [Fw(mitad+1:end) Fw(1:mitad)];
10    finFunción
11 FinAlgoritmo

```

Fig. 34.2. Función en pseudolenguaje pse.



```

def fftshift(Fw):
    Fwshift = np.fft.fftshift(Fw)
    return Fwshift

```

Fig. 34.3. Algoritmo en Python.

35. Función iift2

35.1 Uso

Ifft2 Evalúa la transformada inversa de Fourier en 2D.

La sintaxis para su uso es:

- Ifft2(I)

Donde: "I" es la imagen.

Devuelve la inversa de DFT (la transformada discreta de Fourier en 2D) de la imagen, la cual es evaluada con el algoritmo multidimensional de la FFT.

35.2 Resultado:

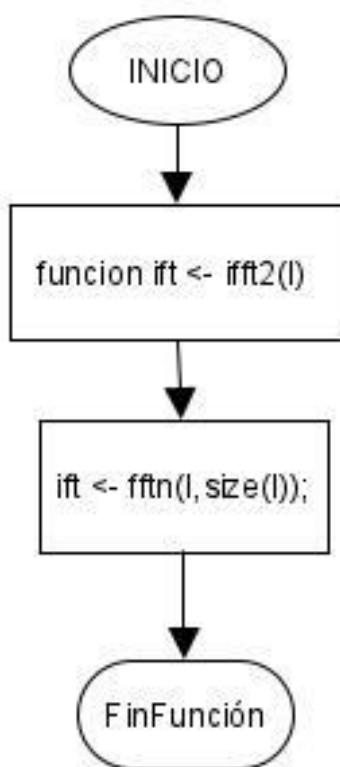


Fig. 35.1. Diagrama de flujo de función.

```

1 //Nombre: ifft2
2 //Parámetros: Donde I es la imagen,
3 //Devuelve la transformada inversa de Fourier de una imagen en 2 dimensiones.
4
5 Algoritmo ifft2
6
7     función ift <- ifft2(I)
8         ift <- ifftn(I, tamaño(I));
9
10    FinFunción
11
12 FinAlgoritmo

```

Fig. 35.2. Función en pseudolenguaje pse.



```

def ifft2(I):
    x, y = I.shape
    ift = np.fft.ifft2(I, (x,y))
    return ift

```

Fig. 35.3. Algoritmo en Python.

36. Ejemplos

Los siguientes ejemplos muestran el filtrado en el dominio de la frecuencia con las funciones mencionadas anteriormente, *fft2*, *fftshift* e *ifft2*.

36.1 Filtro pasa bajo ideal.

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:

- A=imread('marily_einstein.jpg')

2. Se cambia la imagen leída a escala de grises usando la instrucción *rgb2gray*, se almacena en una variable “B” y se muestra con la instrucción *imshow*.

- $B=rgb2gray(A)$
- *imshow(A)*

3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción *fft2*:

- $FT=fft2(B)$
- *imagesc(log(abs(FT)))*
- *colormap(copper)*

4. Se evalúa el espectro de potencia centrado usando la instrucción *fftshift*:

- $FTS=fftshift(FT)$
- *imagesc(log(abs(FTS)))*
- *colormap(copper)*

5. Se crea una plantilla para formar la imagen de filtro:

- $[r c]=size(B)$
- $[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)$
- $f=0.07$
- $R=floor(c/2*f)$

6. Se evalúa el filtro pasa bajo ideal, se almacena en una variable “F” y se muestra con la instrucción *imshow*.

- $F=X.^2+Y.^2 < R.^2$
- *imshow(F)*

7. Se realiza el filtrado:

- $filter=log(abs(FTS)).*F$
- *imagesc(filter);*
- *colormap(copper)*

8. Se evalúa la transformada inversa de Fourier con la instrucción *ifft2*, se almacena en una variable “IF” y se muestra con la instrucción *imshow*.

- $IF=ifft2(double(fftshift(FT)).*double(F))$
- $ID=abs(IF)$
- *imshow(ID)*

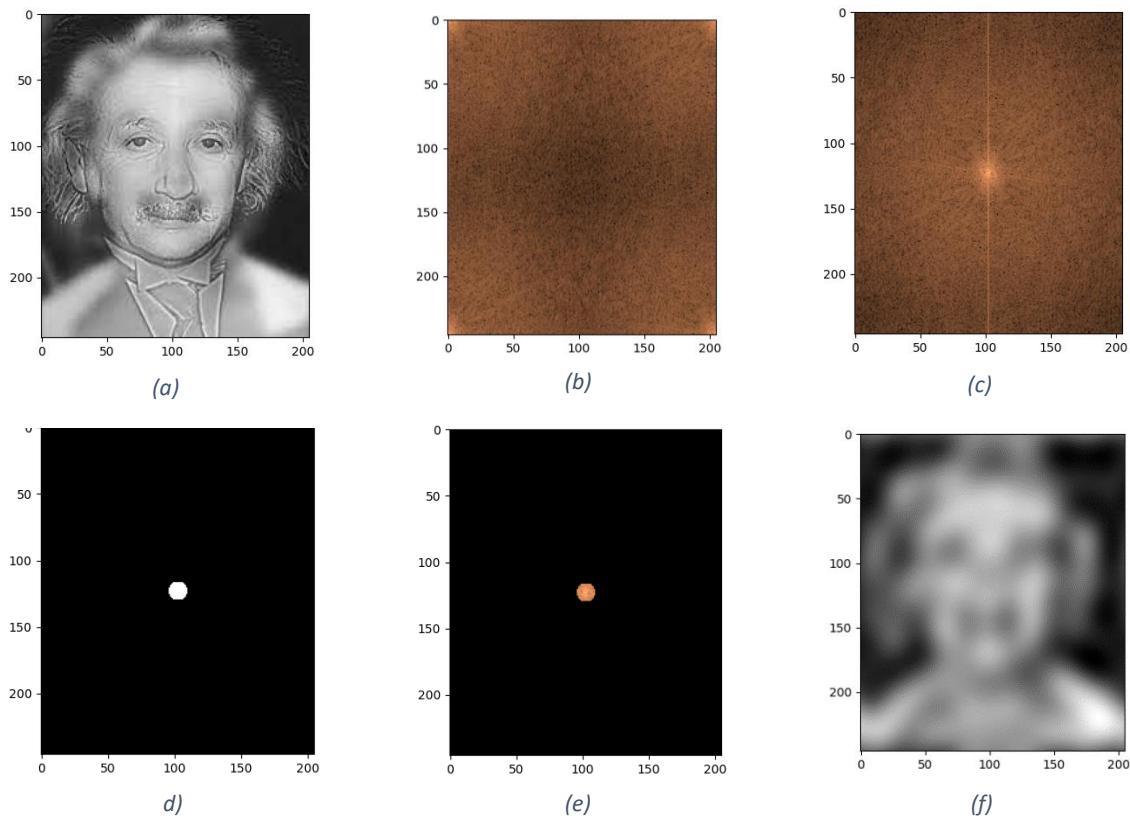


Fig. 36.1. Filtro pasa bajo ideal⁹

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado.
 (d) Filtro pasa bajo ideal. (e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

36.2 Filtro pasa bajo Butterworth.

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:

- `A=imread('marily_einstein.jpg')`

2. Se cambia la imagen leída a escala de grises usando la instrucción `rgb2gray`, se almacena en una variable “B” y se muestra con la instrucción `imshow`:

- `B=rgb2gray(A)`

⁹Imagen (a), tomada de (Desconecta, 2015)

- `imshow(A)`

3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción `fft2` y se almacena en una variable “FT”.

- `FT=fft2(B)`
- `imagesc(log(abs(FT)))`
- `colormap(copper)`

4. Se evalúa el espectro de potencia centrado usando la instrucción `fftshift` y se almacena en una variable “FTS”.

- `FTS=fftshift(FT)`
- `imagesc(log(abs(FTS)))`
- `colormap(copper)`

5. Se crea una plantilla para formar la imagen de filtro.

- `[r c]=size(B)`
- `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
- `f=0.07`
- `R=floor(c/2*f)`

6. Se evalúa el filtro pasa bajo Butterworth.

- `n=1`
- `F1=X.^2+Y.^2`
- `F=1./(1+(F1./R.^2).^(2*n))`
- `imshow(F)`

7. Se realiza el filtrado y se almacena en una variable “filter”.

- `filter=log(abs(FTS)).*F`
- `imagesc(filter)`
- `colormap(copper)`

8. Se evalúa la transformada inversa de Fourier con la instrucción `ifft2`, se almacena en una variable “IF” y se muestra con la instrucción `imshow`.

- `IF=ifft2(double(fftshift(FT)).*double(F))`
- `ID=abs(IF)`
- `imshow(ID)`

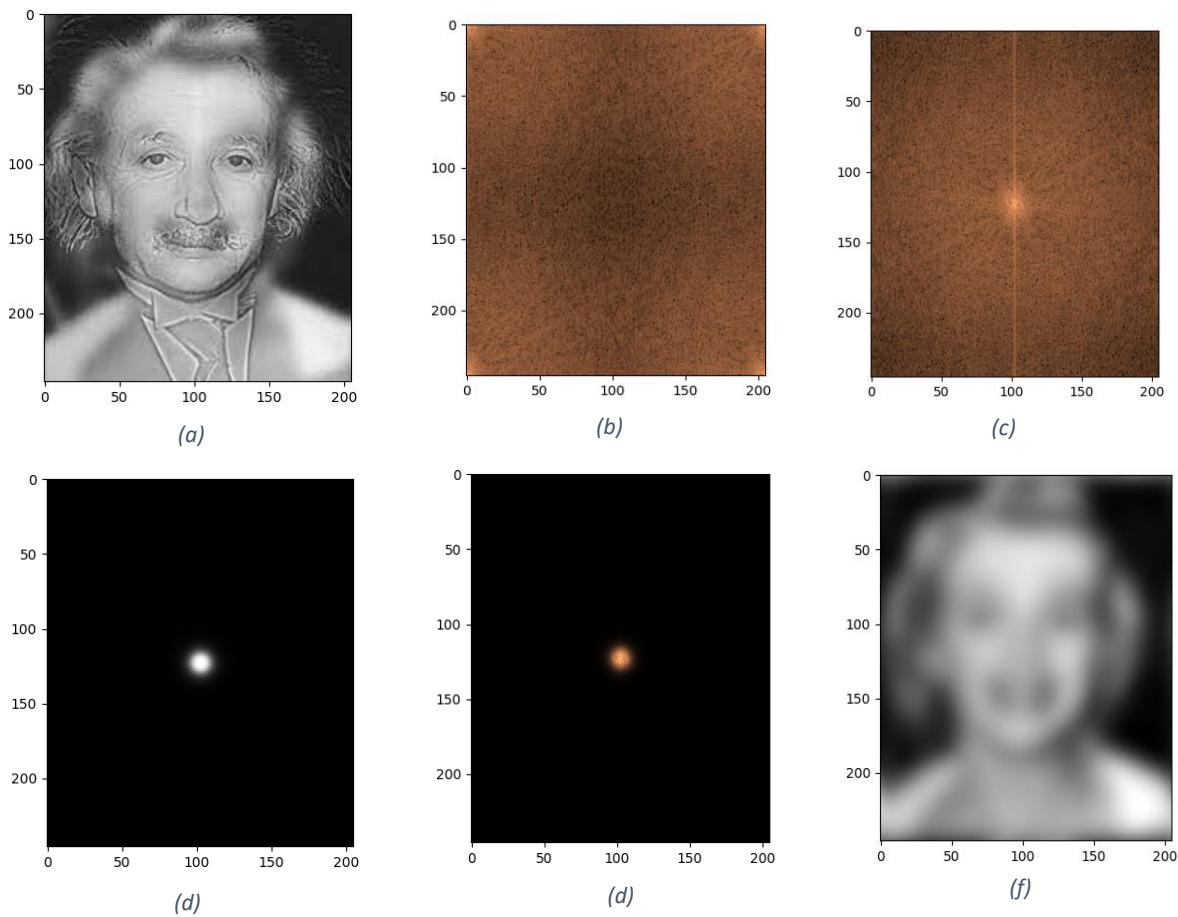


Fig. 36.2. Filtro pasa bajo ideal¹⁰

- (a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado.
 (d) Filtro pasa bajo ideal (e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

36.3 Filtro pasa bajo Gaussiano.

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:
 - `A=imread('marily_einstein.jpg')`
2. Se cambia la imagen leída a escala de grises usando la instrucción `rgb2gray`, se almacena en una variable “B” y se muestra con la instrucción `imshow`.

¹⁰ Imagen (a), tomada de (Desconecta, 2015)

- `B=rgb2gray(A)`
- `imshow(A)`

3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción `fft2` y se almacena en una variable “FT”.

- `FT=fft2(B)`
- `imagesc(log(abs(FT)))`
- `colormap(copper)`

4. Se evalúa el espectro de potencia centrado con la instrucción `fftshift` y se almacena en una variable “FTS”.

- `FTS=fftshift(FT)`
- `imagesc(log(abs(FTS)))`
- `colormap(copper)`

5. Se crea una plantilla para formar la imagen de filtro.

- `[r c]=size(B)`
- `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
- `f=0.07`
- `R=floor(c/2*f)`

6. Se evalúa el filtro pasa bajo Gaussiano, se almacena en una variable “F” y se muestra con la instrucción `imshow`.

- `F1=X.^2+Y.^2`
- `F=exp(-F1/(2*R^2))`
- `imshow(F)`

7. Se realiza el filtrado:

- `filter=log(abs(FTS)).*F`
- `imagesc(filter)`
- `colormap(copper);`

8. Se evalúa la transformada inversa de Fourier, se almacena en una variable “IF” y se muestra usando la instrucción `imshow`.

- `IF=ifft2(double(fftshift(FT)).*double(F))`
- `ID=abs(IF)`
- `imshow(ID)`

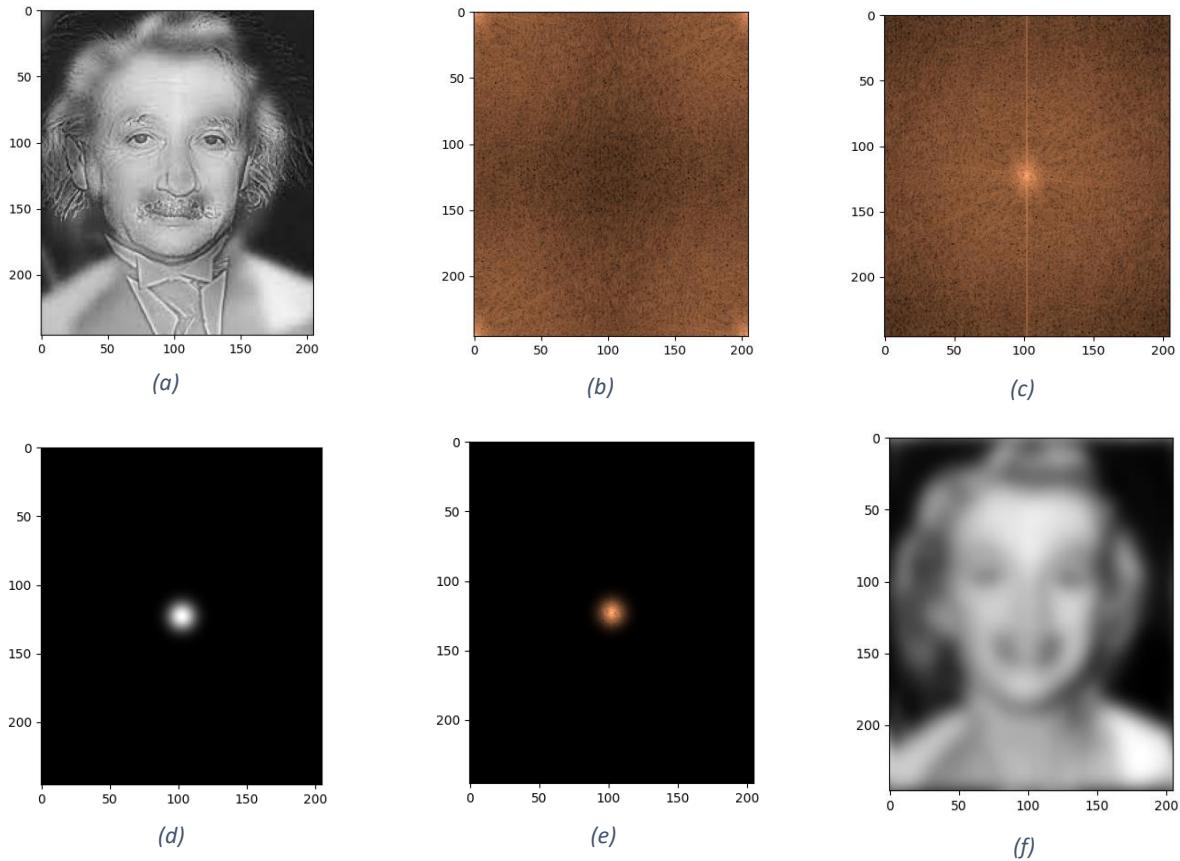


Fig. 36.3. Filtro pasa bajo Gaussiano¹¹

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado. (d) Filtro pasa bajo Gaussiano. (e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

36.4 Filtro pasa alto ideal.

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:
 - A=imread('marily_einstein.jpg')
2. Se cambia la imagen leída a escala de grises usando la instrucción *rgb2gray*, se almacena en una variable “B” y se muestra usando la instrucción *imshow*.

¹¹Imagen (a), tomada de (Desconecta, 2015)

- `B=rgb2gray(A)`
- `imshow(A)`

3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción `fft2` y se almacena en una variable “FT”.

- `FT=fft2(B)`
- `imagesc(log(abs(FT)))`
- `colormap(copper)`

4. Se evalúa el espectro de potencia centrado con el comando `fftshift` y se almacena en una variable “FTS”.

- `FTS=fftshift(FT)`
- `imagesc(log(abs(FTS)))`
- `colormap(copper)`

5. Se crea una plantilla para formar la imagen de filtro.

- `[r c]=size(B)`
- `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
- `f=0.07`
- `R=floor(c/2*f)`

6. Se evalúa el filtro pasa alto ideal, se almacena en una variable “F” y se muestra con la instrucción `imshow`.

- `F=X.^2+Y.^2>R^2`
- `imshow(F)`

7. Se realiza el filtrado y se almacena en una variable “filter”.

- `filter=log(abs(FTS)).*F`
- `imagesc(filter)`
- `colormap(copper)`

8. Se evalúa la transformada inversa de Fourier, se almacena en una variable “IF” y se muestra con la instrucción `imshow`.

- `IF=ifft2(double(fftshift(FT)).*double(F))`
- `ID=abs(IF)`
- `imshow(ID)`

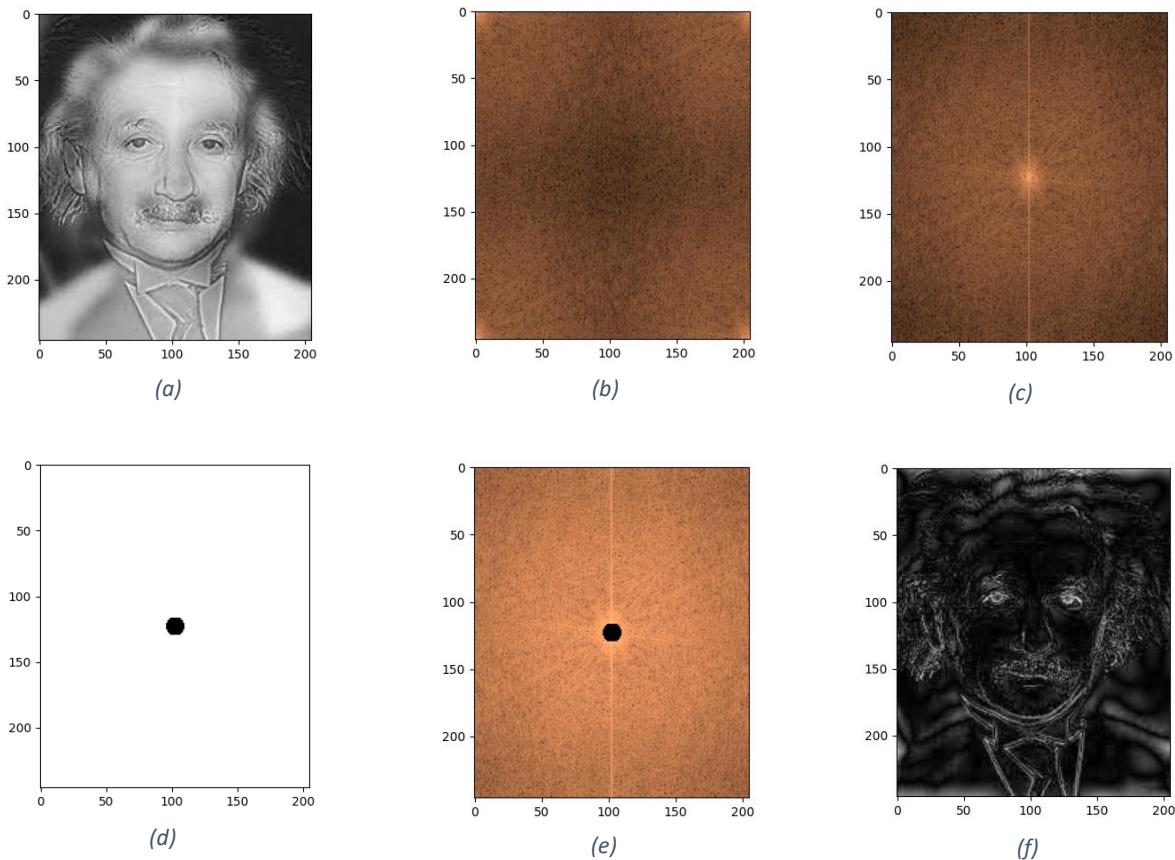


Fig. 36.4. Filtro pasa alto ideal.¹²

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. c) Espectro de potencia centrado. (d) Filtro pasa alto ideal. (e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

36.5 Filtro pasa alto Butterworth.

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:
 - A=imread('marily_einstein.jpg')
2. Se cambia la imagen leída a escala de grises usando la instrucción *rgb2gray*, se almacena en una variable “B” y se muestra usando la instrucción *imshow*.
 - B=rgb2gray(A)
 - imshow(A)
3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción *fft2*.

¹² Imagen (a) tomada de (Desconecta, 2015)

- $\text{FT}=\text{fft2}(B)$
- $\text{imagesc}(\log(\text{abs}(\text{FT})))$
- colormap(copper) ;

4. Se evalúa el espectro de potencia centrado con la instrucción *fftshift*.

- $\text{FTS}=\text{fftshift}(\text{FT})$;
- $\text{imagesc}(\log(\text{abs}(\text{FTS})))$;
- colormap(copper) ;

5. Se crea una plantilla para formar la imagen de filtro.

- $[r c]=\text{size}(B)$
- $[X Y]=\text{meshgrid}(-c/2:c/2-1, -r/2:r/2-1)$
- $f=0.07$
- $R=\text{floor}(c/2*f)$

6. Se evalúa el filtro pasa alto Butterworth y se muestra con la instrucción *imshow*.

- $n=1$
- $F1=X.^2+Y.^2$
- $F=1./(1+(R.^2./F1).^(2*n))$
- $\text{imshow}(F)$

7. Se realiza el filtrado:

- $\text{filter}=\log(\text{abs}(\text{FTS})).*F$
- $\text{imagesc}(\text{filter})$
- colormap(copper)

8. Se evalúa la transformada inversa de Fourier y se muestra con la instrucción *imshow*.

- $\text{IF}=\text{ifft2}(\text{double}(\text{fftshift}(\text{FT})).*\text{double}(F))$
- $\text{ID}=\text{abs}(\text{IF})$
- $\text{imshow}(\text{ID})$

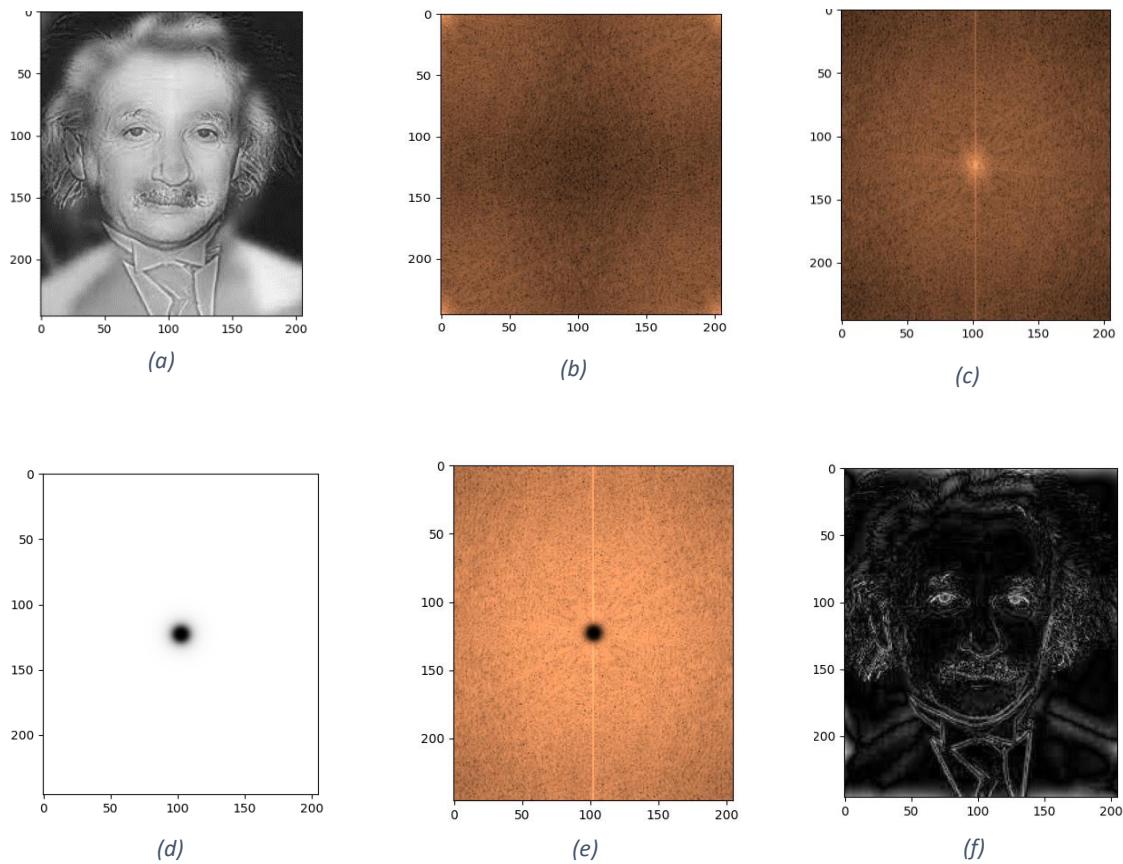


Fig. 36.5.: Filtro pasa alto Butterworth.¹³

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado. (d) Filtro pasa alto Butterworth. e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

36.6 Filtro pasa alto Gausiano

1. Se lee una imagen llamada “marily_einstein.jpg” y se almacena en una variable “A”:

- `A=imread('marily_einstein.jpg')`

2. Se cambia la imagen leída a escala de grises usando la instrucción `rgb2gray`, se almacena en una variable “B” y se muestra con la instrucción `imshow`.

- `B=rgb2gray(A)`

¹³ Imagen (a), tomada de (Desconecta, 2015)

- `imshow(A)`
3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción `fft2`.
- `FT=fft2(B)`
 - `imagesc(log(abs(FT)))`
 - `colormap(copper);`
4. Se evalúa el espectro de potencia centrado con el comando `fftshift`.
- `FTS=fftshift(FT)`
 - `imagesc(log(abs(FTS)))`
 - `colormap(copper);`
5. Se crea una plantilla para formar la imagen de filtro.
- `[r c]=size(B)`
 - `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
 - `f=0.07`
 - `R=floor(c/2*f)`
6. Se evalúa el filtro pasa alto Gaussiano, se almacena en una variable “F1” y se muestra con la instrucción `imshow`.
- `F1=X.^2+Y.^2`
 - `F=1-exp(-F1/(2*R^2))`
 - `imshow(F)`
7. Se realiza el filtrado.
- `filter=log(abs(FTS)).*F`
 - `imagesc(filter)`
 - `colormap(copper)`
8. Se evalúa la transformada inversa de Fourier, se almacena en una variable “IF” y se muestra con la instrucción `imshow`.
- `IF=ifft2(double(fftshift(FT)).*double(F))`
 - `ID=abs(IF)`
 - `imshow(ID)`

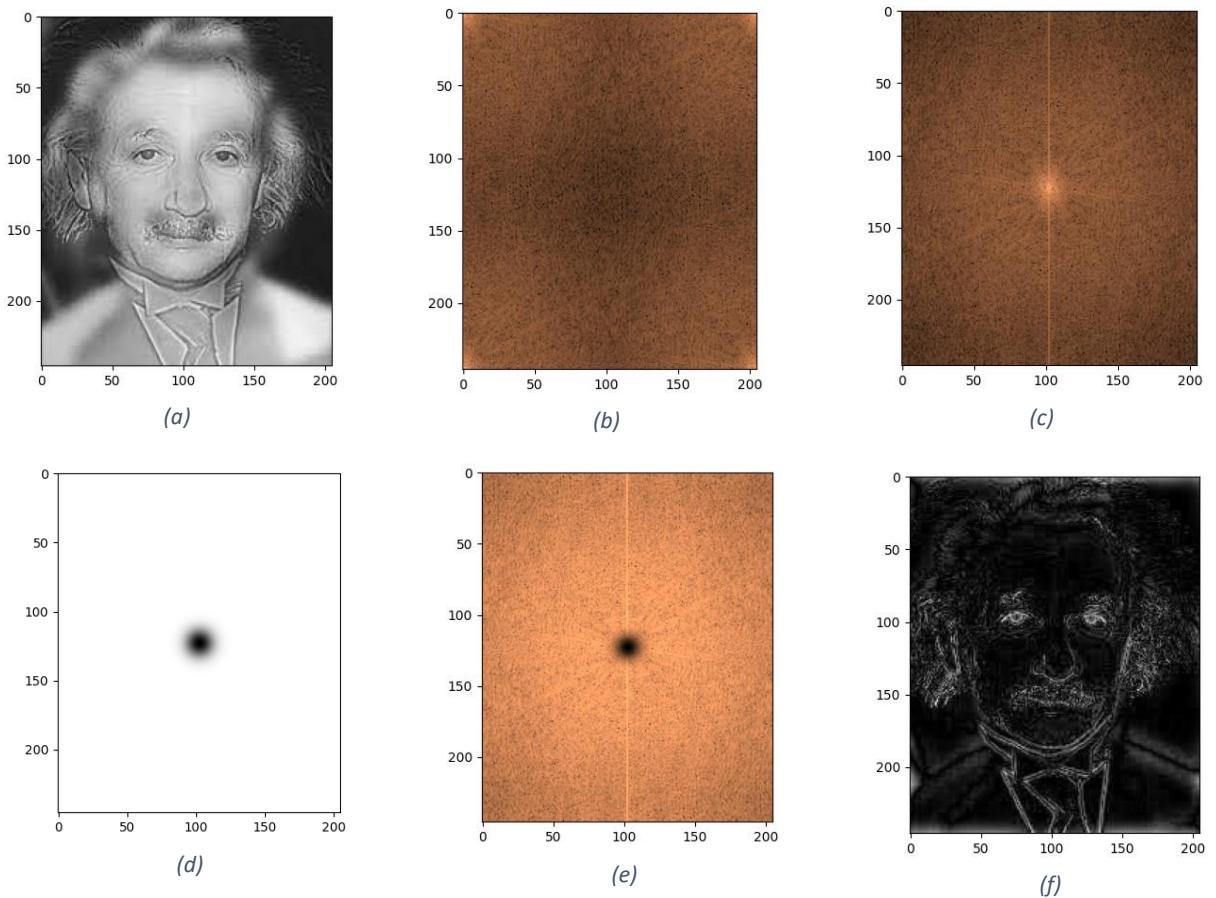


Fig. 36.6. Filtro pasa alto Gaussiano.¹⁴

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado. (d) Filtro pasa alto Gaussiano. (e) Producto entre espectro centrado y filtro. (f)Imagen de salida.

36.7 Filtro eliminador de banda

1. Se lee una imagen llamada “marily_einstein_seno.jpg” y se almacena en una variable “A”:

- A=imread('marily_einstein_seno.jpg')

2. Se cambia la imagen leída a escala de grises usando la instrucción *rgb2gray*, se almacena en una variable “B” y se muestra usando la instrucción *imshow*.

¹⁴ Imagen (a), tomada de (Desconecta, 2015)

- `B=rgb2gray(A)`
- `imshow(A)`

3. Se evalúa el espectro de potencia de la imagen leída usando la instrucción `fft2`:

- `FT=fft2(B)`
- `imagesc(log(abs(FT)))`
- `colormap(copper)`

4. Se evalúa el espectro de potencia centrado con el comando `fftshift`:

- `FTS=fftshift(FT)`
- `imagesc(log(abs(FTS)))`
- `colormap(copper);`

5. Se crea una plantilla para formar la imagen de filtro.

- `[r c]=size(B)`
- `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
- `f=0.07`
- `R=floor(c/2*f)`

6. Se evalúa el filtro eliminador de banda se almacena en una variable “d” y se muestra con la instrucción `imshow`.

- `d=40`
- `F=(X.^2+Y.^2<R^2 | X.^2+Y.^2 >=(R+d)^2)`
- `imshow(F)`

7. Se realiza el filtrado:

- `filter=log(abs(FTS)).*F`
- `imagesc(filter)`
- `colormap(copper)`

9. Se evalúa la transformada inversa de Fourier, se almacena en una variable “IF” y se muestra con la instrucción `imshow`.

- `IF=ifft2(double(fftshift(FT)).*double(F))`
- `ID=abs(IF)`
- `imshow(ID)`

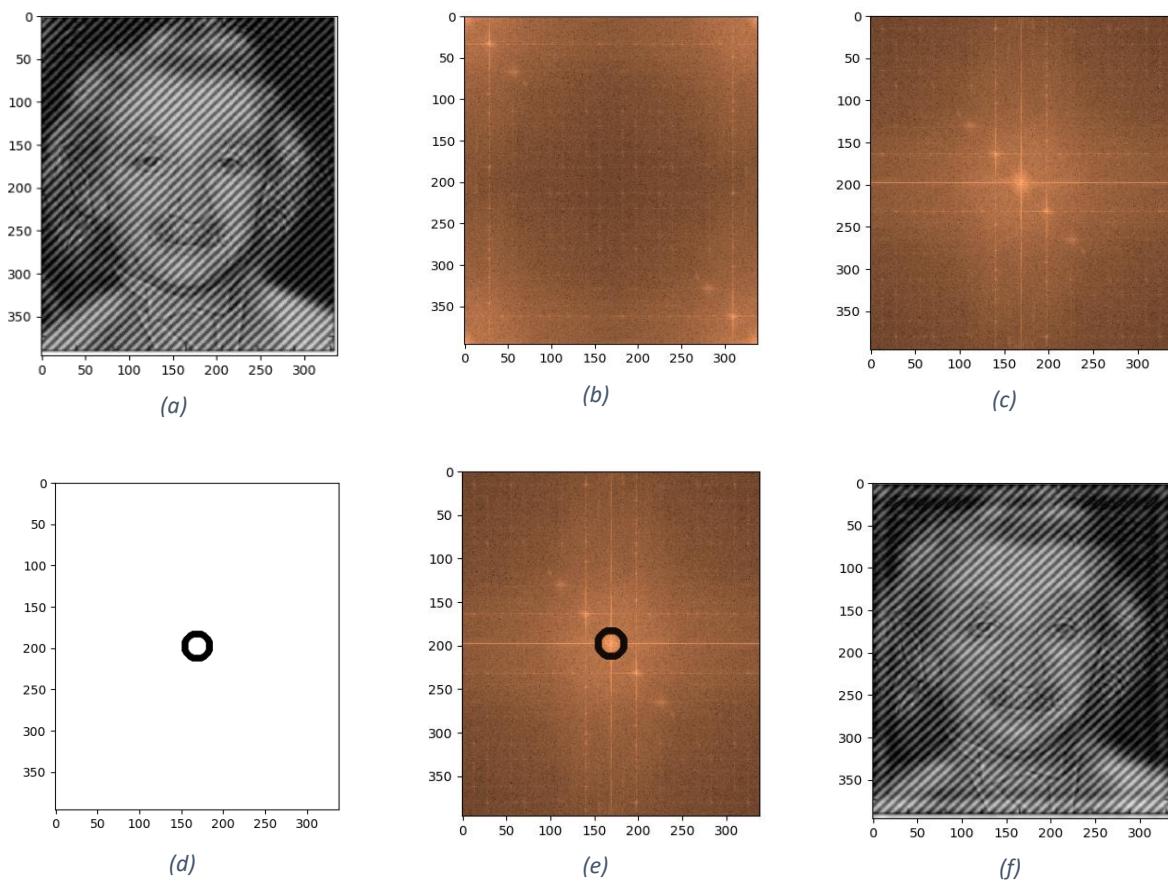


Fig. 36.7. Filtro eliminador de banda¹⁵.

- (a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado.
- (d) Filtro eliminador de banda. e) Producto entre espectro centrado y filtro. (f)Imagen de salida

36.8 Filtro pasa banda.

1. Se lee una imagen llamada “marily_einstein_seno.jpg” y se almacena en una variable “A”:

- `A=imread('marily_einstein_seno.jpg')`

2. Se cambia a escala de grises usando la instrucción `rgb2gray`, se almacena en una variable “B” y se muestra con la instrucción `imshow`:

- `B=rgb2gray(A)`

¹⁵ Imagen (a), tomada de (Google imágenes, 2021)

- `imshow(A)`

3. Se evalúa el espectro de potencia de la imagen usando la instrucción `fft2`.

- `FT=fft2(B)`
- `imagesc(log(abs(FT)))`
- `colormap(copper)`

4. Se evalúa el espectro de potencia centrado usando la instrucción `fftshift`.

- `FTS=fftshift(FT)`
- `imagesc(log(abs(FTS)))`
- `colormap(copper)`

6. Se crea una plantilla para formar la imagen de ltro:

- `r c]=size(B)`
- `[X Y]=meshgrid(-c/2:c/2-1,-r/2:r/2-1)`
- `f=0.07`
- `R=floor(c/2*f)`

7. Se evalúa el filtro pasa banda, se almacena en una variable “d” y se muestra con la instrucción `imshow`.

- `d=30`
- `F=(X.^2+Y.^2<R^2 | X.^2+Y.^2 >=(R+d)^2)`
- `F=~F`
- `imshow(F)`

8. Se realiza el filtrado.

- `filter=log(abs(FTS)).*F`
- `imagesc(filter)`
- `colormap(copper);`

9. Se evalúa la transformada inversa de Fourier, se almacena en una variable “IF”, y se muestra con la instrucción `imshow`.

- `IF=ifft2(double(fftshift(FT)).*double(F))`
- `ID=abs(IF)`
- `imshow(ID)`

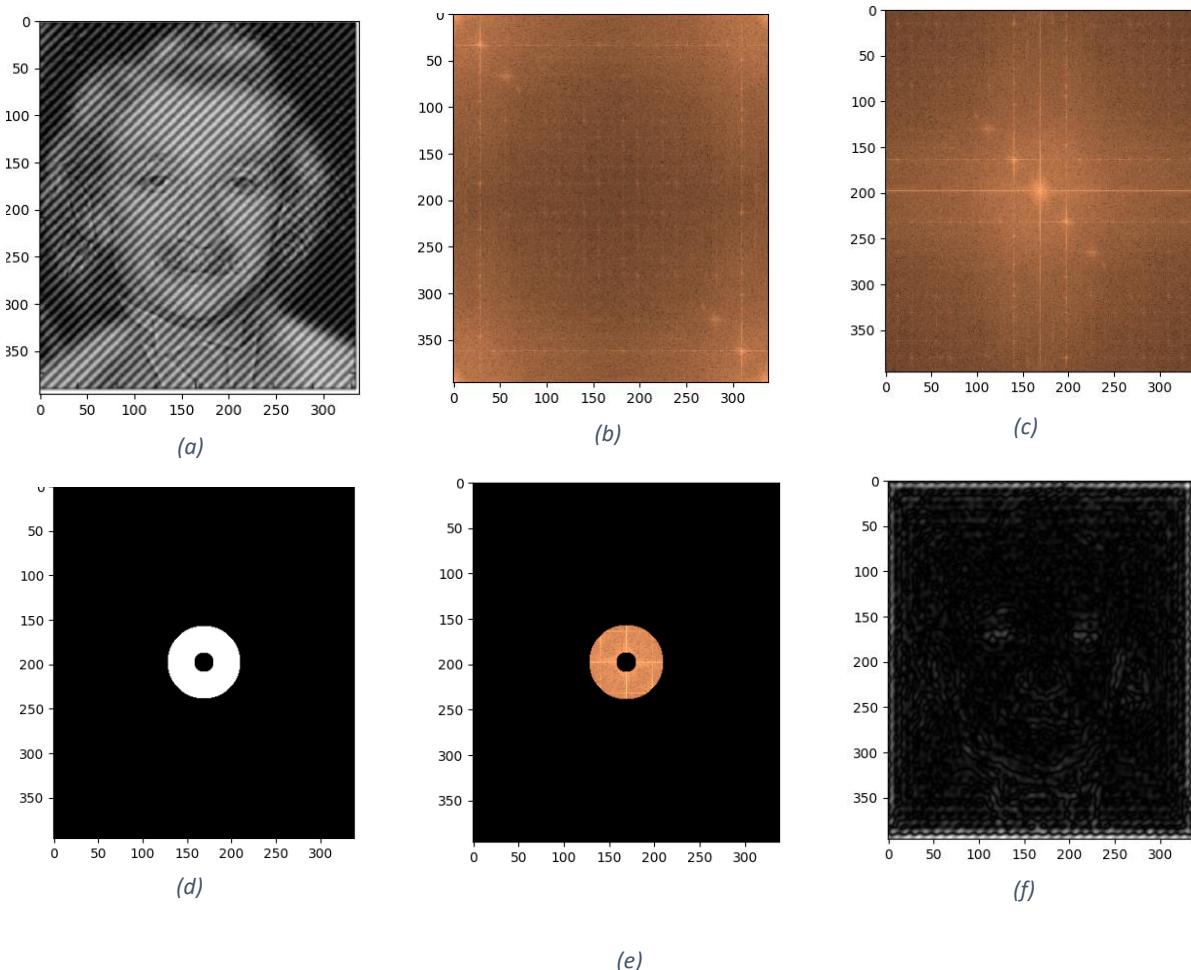


Fig.36.8 Filtro eliminador de banda.¹⁶

(a) Imagen de entrada. (b) Espectro de potencia de la imagen de entrada. (c) Espectro de potencia centrado. d) Filtro eliminador de banda. (e) Producto entre espectro centrado y filtro. (f) Imagen de salida.

37. Función bwperim

37.1 Uso

bwperim Encuentra los objetos del perímetro de una imagen.

La sintaxis para su uso es:

¹⁶ Imagen (a), tomada de (Google imágenes, 2021)

- `bwperim(B, conn = 4)`

Donde: “B” es la imagen binarizada y “conn=4” es la imagen con contorno.

37.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *bwperim*.

1. Se lee una imagen llamada “herramientas.jpg”, y almacena en una variable “RGB”:

- `RGB= imread('herramientas.jpg')`

2. Se cambia a escala de grises y se almacena en una variable “gris”

- `gris=rgb2gray(RGB)`

3. Se binariza la imagen con la instrucción *im2bw* y se almacena en una variable “binaria”.

- `T=graythresh(gris)`
- `binaria=im2bw(gris,T)`

4. Se crea un filtrado morfológico para que la imagen esté libre de píxeles que no tengan vecindad y se muestra usando la instrucción *imshow*.

- `EE=strel('square',3);`
- `dilatada=imdilate(~binaria, EE)`
- `imshow(dilatada)`

5. Se evalúa el contorno de la región en blanco.

- `bwperim(dilatada,4)`

37.3 Resultado.

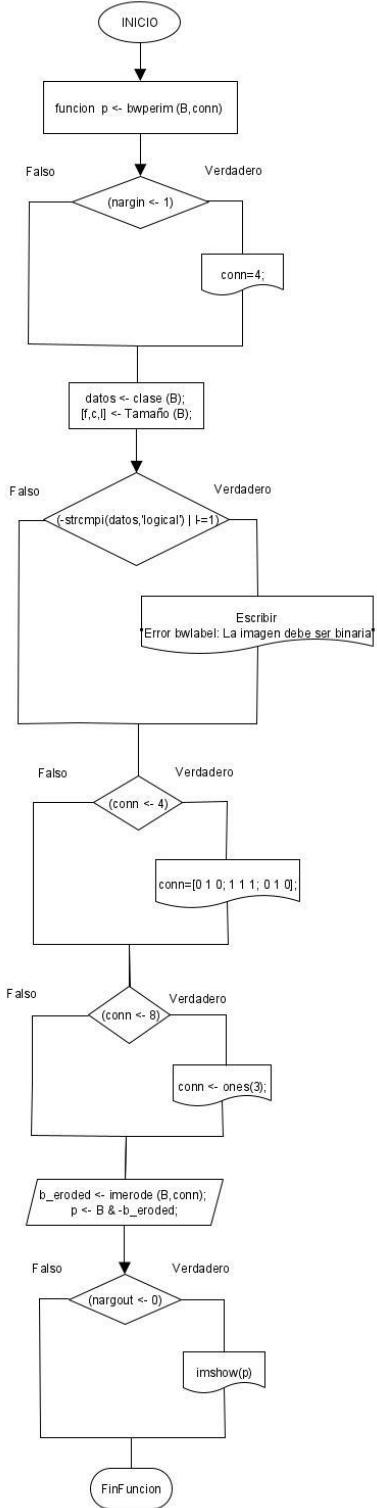


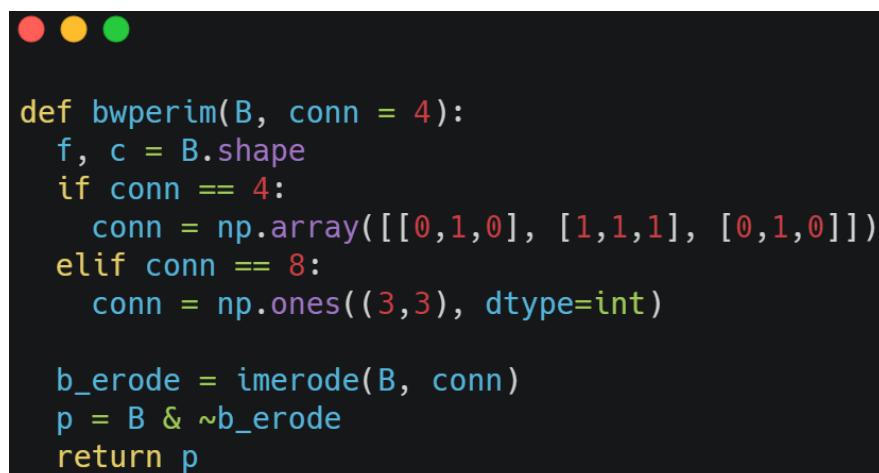
Fig. 37.1 Diagrama de flujo de la función.

```

1 //Nombre: bwperim.
2 // Determina el contorno de una imagen binaria.
3 //Paraámetros: B: Es la imagen binarizada,
4 //              conn: Es el tipo de conector
5 //              p: Es la imagen binaria del contorno (por defecto es 4)
6 //si no se da valor de salida, se muestra la imagen del perimetro directamente.
7 //Retorna: erosiona la imagen binaria,devolviendo la imagen erosionada.
8
9 Algoritmo bwperim
10 funcion p <- bwperim (B,conn)
11
12     Si (nargin <= 1)
13         conn=4;
14     FinSi
15
16     datos <- clase (B);
17     [f,c,l] <- Tamaño (B);
18     Si (-strcmpi(datos,'logical') | l==1)
19         Escribir ("Error bwperim: La imagen debe ser binaria")
20     FinSi
21
22     Si (conn <= 4)
23         conn=[0 1 0; 1 1 1; 0 1 0]; //conexión 4
24     FinSi
25
26     Si (conn <= 8)
27         conn <- ones(3); //conexión 8
28     FinSi
29
30     b_eroded <- imerode (B,conn);
31     p <- B & ~b_eroded;
32
33     Si (nargout <= 0)
34         imshow(p)
35     Finsi
36 FinFuncion
37 FinAlgoritmo

```

Fig. 37.2. Función en pseudolenguaje pse.



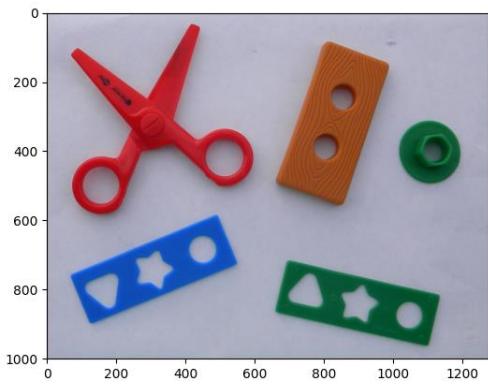
```

def bwperim(B, conn = 4):
    f, c = B.shape
    if conn == 4:
        conn = np.array([[0,1,0], [1,1,1], [0,1,0]])
    elif conn == 8:
        conn = np.ones((3,3), dtype=int)

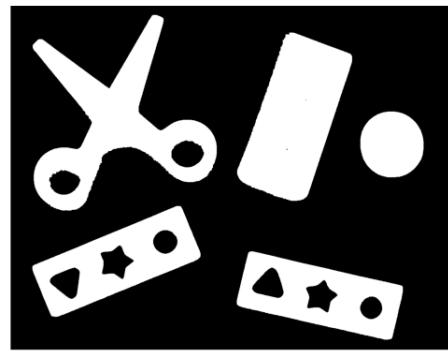
    b_erode = imerode(B, conn)
    p = B & ~b_erode
    return p

```

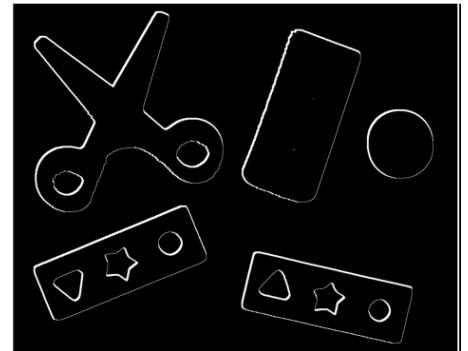
Fig.37.3. Algoritmo en Python.



(a) Imagen de entrada



(b) Imagen binarizada.



(c) Perímetro de la imagen.

Fig.37.4. Imágenes generadas de bwperim en Python.¹⁷

38. Función rgb2HSV.

38.1 Uso.

rgb2HSV Conversión de una imagen en modelo “RGB” a modelo de color “HSV”.

La sintaxis para su uso es:

- `rgb2HSV(I)`

Donde: “I” es la imagen en modelo RGB.

38.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `rgb2HSV`.

¹⁷ Imagen (a), tomada de (Desconecta, 2015)

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”:

- A=imread('herramientas.jpg')

2. Se cambia la imagen al modelo de color “hsv” y se almacena en una variable “hsv”.

- hsv=rgb2hsv(A)

3. Se muestra la imagen con la instrucción *imshow*.

- imshow(HSV)

38.3 Resultado:

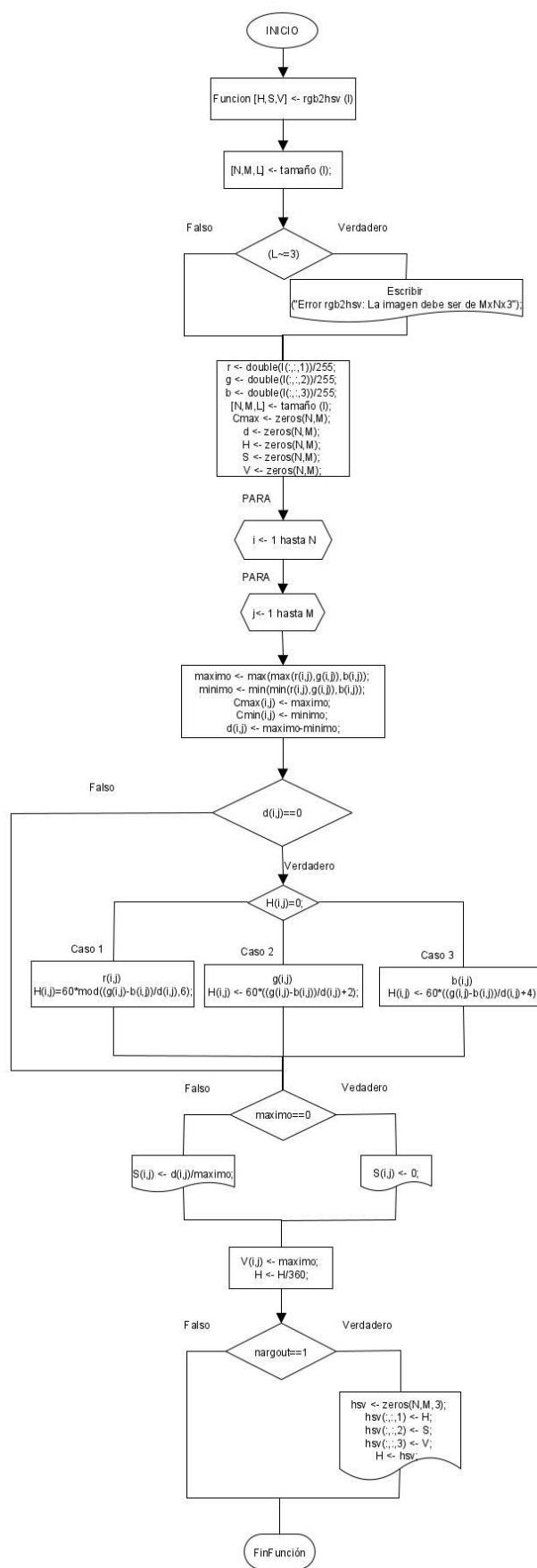


Fig 38.1. Diagrama de flujo de la función

```

1 //Nombre: rgb2hsv
2 //Parámetros:Donde I es la imagen a color
3 //           H,S,V son las matrices de cada una de las capas.
4 //Devuelve la conversión de un mapa de colores RGB en un mapa de colores HSV.
5
6 Algoritmo rgb2hsv
7 Funcion [H,S,V] <- rgb2hsv (I)
8     [N,M,L] <- tamaño (I);
9
10    Si (L~=3)
11        Escribir ("Error rgb2hsv: La imagen debe ser de MxNx3");
12    finsi
13
14    r <- double(I(:,:,1))/255;
15    g <- double(I(:,:,2))/255;
16    b <- double(I(:,:,3))/255;
17    [N,M,L] <- tamaño (I);
18    Cmax <- zeros(N,M);
19    d <- zeros(N,M);
20    H <- zeros(N,M);
21    S <- zeros(N,M);
22    V <- zeros(N,M);
23
24    Para i <- 1 hasta N
25        Para j <- 1 hasta M
26            maximo <- max(max(r(i,j),g(i,j)),b(i,j));
27            minimo <- min(min(r(i,j),g(i,j)),b(i,j));
28            Cmax(i,j) <- maximo;
29            Cmin(i,j) <- minimo;
30            d(i,j) <- maximo-minimo;
31
32
33    Si d(i,j)==0
34        H(i,j)=0;
35    FinSi
36
37    segun maximo hacer
38        caso: r(i,j)
39            caso: r(i,j)
40                H(i,j)=60*mod((g(i,j)-b(i,j))/d(i,j),6);
41
42            caso 2: g(i,j)
43                H(i,j) <- 60*((g(i,j)-b(i,j))/d(i,j)+2);
44
45            caso 3: b(i,j)
46                H(i,j) <- 60*((g(i,j)-b(i,j))/d(i,j)+4);
47        FinSegun
48
49    Si maximo==0
50        S(i,j) <- 0;
51    SiNo
52        S(i,j) <- d(i,j)/maximo;
53    FinSi
54
55    V(i,j) <- maximo;
56    H <- H/360;
57
58    Si nargout==1
59        hsv <- zeros(N,M,3);
60        hsv(:,:,1) <- H;
61        hsv(:,:,2) <- S;
62        hsv(:,:,3) <- V;
63        H <- hsv;
64    FinSi
65
66 FinFuncion
67
68 FinAlgoritmo
69
70 //NOTA:
71 //double: Conversión a un número real.

```

Fig.38.2. Función en pseudolenguaje pse.

```
np.seterr('raise')
np.seterr(divide='ignore', invalid='ignore')

def rgb2hsv(I):
    [N,M,L] = I.shape
    if L != 3:
        print("Error rgb2hsv: La imagen debe ser de MxNx3")

    r = I[...,:]/255.0
    g = I[...,:,1]/255.0
    b = I[...,:,2]/255.0

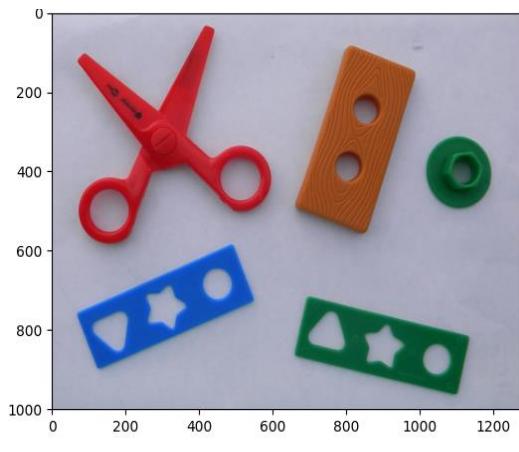
    H, S, V = np.zeros((N, M), np.float32), \np.zeros((N, M), np.float32),  np.zeros((N, M),np.float32)
    hsv = np.zeros([N,M,3], np.float32)
    for i in range(0, N):
        for j in range(0, M):
            maximo = max(r[i, j], g[i, j], b[i, j])
            minimo = min((r[i, j], g[i, j], b[i, j]))

            if max == min:
                H[i,j] = 0
            elif maximo == r[i,j]:
                H[i,j] = 60*np.mod((g[i,j]-b[i,j])/(maximo-minimo), 6)
            elif maximo == g[i,j]:
                H[i,j] = 60*((b[i,j]-r[i,j])/(maximo-minimo) +2)
            elif maximo == b[i,j]:
                H[i,j] = 60*((r[i,j]- g[i,j])/(maximo-minimo)+4)
            H[i, j] = (H[i, j] *255)/ 360

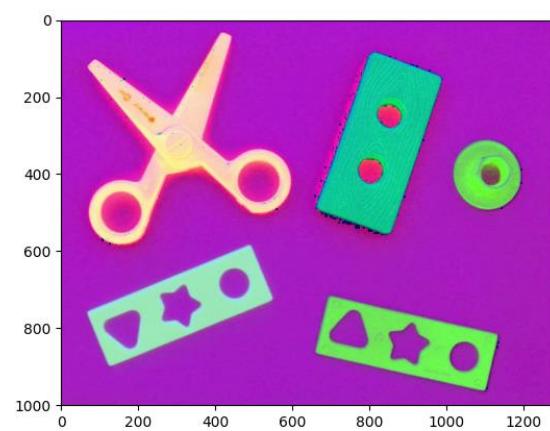
            if maximo == 0:
                S[i,j] = 0
            else:
                S[i,j] = ((maximo-minimo)/maximo*255)

            V[i,j] = (maximo*255)
            hsv[...,:,0] = H
            hsv[...,:,1] = S
            hsv[...,:,2] = V
    return hsv/255.0
```

Fig. 38.3. Algoritmo en Python.



(a) Imagen modelo color RGB.



(b) Imagen convertida a modelo color HSV

Fig.38.4. Conversión de RGB a HSV en Python.

39. Función hsv2rgb.

39.1 Uso.

`hsv2rgb` Conversión de una imagen de modelo “HSV” a modelo de color “RGB”.

La sintaxis para su uso es:

- `hsv2rgb(hsv)`

Donde: “`hsv`” es la imagen en modelo HSV.

39.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `hsv2rgb`.

1. Se lee una imagen llamada “`herramientas.jpg`” y se almacena en una variable “`A`”:

- `A=imread('herramientas.jpg')`

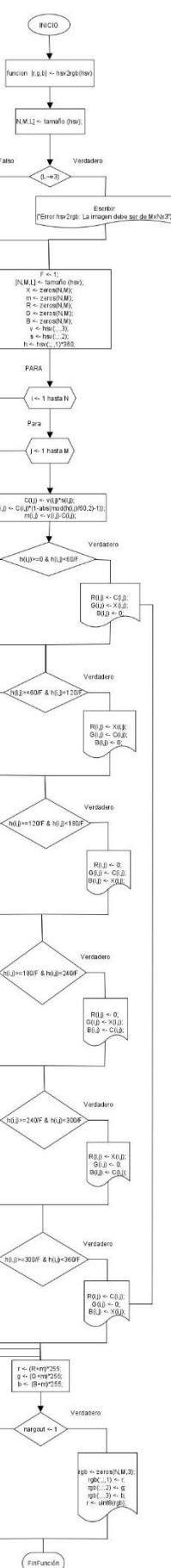
2. Se cambia la imagen al modelo “hsv”, se almacena en la variable “HSV” y se muestra con la instrucción *imshow*.

- HSV=rgb2hsv(A)
- imshow(HSV)

4. Se cambia la imagen de nuevo al modelo RGB, se almacena en una variable “RGB” y se muestra con la instrucción *imshow*.

- RGB= hsv2rgb(HSV)
- imshow(RGB)

39.3 Resultado:



```

1 //Nombre: hsv2rgb
2 //Parámetros: Donde hsv: Es la imagen a color
3 //           r,g,b: Son las matrices de cada una de las capas de color.
4 //
5 Algoritmo hsv2rgb
6 función [r,g,b] <- hsv2rgb(hsv)
7     [N,M,L] <- tamaño (hsv);
8
9     Si (L~=3)
10        Escribir ("Error my_hsv2rgb: La imagen debe ser de MxNx3");
11    FinSi
12
13    F <- 1;
14    [N,M,L] <- tamaño (hsv);
15    X <- zeros(N,M);
16    m <- zeros(N,M);
17    R <- zeros(N,M);
18    G <- zeros(N,M);
19    B <- zeros(N,M);
20    v <- hsv(:,:,3);
21    s <- hsv(:,:,2);
22    h <- hsv(:,:,1)*360;
23
24    Para i <- 1 hasta N
25        Para j <- 1 hasta M
26            C(i,j) <- v(i,j)*s(i,j);
27            X(i,j) <- C(i,j)*(1-abs(mod(h(i,j)/60,2)-1));
28            m(i,j) <- v(i,j)-C(i,j);
29
30            Si h(i,j)>=0 & h(i,j)<60/F
31                R(i,j) <- C(i,j);
32                G(i,j) <- X(i,j);
33                B(i,j) <- 0;
34            SiNo
35
36            Si h(i,j)>=60/F & h(i,j)<120/F
37                R(i,j) <- X(i,j);
38                G(i,j) <- C(i,j);
39                B(i,j) <- 0;
40
41            SiNo
42
43            Si h(i,j)>=120/F & h(i,j)<180/F
44                R(i,j) <- 0;
45                G(i,j) <- C(i,j);
46                B(i,j) <- X(i,j);
47
48            SiNo
49
50            Si h(i,j)>=180/F & h(i,j)<240/F
51                R(i,j) <- 0;
52                G(i,j) <- X(i,j);
53                B(i,j) <- C(i,j);
54
55            SiNo
56
57            Si h(i,j)>=240/F & h(i,j)<300/F
58                R(i,j) <- X(i,j);
59                G(i,j) <- 0;
60                B(i,j) <- C(i,j);
61
62            SiNo
63
64            Si h(i,j)>=300/F & h(i,j)<360/F
65                R(i,j) <- C(i,j);
66                G(i,j) <- 0;
67                B(i,j) <- X(i,j);
68
69            FinSi
70        FinPara
71    FinPara
72
73    ...
74    r <- (R+m)*255;
75    g <- (G+m)*255;
76    b <- (B+m)*255;

```

```
77      . . .
78      Si nargout <- 1
79          rgb <- zeros(N,M,3);
80          rgb(:,:,1) <- r;
81          rgb(:,:,2) <- g;
82          rgb(:,:,3) <- b;
83          r <- uint8(rgb);
84      FinSi
85  FinFuncion
86  FinAlgoritmo
87
88 //NOTA:
89
90 //zeros: Array de ceros.
91 //nargout: Determina el número de salidas de una función
```

Fig.39.2. Función en pseudolenguaje pse.

```

def hsv2rgb(hsv):
    [N,M,L]= hsv.shape

    if (L != 3):
        print ("Error hsv2rgb: La imagen debe ser de MxNx3")

    F=1
    C = np.zeros((N,M), dtype=float)
    X = np.zeros((N,M), dtype=float)
    m = np.zeros((N,M), dtype=float)
    R = np.zeros((N,M), dtype=float)
    G = np.zeros((N,M), dtype=float)
    B = np.zeros((N,M), dtype=float)
    v = hsv[...,:2]
    s = hsv[...,:1]
    h = hsv[...,:1]*360
    for i in range(0, N):
        for j in range(0, M):
            C[i,j] = v[i,j]*s[i,j]
            X[i,j] = C[i,j]*(1-np.abs(((h[i,j]/60)%2)-1))
            m[i,j] = v[i,j]-C[i,j]

            if(h[i,j] >= 0) and (h[i,j] < 60/F):
                R[i,j] = C[i,j]
                G[i,j] = X[i,j]
                B[i,j] = 0

            elif (h[i,j] >= 60/F) and (h[i,j] < 120/F):
                R[i,j] = X[i,j]
                G[i,j] = C[i,j]
                B[i,j] = 0

            elif (h[i,j] >= 120/F) and (h[i,j] < 180/F):
                R[i,j] = 0
                G[i,j] = C[i,j]
                B[i,j] = X[i,j]

            elif (h[i,j] >= 180/F) and (h[i,j] < 240/F):
                R[i,j] = 0
                G[i,j] = X[i,j]
                B[i,j] = C[i,j]

            elif (h[i,j] >= 240/F) and (h[i,j] < 300/F):
                R[i,j] = X[i,j]
                G[i,j] = 0
                B[i,j] = C[i,j]

            elif (h[i,j] >= 300/F) and (h[i,j] < 360/F):
                R[i,j] = C[i,j]
                G[i,j] = 0
                B[i,j] = X[i,j]

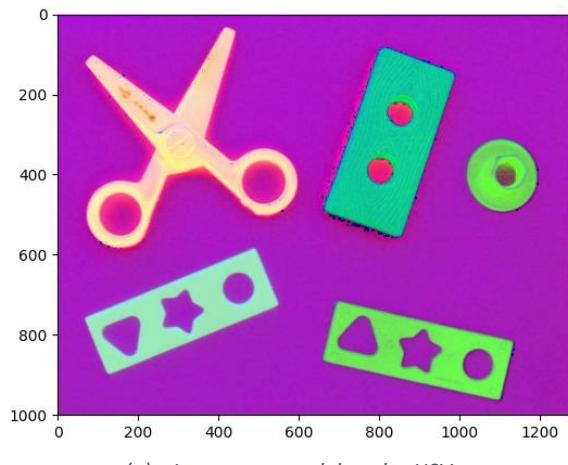
    r = (R+m)*255
    g = (G+m)*255
    b = (B+m)*255

    rgb = np.zeros((N,M,3), dtype=float)
    rgb[...,:0] = r
    rgb[...,:1] = g
    rgb[...,:2] = b

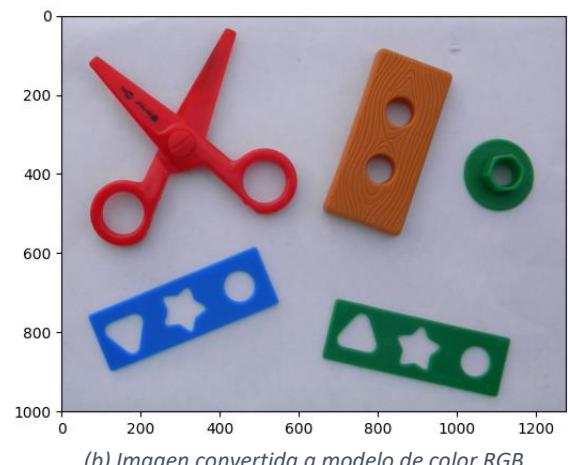
    return rgb/255

```

Fig.39.3. Algoritmo en Python.



(a) Imagen en modelo color HSV.



(b) Imagen convertida a modelo de color RGB.

Fig.39.4. Conversión de HSV a RGB en Python.

40. Función `rgb2xyz`.

40.1 Uso.

`rgb2xyz` Conversión de una imagen de modelo RGB a modelo de color XYZ.

La sintaxis para su uso es:

- `rgb2xyz(RGB)`

Donde: “RGB” es la imagen.

40.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `rgb2xyz`.

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A.”

- `A=imread('herramientas.jpg')`

2. Se cambia la imagen al modelo de color “XYZ” y se almacena en una variable “XYZ”.

- `XYZ=rgb2xyz(A);`

3. Se muestra la imagen con la instrucción *imshow*.

- `imshow(XYZ)`

40.3 Resultado.

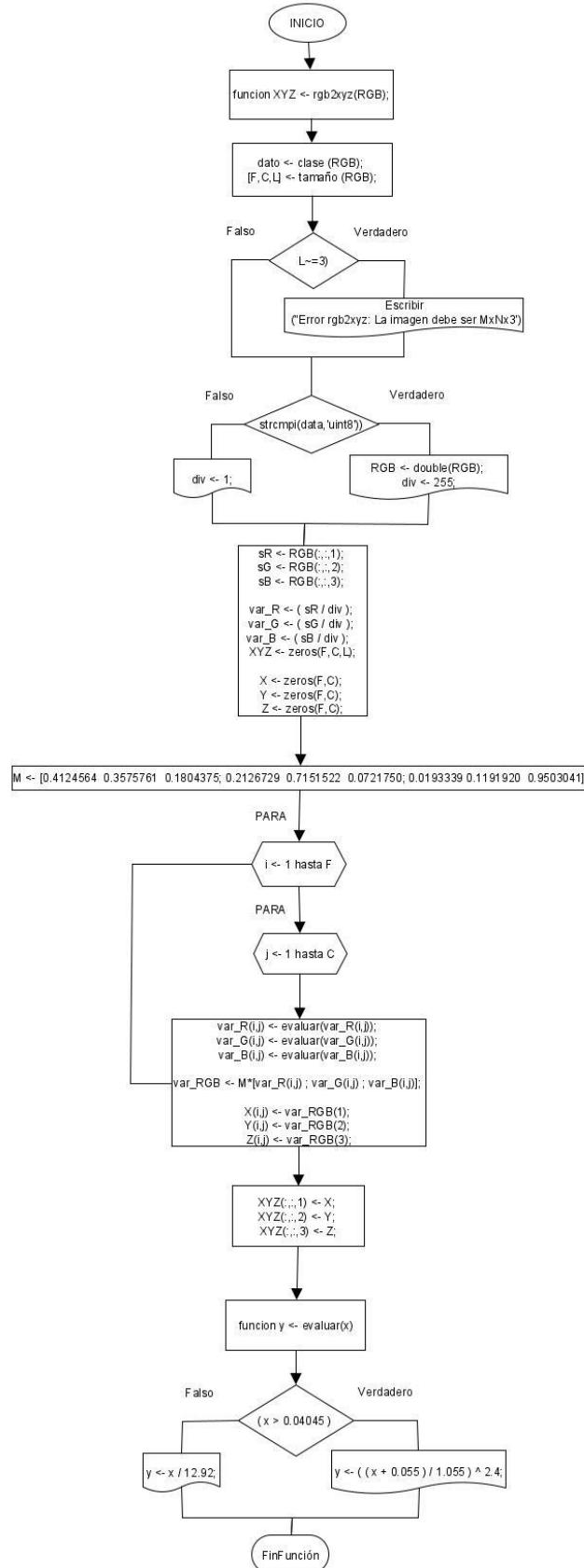


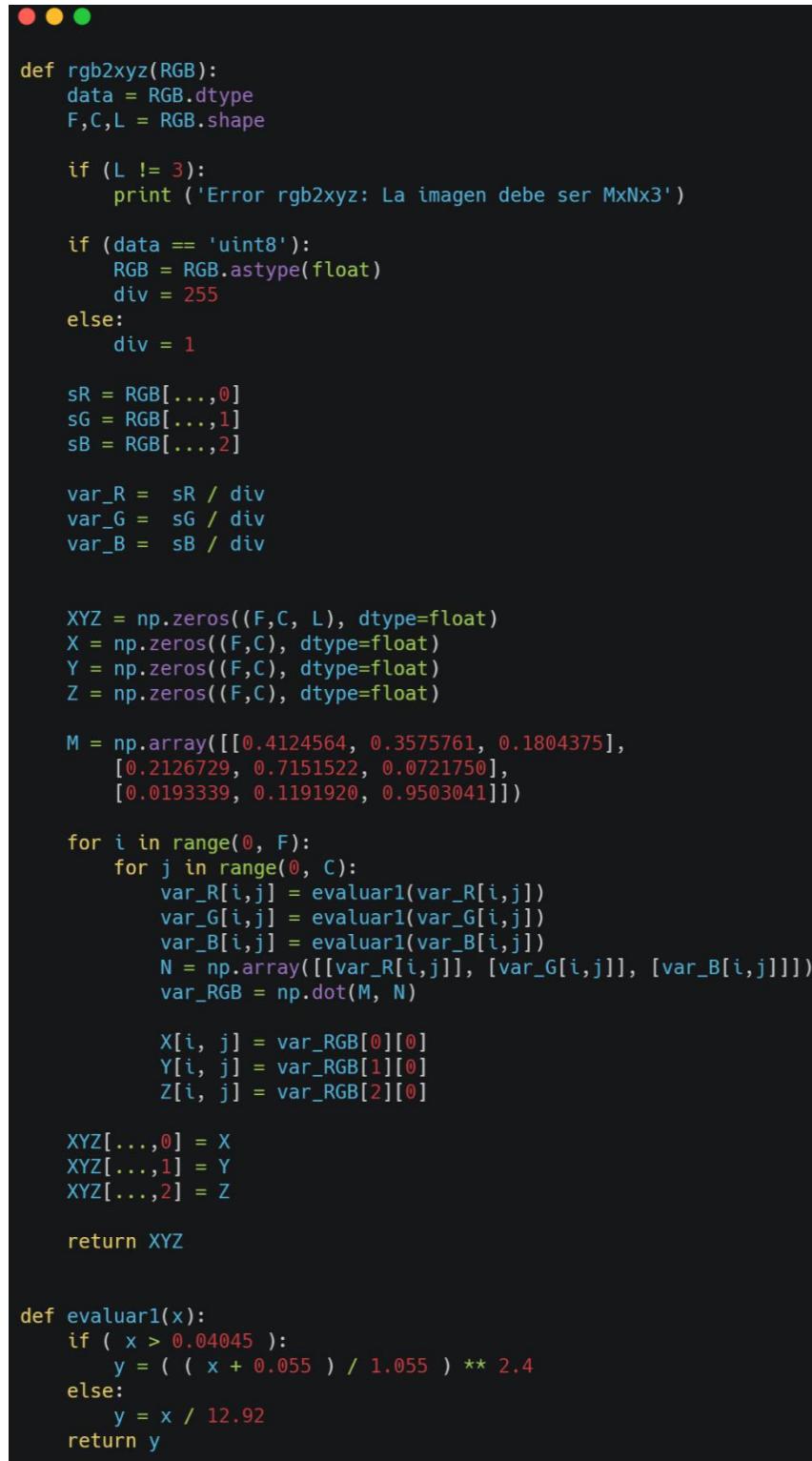
Fig.40.1. Diagrama de flujo de la función.

```

1 //Nombre: rgb2xyz
2 //Parámetros: Donde RGB es la imagen a color
3 //           XYZ es la imagen en el modelo XYZ.
4
5 Algoritmo rgb2xyz
6 funcion XYZ <- rgb2xyz(RGB);
7     dato <- clase (RGB);
8     [F,C,L] <- tamaño (RGB);
9
10    Si (L~=3)
11        Escribir ("Error rgb2xyz: La imagen debe ser MxNx3")
12    FinSi
13
14
15    Si (strcmpi(data,'uint8'))
16        RGB <- double(RGB);
17        div <- 255;
18    SiNo
19        div <- 1;
20    FinSi
21
22
23    sR <- RGB(:,:,1);
24    sG <- RGB(:,:,2);
25    sB <- RGB(:,:,3);
26
27    var_R <- ( sR / div );
28    var_G <- ( sG / div );
29    var_B <- ( sB / div );
30
31
32    XYZ <- zeros(F,C,L);
33    X <- zeros(F,C);
34    Y <- zeros(F,C);
35    Z <- zeros(F,C);
36
37    M <- [0.4124564  0.3575761  0.1804375; 0.2126729  0.7151522  0.0721750; 0.0193339 0.1191920  0.9503041];
38
39    Para i <- 1 hasta F
40        Para j <- 1 hasta C
41            var_R(i,j) <- evaluar(var_R(i,j));
42            var_G(i,j) <- evaluar(var_G(i,j));
43            var_B(i,j) <- evaluar(var_B(i,j));
44
45            var_RGB <- M*[var_R(i,j) ; var_G(i,j) ; var_B(i,j)];
46
47            X(i,j) <- var_RGB(1);
48            Y(i,j) <- var_RGB(2);
49            Z(i,j) <- var_RGB(3);
50
51        FinPara
52    FinPara
53
54    XYZ(:,:,1) <- X;
55    XYZ(:,:,2) <- Y;
56    XYZ(:,:,3) <- Z;
57
58
59    funcion y <- evaluar(x)
60
61        Si( x > 0.04045 )
62            y <- ( ( x + 0.055 ) / 1.055 ) ^ 2.4;
63        SiNo
64            y <- x / 12.92;
65        FinSi
66
67 FinFuncion
68 FinAlgoritmo

```

Fig.40.2. Función en pseudolenguaje pse.



```

def rgb2xyz(RGB):
    data = RGB.dtype
    F,C,L = RGB.shape

    if (L != 3):
        print ('Error rgb2xyz: La imagen debe ser MxNx3')

    if (data == 'uint8'):
        RGB = RGB.astype(float)
        div = 255
    else:
        div = 1

    sR = RGB[...,:0]
    sG = RGB[...,:1]
    sB = RGB[...,:2]

    var_R = sR / div
    var_G = sG / div
    var_B = sB / div

    XYZ = np.zeros((F,C, L), dtype=float)
    X = np.zeros((F,C), dtype=float)
    Y = np.zeros((F,C), dtype=float)
    Z = np.zeros((F,C), dtype=float)

    M = np.array([[0.4124564, 0.3575761, 0.1804375],
                  [0.2126729, 0.7151522, 0.0721750],
                  [0.0193339, 0.1191920, 0.9503041]])

    for i in range(0, F):
        for j in range(0, C):
            var_R[i,j] = evaluar1(var_R[i,j])
            var_G[i,j] = evaluar1(var_G[i,j])
            var_B[i,j] = evaluar1(var_B[i,j])
            N = np.array([[var_R[i,j]], [var_G[i,j]], [var_B[i,j]]])
            var_RGB = np.dot(M, N)

            X[i, j] = var_RGB[0][0]
            Y[i, j] = var_RGB[1][0]
            Z[i, j] = var_RGB[2][0]

    XYZ[...,:0] = X
    XYZ[...,:1] = Y
    XYZ[...,:2] = Z

    return XYZ

def evaluar1(x):
    if ( x > 0.04045 ):
        y = ( ( x + 0.055 ) / 1.055 ) ** 2.4
    else:
        y = x / 12.92
    return y

```

Fig.40.3. Agoritmo en Python.

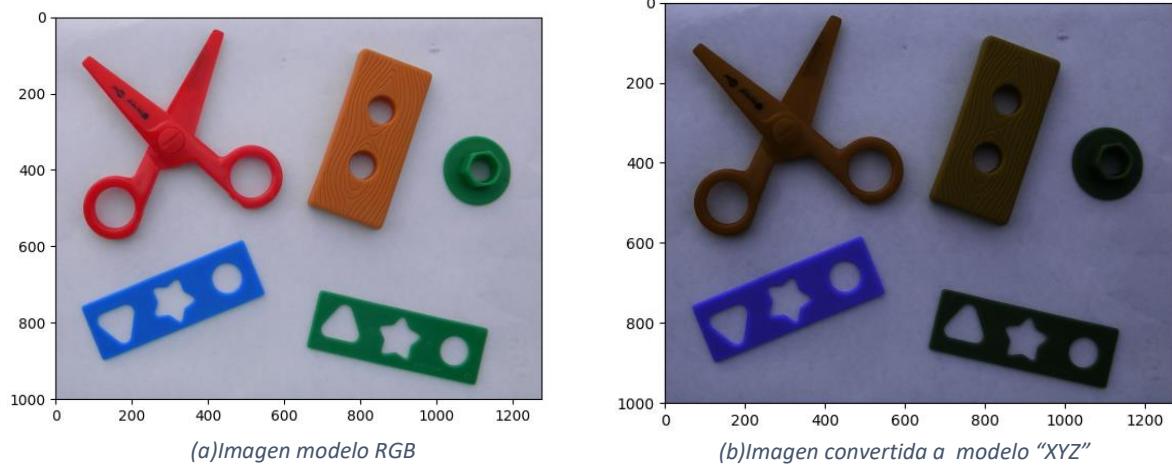


Fig 40.4. Conversión de RGB a XYZ en Python.¹⁸

41. Función xyz2rgb.

41.1 Uso.

`xyz2rgb` Conversión de una imagen de modelo XYZ a modelo de color RGB.

La sintaxis para su uso es:

- `xyz2rgb(XYZ)`

Donde: XYZ es la imagen.

41.2 Ejemplo:

¹⁸ Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *xyz2rgb*.

1. Se lee la imagen llamada “herramientas.jpg” y se almacena en una variable “A”

- `A=imread('herramientas.jpg')`

2. Se cambia la imagen al modelo “xyz”, se almacena en la variable “XYZ” y se muestra la imagen con la instrucción *imshow*.

- `XYZ = rgb2xyz(A)`
- `imshow(XYZ)`

3. Se cambia la imagen al modelo “RGB”, se almacena en una variable “RGB” y se muestra la imagen con la instrucción *imshow*.

- `RGB = xyz2rgb(XYZ_rec)`
- `imshow(RGB)`

41.3 Resultado.

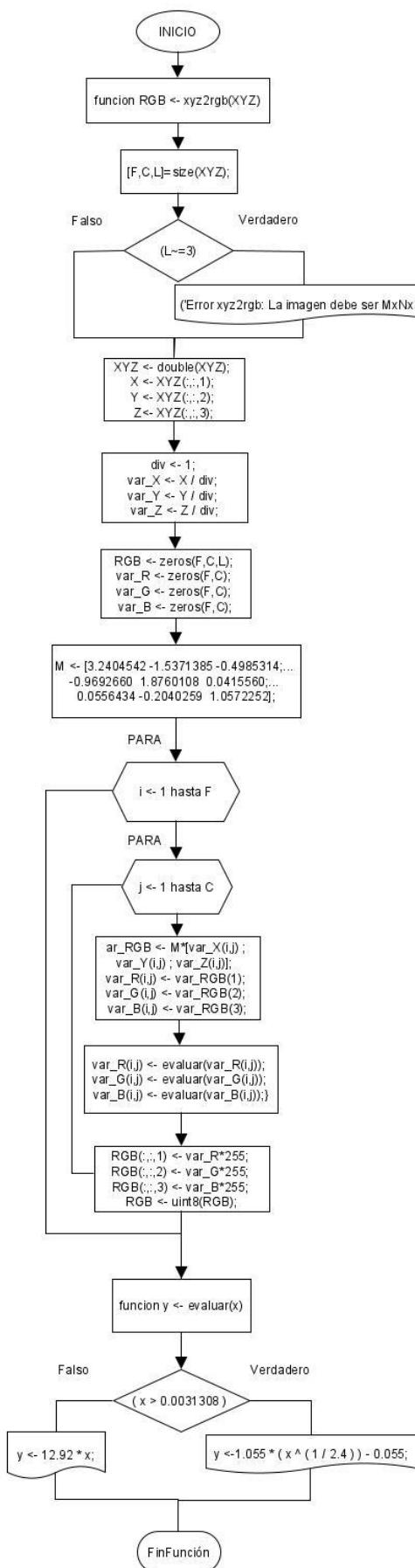


Fig 41.1. Diagrama de flujo de la función.

```

1 //Nombre: xyz2rgb(XYZ)
2 //Parámetros: Donde xyz es la imagen a color
3 //              RGB es la imagen en el modelo RGB.
4 //Devuelve: Imagen a modelo rgb
5
6 Algoritmo xyz2rgb(XYZ)
7 funcion RGB <- xyz2rgb(XYZ)
8     [F,C,L]=size(XYZ);
9
10    Si (L~=3)
11        ('Error xyz2rgb: La imagen debe ser MxNx3')
12    finsi
13
14    XYZ <- double(XYZ);
15    X <- XYZ(:,:,1);
16    Y <- XYZ(:,:,2);
17    Z<- XYZ(:,:,3);
18
19    div <- 1;
20    var_X <- X / div;
21    var_Y <- Y / div;
22    var_Z <- Z / div;
23
24
25    RGB <- zeros(F,C,L);
26    var_R <- zeros(F,C);
27    var_G <- zeros(F,C);
28    var_B <- zeros(F,C);
29
30    M <- [3.2404542 -1.5371385 -0.4985314;...
31           -0.9692660  1.8760108  0.0415560;...
32           0.0556434 -0.2040259  1.0572252];
33
34    Para i <- 1 hasta F
35        para j <- 1 hasta C
36            var_RGB <- M*[var_X(i,j) : var_Y(i,j) : var_Z(i,j)];
37            var_R(i,j) <- var_RGB(1);
38            var_G(i,j) <- var_RGB(2);
39            var_B(i,j) <- var_RGB(3);
40
41            var_R(i,j) <- evaluar(var_R(i,j));
42            var_G(i,j) <- evaluar(var_G(i,j));
43            var_B(i,j) <- evaluar(var_B(i,j));
44
45        FinPara
46    FinPara
47
48    RGB(:,:,1) <- var_R*255;
49    RGB(:,:,2) <- var_G*255;
50    RGB(:,:,3) <- var_B*255;
51    RGB <- uint8(RGB);
52 FinFuncion
53
54 funcion y <- evaluar(x)
55
56     si ( x > 0.0031308 )
57         y <-1.055 * ( x ^ ( 1 / 2.4 ) ) - 0.055;
58
59     SiNo
60         y <- 12.92 * x;
61     FinSi
62
63 FinFuncion
64 FinAlgoritmo

```

Fig.41.2 Función en pseudolenguaje pse.

```


def xyz2rgb(XYZ):
    div = 1

    F,C,L = XYZ.shape
    if (L != 3):
        print ('Error xyz2rgb: La imagen debe ser MxNx3' )

    X = XYZ[...,:,0]
    Y = XYZ[...,:,1]
    Z = XYZ[...,:,2]

    var_X = X / div
    var_Y = Y / div
    var_Z = Z / div

    RGB = np.zeros((F,C, L), dtype=float)
    var_R = np.zeros((F,C), dtype=float)
    var_G = np.zeros((F,C), dtype=float)
    var_B = np.zeros((F,C), dtype=float)

    M = np.array([[3.2404542, -1.5371385, -0.4985314],
                  [-0.9692660,  1.8760108,  0.0415560],
                  [0.0556434, -0.2040259,  1.0572252]])

    for i in range(0, F):
        for j in range(0, C):
            N = np.array([[var_X[i,j]], [var_Y[i,j]], [var_Z[i,j]]])
            var_RGB = np.dot(M, N)

            var_R[i, j] = var_RGB[0][0]
            var_G[i, j] = var_RGB[1][0]
            var_B[i, j] = var_RGB[2][0]

            var_R[i,j] = evaluar2(var_R[i,j])
            var_G[i,j] = evaluar2(var_G[i,j])
            var_B[i,j] = evaluar2(var_B[i,j])

    RGB[...,:,0] = var_R
    RGB[...,:,1] = var_G
    RGB[...,:,2] = var_B

    return RGB

def evaluar2(x):
    if x > 0.0031308:
        y = 1.055 * (x**(1/2.4)) - 0.055
    else:
        y= 12.92*x
    return y

```

Fig.41.3.Algoritmo en Python.

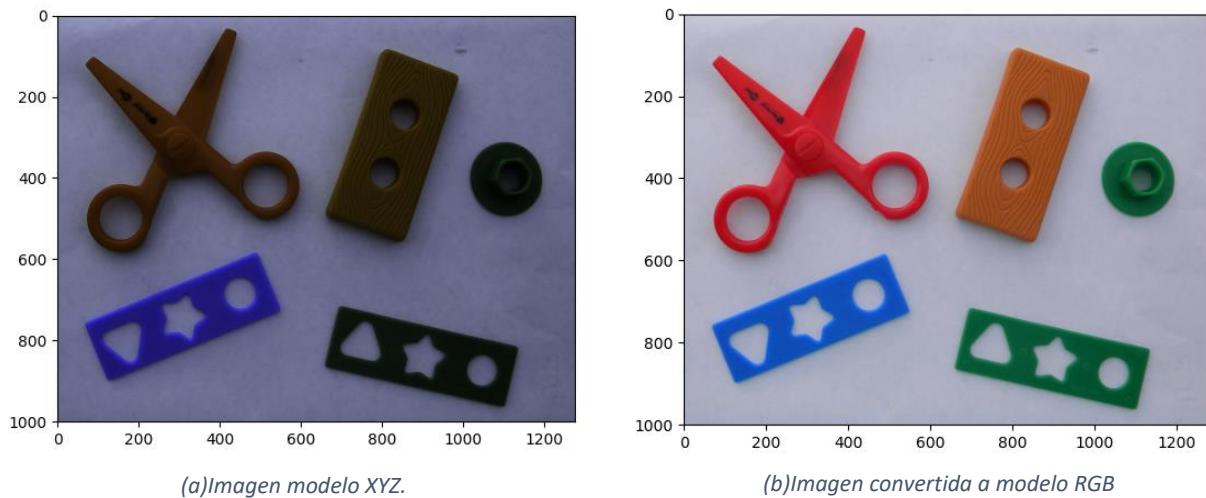


Fig.41.4. Conversión de XYZ a RGB en Python.

42. Función xyz2lab

42.1 Uso.

`xyz2lab` Conversión de una imagen modelo XYZ a modelo de color LAB.

La sintaxis para su uso es:

- `xyz2lab(myXYZ)`

Donde: “myXYZ” es la imagen.

42.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `xyz2lab`.

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”

- `A=imread(herramientas.jpg')`

2. Se cambia la imagen al modelo de color “XYZ” y se almacena en una variable “XYZ”.

- `XYZ= rgb2xyz(A)`

3. Se cambia la imagen al modelo de color “LAB”, se almacena en una variable “lab” y se muestra con la instrucción *imshow*.

- `lab=xyz2lab(XYZ)`
- `imshow(lab)`

42.3 Resultado.

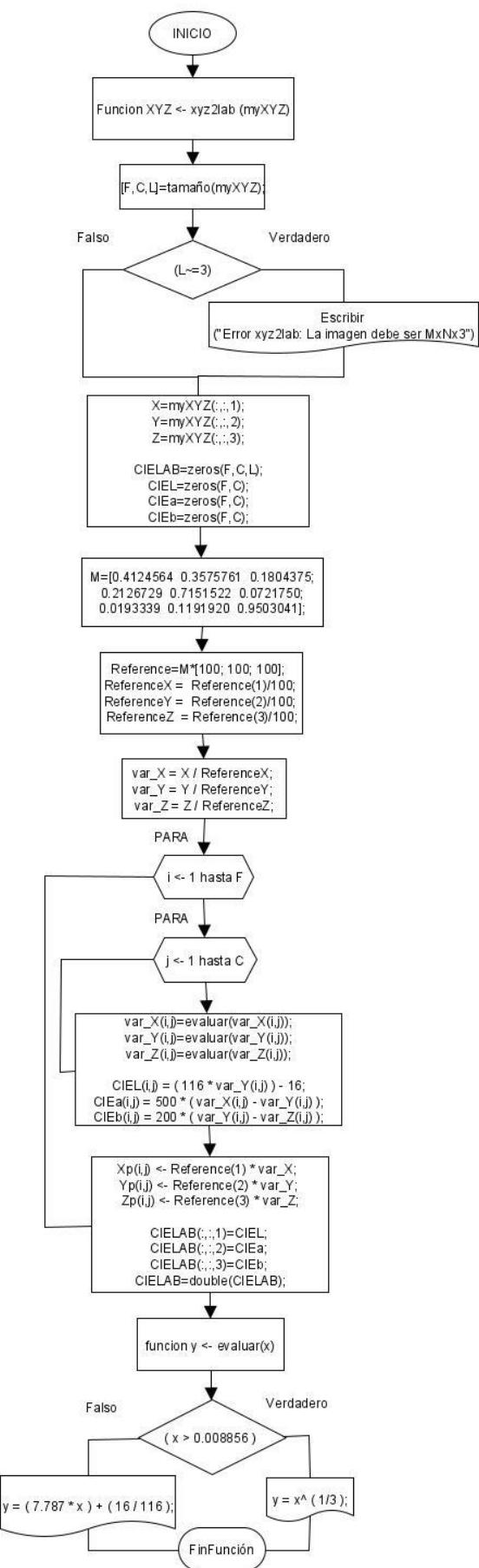


Fig.42.1.Diagrama d flujo de la función.

```

1 //Nombre: xyz2lab
2 //Parámetros:Donde xyz es la imagen en el modelo xyz
3 //           lab es la imagen en el modelo XYZ.
4 //Devuelve la conversión de una imagen del modelo de color LAB al modelo de color XYZ.
5
6 Algoritmo xyz2lab
7 función CIELAB=xyz2lab(myXYZ)
8   [F,C,L]=tamaño(myXYZ);
9
10  Si (L~=3)
11    Escribir ("Error xyz2lab: La imagen debe ser MxNx3")
12  FinSi
13
14  X=myXYZ (:,:,1);
15  Y=myXYZ (:,:,2);
16  Z=myXYZ (:,:,3);
17
18  CIELAB=zeros(F,C,L);
19  CIEL=zeros(F,C);
20  CIEa=zeros(F,C);
21  CIEb=zeros(F,C);
22
23  M=[0.4124564  0.3575761  0.1804375;
24    0.2126729  0.7151522  0.0721750;
25    0.0193339  0.1191920  0.9503041];
26
27  Reference=M*[100; 100; 100];
28  ReferenceX = Reference(1)/100;
29  ReferenceY = Reference(2)/100;
30  ReferenceZ = Reference(3)/100;
31
32  var_X = X / ReferenceX;
33  var_Y = Y / ReferenceY;
34  var_Z = Z / ReferenceZ;
35
36  Para i <- 1 hasta F
37    para j <- 1 hasta C
38      var_X(i,j)=evaluar(var_X(i,j));
39      var_Y(i,j)=evaluar(var_Y(i,j));
40      var_Z(i,j)=evaluar(var_Z(i,j));
41
42      CIEL(i,j) = ( 116 * var_Y(i,j) ) - 16;
43      CIEa(i,j) = 500 * ( var_X(i,j) - var_Y(i,j) );
44      CIEb(i,j) = 200 * ( var_Y(i,j) - var_Z(i,j) );
45
46      Xp(i,j) <- Reference(1) * var_X;
47      Yp(i,j) <- Reference(2) * var_Y;
48      Zp(i,j) <- Reference(3) * var_Z;
49    FinPara
50  FinPara
51
52  CIELAB(:,:,1)=CIEL;
53  CIELAB(:,:,2)=CIEa;
54  CIELAB(:,:,3)=CIEb;
55  CIELAB=double(CIELAB);
56
57  función y <- evaluar(x)
58
59  si ( x > 0.008856 )
60    y = x^ ( 1/3 );
61  SiNo
62    y = ( 7.787 * x ) + ( 16 / 116 );
63  FinSi
64
65  FinFuncion
66
67 FinFuncion
68 FinAlgoritmo
69
70 //NOTA:
71 //double: Conversión a un número real.

```

Fig.42.2 Función en pseudolenguaje pse.

```

def xyz2lab(myXYZ):
    [F,C,L]= myXYZ.shape

    if (L != 3):
        print ('Error xyz2lab: La imagen debe ser MxNx3')

    X = myXYZ[:, :, 0]
    Y = myXYZ[:, :, 1]
    Z = myXYZ[:, :, 2]

    CIELAB = np.zeros((F,C,L), dtype=float)
    CIEL = np.zeros((F,C), dtype=float)
    CIEa = np.zeros((F,C), dtype=float)
    CIEb = np.zeros((F,C), dtype=float)

    #D65
    M= np.array([[0.4124564,  0.3575761,  0.1804375],
                 [0.2126729,  0.7151522,  0.0721750],
                 [0.0193339,  0.1191920,  0.9503041]])
    N = np.array([[100],[100],[100]])
    Reference = np.dot(M, N) #Respecto a 100 en D65
    ReferenceX = Reference[0][0]/100
    ReferenceY = Reference[1][0]/100
    ReferenceZ = Reference[2][0]/100

    var_X = X / ReferenceX
    var_Y = Y / ReferenceY
    var_Z = Z / ReferenceZ

    for i in range(0, F):
        for j in range(0, C):
            var_X[i,j]= evaluar(var_X[i,j])
            var_Y[i,j]= evaluar(var_Y[i,j])
            var_Z[i,j]= evaluar(var_Z[i,j])

            CIEL[i,j] = (116.0 * var_Y[i,j]) - 16.0
            CIEa[i,j] = 500.0 * (var_X[i,j] - var_Y[i,j])
            CIEb[i,j] = 200.0 * (var_Y[i,j] - var_Z[i,j])
    print(16/116)

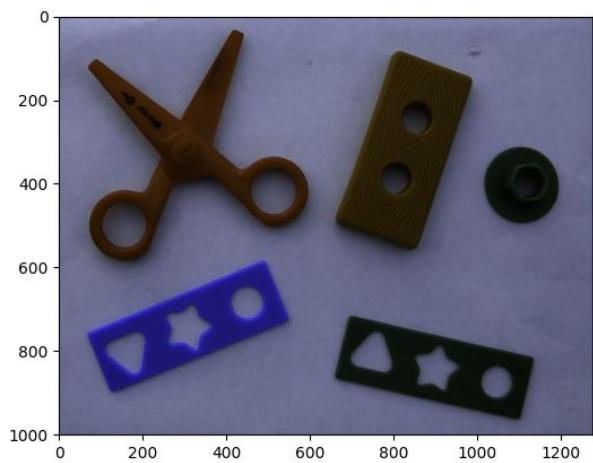
    CIELAB[:, :, 0]= CIEL
    CIELAB[:, :, 1]= CIEa
    CIELAB[:, :, 2]= CIEb
    return CIELAB

def evaluar(x):
    if x > 0.008856:
        y = x**(1/3)
    else:
        y = (7.787*x) + (16.0/116.0)

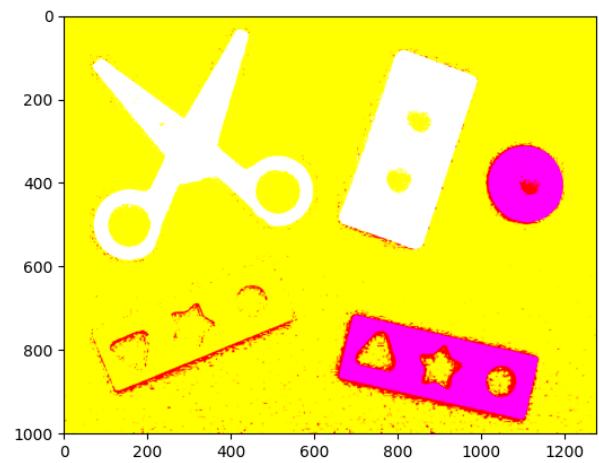
    return y

```

Fig.42.3 Algoritmo en Python.



(a)Imagen modelo XYZ.



(b)Imagen convertida a modelo de color LAB.

Fig.42.4. Conversión de XYZ a LAB en Python.

43. Función lab2xyz.

43.1 Uso.

`lab2xyz` Conversión de una imagen de modelo LAB a modelo de color XYZ.

La sintaxis para su uso es:

- `lab2xyz(lab)`

Donde: “`lab`” es la imagen.

43.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `lab2xyz`.

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”.

- `A = imread('herramientas.jpg')`

2. Se cambia la imagen de modelo “RGB” a modelo de color “XYZ” y se almacena en una variable “XYZ”.

- $\text{XYZ} = \text{rgb2xyz(A)}$

3. Se cambia la imagen a modelo de color “LAB”, se almacena en una variable “lab” y se muestra la imagen con la instrucción *imshow*

- $\text{lab} = \text{xyz2lab(XYZ)}$
- imshow(lab)

4. Se cambia la imagen a modelo de color “XYZ”, se almacena en una variable “XYZ1” y se muestra la imagen con la instrucción *imshow*.

- $\text{XYZ1} = \text{lab2xyz(lab)}$
- imshow(XYZ1)

43.3 Resultado:

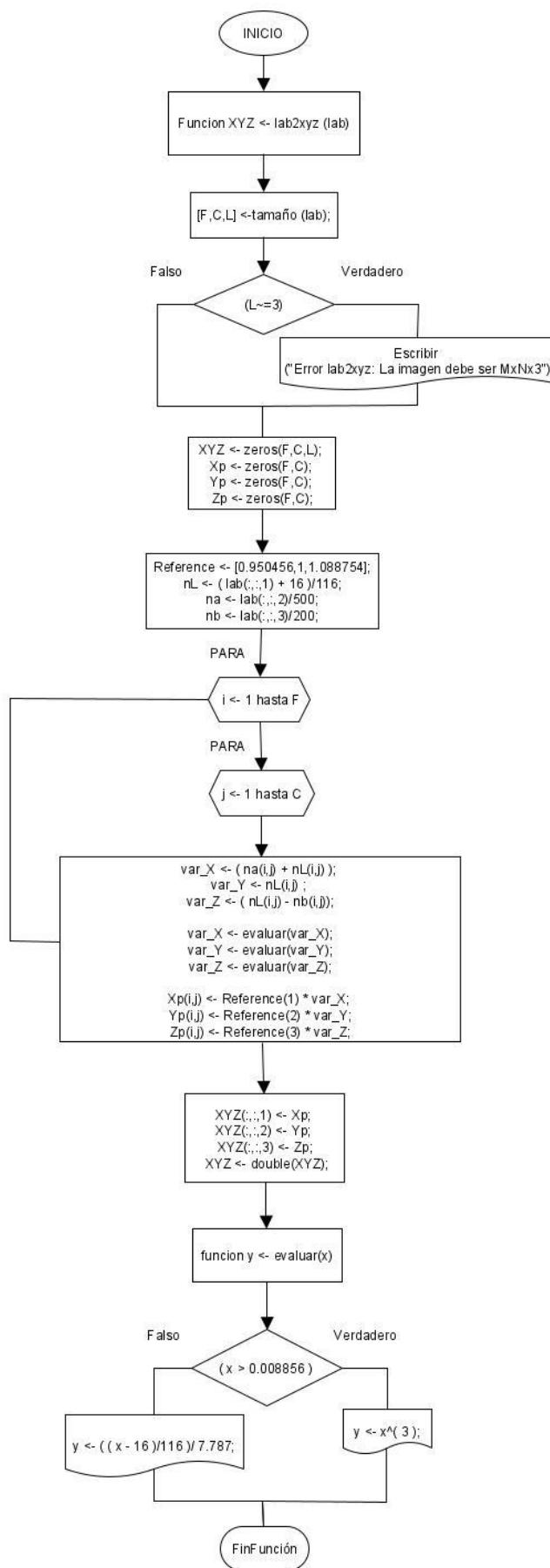


Fig.43.1. Diagrama de flujo de la función.

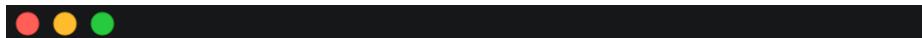
```

1 //Nombre: lab2xyz
2 //Parámetros:Donde lab es la imagen en el modelo lab
3 //           xyz es la imagen en el modelo XYZ.
4 //Devuelve la conversión de una imagen del modelo de color LAB al modelo de color XYZ.
5
6 Algoritmo lab2xyz
7 Funcion XYZ <- lab2xyz (lab)
8     [F,C,L] <-tamaño (lab);
9
10    Si (L~=3)
11        Escribir ("Error lab2xyz: La imagen debe ser MxNx3")
12    FinSi
13
14    XYZ <- zeros(F,C,L);
15    Xp <- zeros(F,C);
16    Yp <- zeros(F,C);
17    Zp <- zeros(F,C);
18
19    Reference <- [0.950456,1,1.088754]; //referencia d65 a 2 grados
20    nL <- ( lab(:,:,1) + 16 )/116;
21    na <- lab(:,:,2)/500;
22    nb <- lab(:,:,3)/200;
23
24
25    Para i <- 1 hasta F
26        para j <- 1 hasta C
27            var_X <- ( na(i,j) + nL(i,j) );
28            var_Y <- nL(i,j) ;
29            var_Z <- ( nL(i,j) - nb(i,j));
30
31            var_X <- evaluar(var_X);
32            var_Y <- evaluar(var_Y);
33            var_Z <- evaluar(var_Z);
34
35            Xp(i,j) <- Reference(1) * var_X;
36            Yp(i,j) <- Reference(2) * var_Y;
37            Zp(i,j) <- Reference(3) * var_Z;
38        FinPara
39    FinPara
40
41    XYZ(:,:,1) <- Xp;
42    XYZ(:,:,2) <- Yp;
43    XYZ(:,:,3) <- Zp;
44    XYZ <- double(XYZ);
45
46    funcion y <- evaluar(x)
47
48        si ( x > 0.008856 )
49            y <- x^( 3 );
50        SiNo
51            y <- ( ( x - 16 )/116 )/ 7.787;
52        FinSi
53
54    FinFuncion
55 FinFuncion
56 FinAlgoritmo
57
58 //NOTA:
59 //double: Conversión a un número real.

```

Fig.43.2.Funcióen en pseudolenguaje pse.

```


def lab2xyz(lab):
    F,C,L = lab.shape
    if (L != 3):
        print ('Error lab2xyz: La imagen debe ser MxNx3')

    XYZ = np.zeros((F,C,L), dtype=float)
    Xp = np.zeros((F,C), dtype=float)
    Yp = np.zeros((F,C), dtype=float)
    Zp = np.zeros((F,C), dtype=float)

    Reference = [0.950456,1,1.088754]
    nL = (lab[...,0] + 16)/116
    na = lab[...,1]/500
    nb = lab[...,2]/200

    for i in range(0, F):
        for j in range(0, C):
            var_X = (na[i,j] + nL[i,j])
            var_Y = nL[i,j]
            var_Z = (nL[i,j] - nb[i,j])

            var_X=evaluar3(var_X);
            var_Y=evaluar3(var_Y);
            var_Z=evaluar3(var_Z);

            Xp[i,j] = Reference[0] * var_X
            Yp[i,j] = Reference[1] * var_Y
            Zp[i,j] = Reference[2] * var_Z

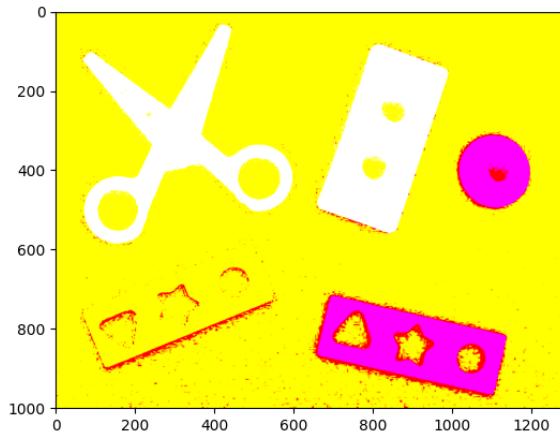
    XYZ[...,0] = Xp
    XYZ[...,1] = Yp
    XYZ[...,2] = Zp

    return XYZ

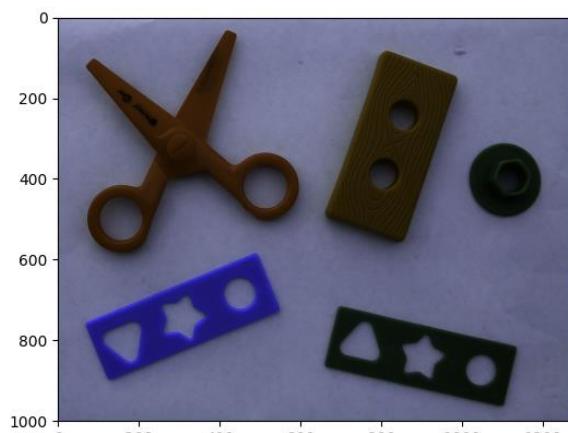
def evaluar3(x):
    if x > 0.008856:
        y = x**3
    else:
        y = ((x-16)/116)/7.787
    return y

```

Fig.43.3 Algoritmo en Python.



(a)Imagen modelo LAB.



(b)Imagen convertida a modelo color XYZ.

Fig.43.4 Conversión de LAB a XYZ en Python.

44. Función `rgb2lab`.

44.1 Uso.

`rgb2lab` Conversión de una imagen de modelo RGB a modelo de color LAB.

La sintaxis para su uso es:

- `rgb2lab(RGB)`

Donde: “RGB” es la imagen.

44.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función `rgb2lab`.

1. Se lee una imagen llamada “herramientas.jpg” y se almacena en una variable “A”.

- `A=imread('herramientas.jpg')`

2. Se cambia la imagen al modelo “LAB”, se almacena en la variable “lab” y se muestra con la instrucción `imshow`.

- `lab=rgb2lab(A)`
- `imshow(lab)`

44.3 Resultado:

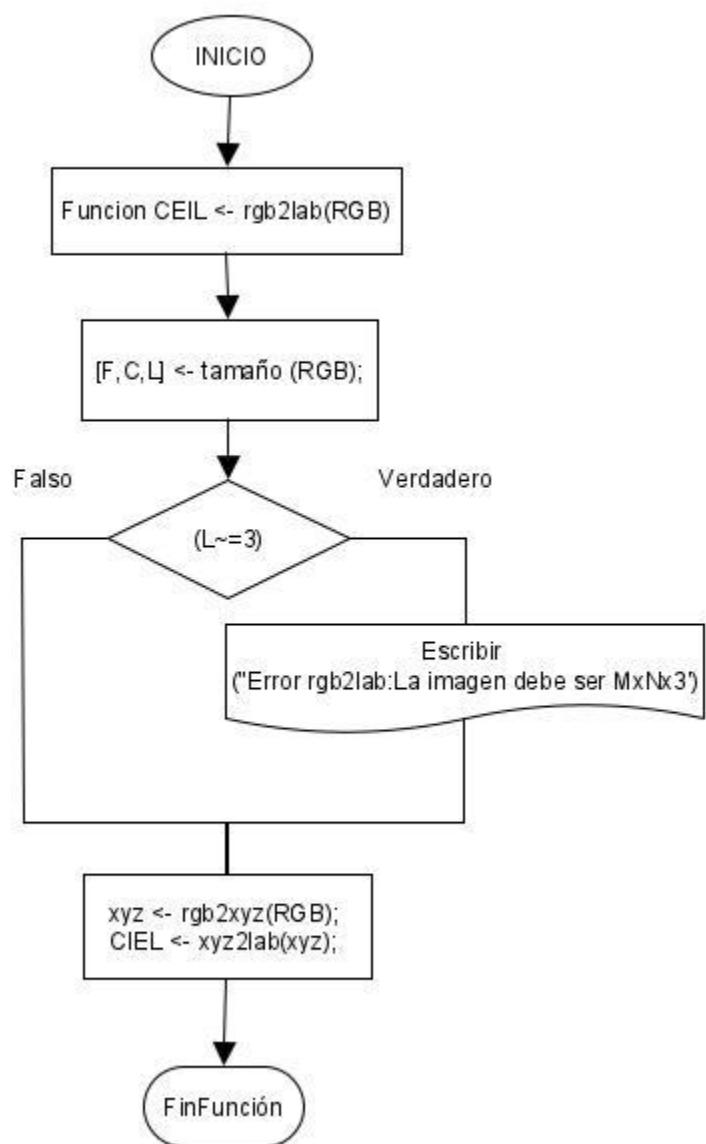


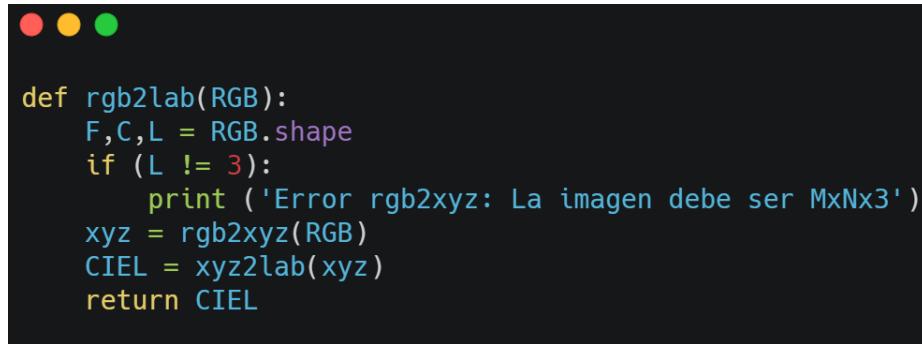
Fig.44.1. Diagrama de flujo de la función.

```

1 //Nombre: rgb2lab
2 //Parámetros:Donde RGB es la imagen en el modelo RGB
3 //           CEIL es la imagen en el modelo LAB, cada capa de la imagen resultante pertenece
4 //           a una capa del modelo L=lab(:,:,1), a=lab(:,:,2) y L=lab(:,:,3).
5 //Devuelve conversión de el valor de blanco RGB a L*a*b.
6
7 Algoritmo rgb2lab
8 Funcion CEIL <- rgb2lab(RGB)
9     [F,C,L] <- tamaño (RGB);
10
11    Si (L~=3)
12        Escribir ("Error rgb2lab:La imagen debe ser MxNx3")
13    FinSi
14
15    xyz <- rgb2xyz(RGB);
16    CIEL <- xyz2lab(xyz);
17
18 FinFuncion
19 FinAlgoritmo
20
21 //NOTA:
22 //CEIL: Redondea hacia el infinito positivo

```

Fig.44.2 Función en pseudolenguaje pse.

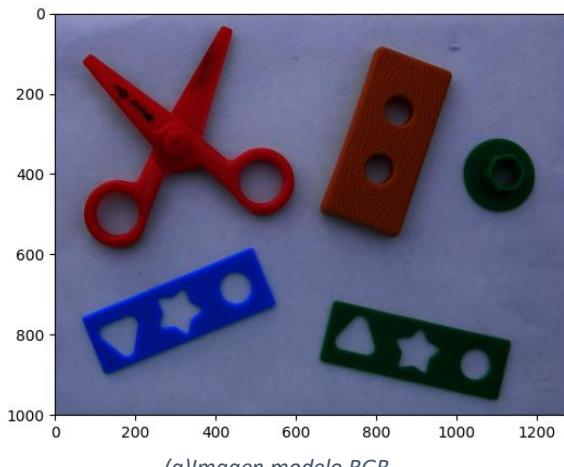


```

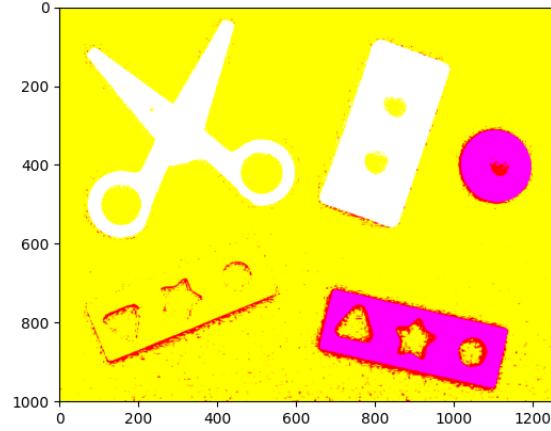
def rgb2lab(RGB):
    F,C,L = RGB.shape
    if (L != 3):
        print ('Error rgb2xyz: La imagen debe ser MxNx3')
    xyz = rgb2xyz(RGB)
    CIEL = xyz2lab(xyz)
    return CIEL

```

Fig.44.3. Algoritmo en Python.



(a)Imagen modelo RGB



(b)Imagen convertida a modelo de color LAB.

*Fig.44.4 Conversión RGB a LAB en Python.*¹⁹

45. Función lab2rgb.

45.1 Uso.

lab2rgb Conversión de una imagen de modelo LAB a modelo de color RGB.

La sintaxis para su uso es:

- lab2rgb(LAB)

Donde: "LAB" es la imagen.

45.2 Ejemplo:

El siguiente ejemplo muestra los pasos a seguir para el uso de la función *lab2rgb*.

1. Se lee la imagen llamada "herramientas.jpg" y se almacena en una variable "A"
 - A=imread('herramientas'.jpg)
2. Se cambia la imagen al modelo "LAB", se almacena en una variable "lab" y se muestra con la instrucción *imshow*.
 - lab=rgb2lab(A)
 - imshow(lab)
3. Se cambia la imagen al modelo "RGB", se almacena en una variable "rgb" y se muestra con la instrucción *imshow*.
 - rgb = lab2rgb(lab)
 - imshow(rgb)

45.3 Resultado:

¹⁹ Imagen (a), tomada de (Toolbox básico para el procesamiento de imágenes, 2017)

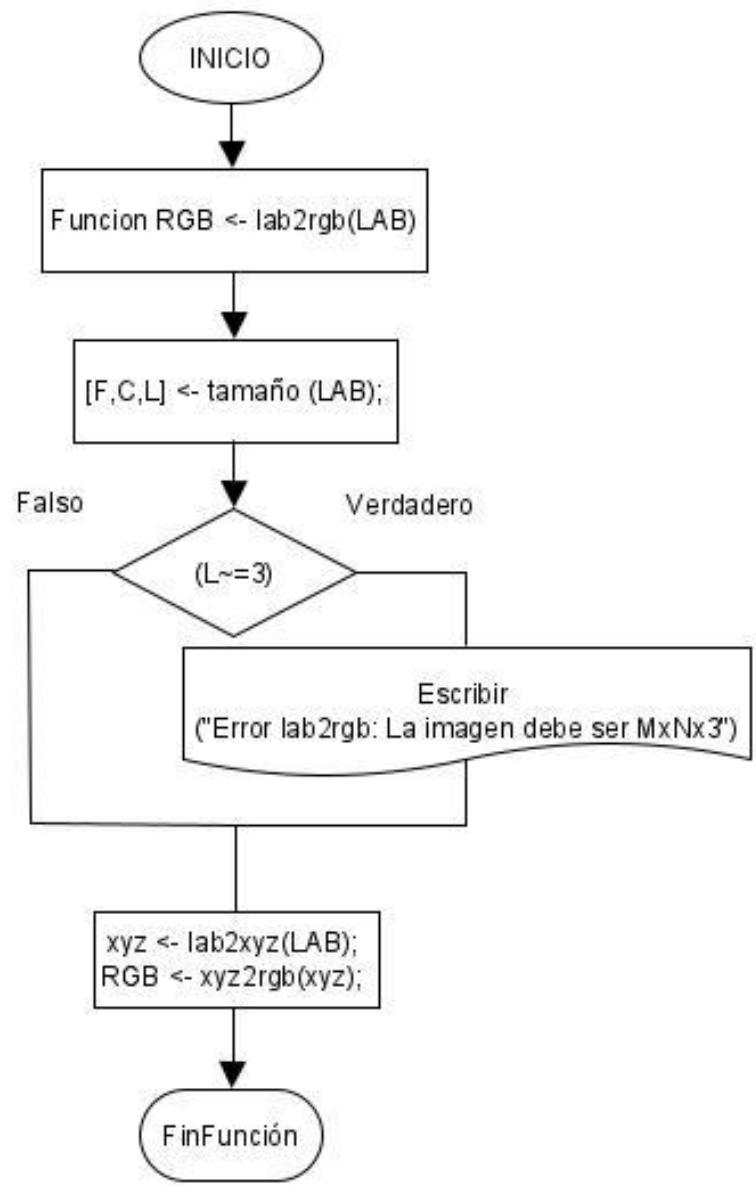


Fig.45.1 Diagrama de flujo de la Función.

```

1 //Nombre: lab2rgb
2 //Parámetros: Donde LAB es la imagen en el modelo LAB
3 //          RGB es la imagen en el modelo RGB.
4 //Devuelve la conversión de una imagen del modelo de color LAB al modelo de color RGB.
5
6 Algoritmo lab2rgb
7 Función RGB <- lab2rgb(LAB)
8     [F,C,L] <- tamaño (LAB);
9
10    Si (L~≡3)
11        Escribir ("Error lab2rgb: La imagen debe ser MxNx3")
12    FinSi
13
14    xyz <- lab2xyz(LAB);
15    RGB <- xyz2rgb(xyz);
16
17 FinFunción
18 FinAlgoritmo

```

Fig.45.2. Función en pseudolenguaje pse.

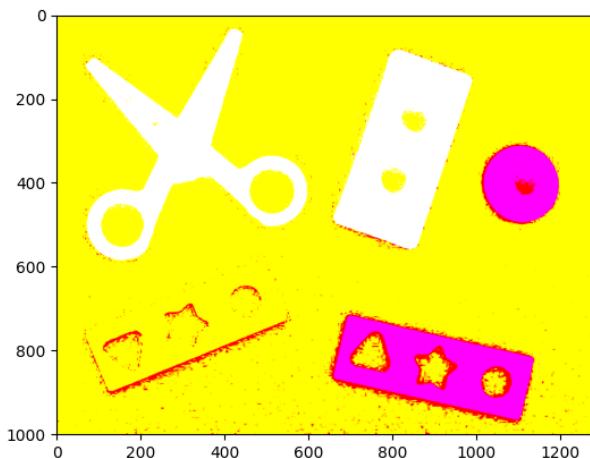


```

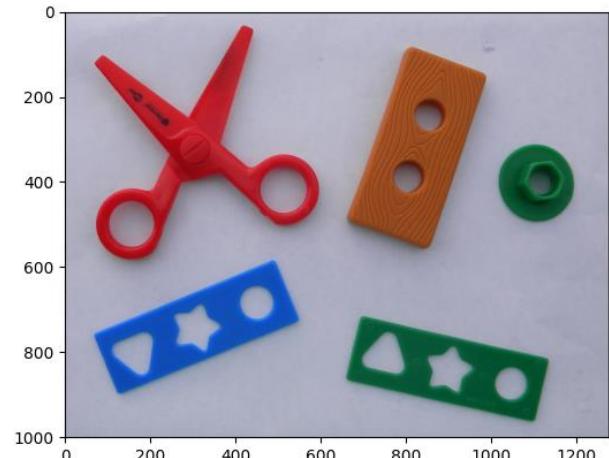
def lab2rgb(LAB):
    F,C,L = LAB.shape
    if (L != 3):
        print ('Error rgb2xyz: La imagen debe ser MxNx3' )
    xyz = lab2xyz(LAB)
    RGB = xyz2rgb(xyz)
    return RGB

```

Fig.45.3.Algoritmo en Python.



(a)Imagen modelo LAB.



(b)Imagen convertida a modelo RGB.

Fig.45.4 Conversión de LAB a RGB en Python.

BIBLIOGRAFÍA.

- [1] MathWorks - Makers of MATLAB and Simulink.
<https://es.mathworks.com/products/matlab.html>
- [2] *Libro guía 2017-Dic06 Procesamiento de imágenes, Toolbox básico. UTP*
- [3] González, R. C., & Woods, R. E. (1996). *Tratamiento digital de imágenes* (Vol. 3). New York: Addison-Wesley.
- [4] <https://aprendeconalf.es/docencia/python/manual/matplotlib/>
- [5] <https://likegeeks.com/es/tutorial-matplotlib/>
- [6] <https://es.mathworks.com/solutions/image-video-processing.html>
- [7] Burger, W., & Burge, M. J. (2016). *Digital image processing: an algorithmic introduction using Java*. Springer.
- [8] <https://www.freecodecamp.org/espanol/news/operadores-basicos-en-python-con-ejemplos/>
- [9] http://do1.dr-chuck.com/pythonlearn/ES_es/pythonlearn.pdf Python para todos
Explorando la información con Python 3/Charles R. Severance.
- [10] <https://www.editorialeidec.com/wp-content/uploads/2020/10/Algoritmos-resueltos-con-Python.pdf> Algoritmos resueltos con Python/Edgardo Condor.

REFERENCIAS.

1. (Toolbox básico para el procesamiento de imágenes, 2017)
- 2-(Desconecta, 2015)
- 3.(Google imágenes, 2021)

