Catálogo de Algoritmos #2

Integrantes:

- Gabriel Brenes Vega - Carnet 2015127420
- Kenneth Hernández Salazar - Carnet 2017102682
- Isaac Porras Pérez - Carnet 2017107550

1. Semana 7: Vecindarios y Convolución

1.1. Convolución de una matriz

Código 1: Código para el método de convolución en 1 dimensión.

```
import numpy as np

def MatrixConvolution1D():
    a=np.array([1,3,5,7,6])
    b=np.array([1,2,3])
    m = len(a)
    n = len(b)
    convolution= np.zeros(m+n-1)
    for i in range(1,(m+n-1)+1):#Starts in 1 because that is the way the method
                                #works, so, is necesary to adjust the indexes
        for j in range(max(1,i+1-n),min(i,m)+1): #S={max(1,i+1-n),....,min(i,n)}
            convolution[i-1]=convolution[i-1]+a[j-1]*b[(i-1)-j+1]
    print("Convolution 1D using step by step")
    print(convolution)
    conv = np.convolve(a,b)
    print("Convolution 1D using numpy method")
    print(conv)
MatrixConvolution1D()
```

Código 2: Código para el método de convolución en 2 dimensiones.

```
import numpy as np
import scipy as scp
from scipy import signal

def MatrixConvolution2D():
    a=[[1,0,1],[4,3,1],[-1,0,2],[3,0,-7]]
    b=[[1,-1,2,3],[-4,0,1,5],[3,2,-1,0]]
    m1,n1=np.shape(a)
    m2,n2=np.shape(b)
    convolution= np.zeros((m1+m2-1,n1+n2-1))
    for j in range(1,(m1+m2-1)+1):#Starts in 1 because that is the way the method
                                #works, so, is necesary to adjust the indexes
        for k in range(1,(n1+n2-1)+1):#Starts in 1 because that is the way the
                                      #method works, so, is necesary to adjust
                                      #the indexes
            for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),..,min(j,m1)
```

```
            for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),...
                                                      #.,min(k,n1)
                convolution[j-1][k-1]=convolution[j-1][k-1]+
                                    (a[p-1][q-1])*(b[(j-1)-(p)+1][(k-1)-(q)+1])
    print("Convolution 2D using step by step")
    print(convolution)
    conv= signal.convolve2d(a,b,mode='full',boundary='fill',fillvalue=0)
    print("Convolution 2D using scipy method")
    print(conv)

MatrixConvolution2D()
```

2. Semana 8: Filtros en el Dominio Espacial

2.1. Filtro Promedio

Código 3: Código para el método del filtro promedio.

```
clc;
clear;
close all;

pkg load image

A = imread('child.jpg');
subplot(1,2,1);
imshow(A);
title('Original Image')

%Create the average filter mask
B=(1/9)*ones(3,3);
A=im2double(A);
C=conv2(A,B,'same');
C=im2uint8(C);
subplot(1,2,2)
imshow(C);
title('Average Filter')
```

2.2. Filtro Gaussiano

Código 4: Código para el método del filtro Gaussiano.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

def Convolution(a,b,type_convolution):
    m1,n1=np.shape(a)
    m2,n2=np.shape(b)
    convolution= np.zeros((m1+m2-1,n1+n2-1))
    if type_convolution=="full":
        for j in range(1,(m1+m2-1)+1):#Starts in 1 because that is the way the method
                                        #works, so, is necesary to adjust the indexes
            for k in range(1,(n1+n2-1)+1):#Starts in 1 because that is the way the
                                        #method works, so, is necesary to adjust the
                                        #indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),....,
                                                            #min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                                #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                                                *(b[(j-1)-(p)+1][(k-1)-(q)+1])
    else:
        for j in range(1,m1+1):#Starts in 1 because that is the way the method works,
                                #so, is necesary to adjust the indexes
            for k in range(1,n1+1):#Starts in 1 because that is the way the method
                                    #works, so, is necesary to adjust the indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),...
                                                        #.,min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                            #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                        *(b[(j-1)-(p)+1][(k-1)-(q)+1])
    return convolution

def GaussianFilter():
    image=plt.imread('child2.jpg')
    image=image.astype(float)
    image=np.asarray(image)
    image=np.clip(image, 0, 255)
    filterG=(1/16)*np.array([[1, 2, 1],[2, 4, 2],[1, 2, 1]])
    fig = plt.figure()
    image=image.astype(np.uint8)#image at the moment is
    #an array of type flot and here it becomes type uint8
    convolution=Convolution(image,filterG,"same")
    convolution=np.clip(convolution, 0, 255)
    convolution=convolution.astype(np.uint8)#image at the moment is
    #an array of type flot and here it becomes type uint8
    convolution=np.asarray(convolution)
    ax1 = fig.add_subplot(1,2,1)
    ax2 = fig.add_subplot(1,2,2)
    ax1.set_title("Original Image")
    ax2.set_title("Image with Gaussian Filter")
```

```
    ax1.imshow(image,cmap='gray')
    ax2.imshow(convolution,cmap='gray',vmin=0,vmax=255)
GaussianFilter()
```

2.2.1. Filtro Laplaciano

Código 5: Código para el método del filtro Laplaciano.

```
clc;
clear;
close all;

pkg load image

A = imread('child2.jpg');
subplot(1,2,1);
imshow(A);
title('Original Image')

%Laplacian filter mask
B=[1 1 1;1 -8 1;1 1 1];
A=im2double(A);
%C=conv2(A,B,'same');
C=Convolution(A,B,'same');
subplot(1,2,2)
imshow(C);
title('Laplacian Filter')
```

2.2.2. Filtro de Sobel

Código 6: Código para el método del filtro Sobel.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

def Convolution(a,b,type_convolution):
    m1,n1=np.shape(a)
    m2,n2=np.shape(b)
    convolution= np.zeros((m1+m2-1,n1+n2-1))
    if type_convolution=="full":
        for j in range(1,(m1+m2-1)+1):#Starts in 1 because that is the way the method
                                       #works, so, is necesary to adjust the indexes
            for k in range(1,(n1+n2-1)+1):#Starts in 1 because that is the way the
                                       #method works, so, is necesary to adjust the
                                       #indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),....,
                                                          #min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                              #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                                              *(b[(j-1)-(p)+1][(k-1)-(q)+1])
```

```
    else:
        for j in range(1,m1+1):#Starts in 1 because that is the way the method works,
                                #so, is necesary to adjust the indexes
            for k in range(1,n1+1):#Starts in 1 because that is the way the method
                                    #works, so, is necesary to adjust the indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),...
                                                           #.,min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                               #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                        *(b[(j-1)-(p)+1][(k-1)-(q)+1])
    return convolution

def SobelFilter():
    image=plt.imread('baby_yoda.jpg')
    image=image.astype(float)
    image=np.asarray(image)
    image=np.clip(image, 0, 255)
    #filterS=np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])
    filterBx=np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])
    filterBy=np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
    fig = plt.figure()
    image=image.astype(np.uint8)#image at the moment is
    #an array of type flot and here it becomes type uint8
    Cx=Convolution(image,filterBx,"same")
    Cy=Convolution(image,filterBy,"same")
    Cx=np.clip(Cx, 0, 255)
    #an array of type flot and here it becomes type uint8
    Cy=np.clip(Cy, 0, 255)
    #an array of type flot and here it becomes type uint8
    C=np.sqrt(Cx**2+Cy**2)
    C=np.asarray(C)
    C=C.astype(np.uint8)
    #an array of type flot and here it becomes type uint8
    ax1 = fig.add_subplot(1,2,1)
    ax2 = fig.add_subplot(1,2,2)
    ax1.set_title("Original Image")
    ax2.set_title("Image with Sobel Filter")
    ax1.imshow(image,cmap='gray')
    ax2.imshow(C,cmap='gray',vmin=0,vmax=255)
SobelFilter()
```

2.3. Enfatizar Bordes

Código 7: Código para el método de enfatizar bordes.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

def Convolution(a,b,type_convolution):
    m1,n1=np.shape(a)
    m2,n2=np.shape(b)
    convolution= np.zeros((m1+m2-1,n1+n2-1))
    if type_convolution=="full":
        for j in range(1,(m1+m2-1)+1):#Starts in 1 because that is the way the method
                                        #works, so, is necesary to adjust the indexes
            for k in range(1,(n1+n2-1)+1):#Starts in 1 because that is the way the
                                        #method works, so, is necesary to adjust the
                                        #indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),....,
                                                            #min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                                #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                                                *(b[(j-1)-(p)+1][(k-1)-(q)+1])
    else:
        for j in range(1,m1+1):#Starts in 1 because that is the way the method works,
                                #so, is necesary to adjust the indexes
            for k in range(1,n1+1):#Starts in 1 because that is the way the method
                                    #works, so, is necesary to adjust the indexes
                for p in range(max(1,j-m2+1),min(j,m1)+1):#S={max(1,j-m2+1),...
                                                        #.,min(j,m1)
                    for q in range(max(1,k-n2+1),min(k,n1)+1):#S={max(1,k-n2+1),..
                                                            #..,min(k,n1)
                        convolution[j-1][k-1]=convolution[j-1][k-1]+(a[p-1][q-1])
                        *(b[(j-1)-(p)+1][(k-1)-(q)+1])
    return convolution

def EmphasizeEdges():
    image=plt.imread('baby_yoda.jpg')
    image=image.astype(float)
    image=np.asarray(image)
    image=np.clip(image, 0, 255)
    filterEE=np.array([[1, 1, 1],[1, -8, 1],[1, 1, 1]])
    fig = plt.figure()
    image=image.astype(np.uint8)#image at the moment is
    #an array of type flot and here it becomes type uint8
    convolution=Convolution(image,filterEE,"same")
    c=1
    D=image+c*convolution;
    D=np.asarray(D)
    D=D.astype(np.uint8)#image at the moment is
    #an array of type flot and here it becomes type uint8
    ax1 = fig.add_subplot(1,2,1)
    ax2 = fig.add_subplot(1,2,2)
    ax1.set_title("Original Image")
```

```
    ax2.set_title("Emphasize Edges")
    ax1.imshow(image,cmap='gray')
    ax2.imshow(D,cmap='gray')
EmphasizeEdges()
```

3. Semana 9: Filtros en el Dominio de la Frecuencia

3.1. DFT-2D

Código 8: Código para el método DFT2D (Transformada Rápida de Fourier en 2 Dimensiones).

```
import numpy as np
import matplotlib.pyplot as plt

def im2double(image):
    number_decimals=7
    info=np.iinfo(image.dtype)
    out=image.astype(float) / info.max
    out=out.round(number_decimals)
    return out

def DFT2D():
    image=plt.imread('chest.jpg')
    image_copy=image
    image=np.asarray(image)
    image=im2double(image)
    DFT2D=np.fft.fft2(image)
    DFT2D_Shift=np.fft.fftshift(DFT2D)
    result_DFT2D=np.log(1+np.abs(DFT2D))
    result_DFT2DShift=np.log(1+np.abs(DFT2D_Shift))
    fig = plt.figure()
    ax1 = fig.add_subplot(1,5,1)
    ax2 = fig.add_subplot(1,5,3)
    ax3 = fig.add_subplot(1,5,5)
    ax1.set_title("Original Image")
    ax2.set_title("DFT-2D")
    ax3.set_title("DFT-2D Shift")
    ax1.imshow(image_copy,cmap='gray')
    ax2.imshow(result_DFT2D,cmap='gray',vmin=0,vmax=10)
    ax3.imshow(result_DFT2DShift,cmap='gray',vmin=0,vmax=10)
DFT2D()
```

3.2. Filtro Ideal (Paso-Bajo)

Código 9: Código para el método del filtro ideal paso bajo.

```
clear;
clc;
close all;

pkg load image;

A=imread('edificio_china.jpg');
subplot(2,2,1);
imshow(A);
title('Original Image');

%Calculation of DFT-2D

A=im2double(A);
F=fft2(A);
F_A=fftshift(F);
subplot(2,2,2);
imshow(log(1+abs(F_A)),[]);
title('Image DFT-2D (Shift)');

%Apply the filter using the convolution theorem

%Calculate the filter mask
[m,n]=size(A);
D=zeros(m,n);
for u=1:m
  for v=1:n
    D(u,v)=sqrt(u^2+v^2);
  endfor
endfor


H = zeros(m,n); D0 = 100; ind = (D<=D0); H(ind) = 1;

HSI = H(1:floor(m/2), 1:floor(n/2));

HSD = imrotate(HSI, 90)'; % Upper Right
HID = imrotate(HSI, 180); % Upper Right
HII = imrotate(HSI, 270)'; % Lower Left

[m1, n1] = size(HSI);
H(1:m1, n-n1+1:n) = HSD;
H(m-m1+1:m, n-n1+1:n) = HID;
H(m-m1+1:m, 1:n1) = HII;

% Apply the filter
H_shift = fftshift(H);
DFT2_filt = F.*H;
FM_shift = fftshift(DFT2_filt);
subplot(2,2,3)
imshow(log(1+abs(FM_shift)), [])
```

```
title('Image DFT—2D with Ideal Filter')


%Image Filtered
I_new = abs(ifft2(FM_shift));
subplot(2,2,4)
imshow(I_new)
title('Image with Ideal Filter');
```

3.3. Filtro Gaussiano (Paso-Bajo)

Código 10: Código para el método del filtro Gaussiano paso bajo (con Transformada Rápida de Fourier en 2 Dimensiones).

```
import numpy as np
import matplotlib.pyplot as plt

def im2double(image):
    number_decimals=7
    info=np.iinfo(image.dtype)
    out=image.astype(float) / info.max
    out=out.round(number_decimals)
    return out

def GaussianFilterDFT2D():
    image=plt.imread('edificio_china.jpg')
    image_copy=image
    m,n=np.shape(image)

    image = im2double(image)

    #Calculation of DFT-2D

    F = np.fft.fft2(image)
    F_shift = np.fft.fftshift(F)

    #Calculate the Gaussian Filter
    D = np.zeros((m,n))
    for i in range(1,m+1):
        for j in range(1,n+1):
            D[i-1,j-1] = np.sqrt(i**2+j**2)

    H = np.zeros((m,n))
    sigma = 50
    for i in range(1,m+1):
        for j in range(1,n+1):
            H[i-1,j-1] = np.exp(-(D[i-1][j-1])**2 / (2 * sigma**2))

    index1=int(np.floor(m/2))
    index2=int(np.floor(n/2))

    HSI = H[:index1, :index2]

    HSD = np.transpose(np.rot90(HSI,1))
```

```
    HID = np.rot90(HSI,2)
    HII = np.transpose(np.rot90(HSI,3))

    m1, n1 = np.shape(HSI)
    H[0:m1, n-n1:n] = HSD
    H[m-m1:m+1, n-n1:n] = HID
    H[m-m1:m+1, 0:n1] = HII

    H = np.asarray(H)
    F = np.asarray(F)

    #Apply the filter
    H_shift = np.fft.fftshift(H)
    DFT2_filt = F*H
    FM_shift =  np.fft.fftshift(DFT2_filt)

    #Filtered Image
    I_new = np.abs(np.fft.ifft2(DFT2_filt));

    result_F=np.log(1+np.abs(F_shift))
    result_FM=np.log(1+np.abs(FM_shift))

    fig = plt.figure()
    ax1 = fig.add_subplot(3,3,1)
    ax2 = fig.add_subplot(3,3,3)
    ax3 = fig.add_subplot(3,3,7)
    ax4 = fig.add_subplot(3,3,9)
    ax1.set_title("Original Image")
    ax2.set_title("Image DFT-2D (Shift)")
    ax3.set_title("Image DFT-2D with Gaussian Filter")
    ax4.set_title("Image with Gaussian Filter")
    ax1.imshow(image_copy,cmap='gray')
    ax2.imshow(result_F,cmap='gray',vmin=0,vmax=10)
    ax3.imshow(result_FM,cmap='gray',vmin=0,vmax=10)
    ax4.imshow(I_new,cmap='gray')
GaussianFilterDFT2D()
```

3.4. Filtro Butterworth (Paso-Bajo)

Código 11: Código para el método del filtro Butterworth paso bajo.

```
clc;
clear;
close all;

pkg load image

A = imread('edificio_china.jpg');

[m,n] = size(A);

subplot(2,2,1)
imshow(A)
title('Original Image')

A = im2double(A);

%Calculation of DFT-2D
F = fft2(A);
F_shift = fftshift(F);
subplot(2,2,2)
imshow(log(1+abs(F_shift)), [])
title('Image DFT-2D (Shift)')

dist = zeros(m,n);
for i = 1:m
  for j = 1:n
    dist(i,j) = sqrt(i^2+j^2);
  endfor
endfor

H = zeros(m,n); D0 = 50; orden = 2;
H = 1 ./(1+(D0./dist).^(-2*orden));

HSI = H(1:floor(m/2), 1:floor(n/2));

HSD = imrotate(HSI, 90)';
HID = imrotate(HSI, 180);
HII = imrotate(HSI, 270)';

[m1, n1] = size(HSI);
H(1:m1, n-n1+1:n) = HSD;
H(m-m1+1:m, n-n1+1:n) = HID;
H(m-m1+1:m, 1:n1) = HII;

% Apply the filter
H_shift = fftshift(H);
DFT2_filt = F.*H;
FM_shift =  fftshift(DFT2_filt);
subplot(2,2,3)
imshow(log(1+abs(FM_shift)), [])
title('Image DFT-2D with Butterworth Filter')
```

```
%Image Filtered
I_new = abs(ifft2(DFT2_filt));
subplot(2,2,4)
imshow(I_new)
title('Image with Butterworth Filter');
```

3.5. Filtro Ideal (Paso-Alto)

Código 12: Código para el método del Filtro Ideal (Paso-Alto).

```
import imageio
import matplotlib.pyplot as plt
import numpy as np


def idealHighPassFilter(I):
    """
    High pass filter using the fourier transform
    :param I: Image to filter
    :return: Plot of two images (original and filtered)
    """
    # Get the size of the image
    M = I.shape[0]
    N = I.shape[1]
    # Get the Fourier Transform of the image
    fourierTransform = np.fft.fftshift(np.fft.fft2(I[:,:,1]))
    # Asign the cut-off frequency
    D0 = 1
    # Get Euclidean Distance
    D = np.zeros([M, N])
    for u in range(M):
        for v in range(N):
            # distance calculation
            D[u, v] = np.sqrt(u ** 2 + v ** 2)

    H = D > D0
    # Masc applied to image
    m_masc = H.shape[0]
    n_masc = H.shape[1]

    for x in range(int(m_masc / 2)):
        for y in range(int(n_masc / 2)):
            H[m_masc - x - 1, n_masc - y - 1] = H[x, y]
            H[m_masc - x - 1, y] = H[x, y]
            H[x, n_masc - y - 1] = H[x, y]

    H = np.fft.fftshift(H)

    G_T = fourierTransform * H

    G = np.fft.fftshift(G_T)
```

```
    I_f = np.fft.ifft2(G_T)
    plt.figure()
    # Original Image
    plt.subplot(1, 2, 1), plt.title("Imagen original")
    plt.imshow(I, cmap='gray')
    # Output image
    plt.subplot(1, 2, 2), plt.title("Imagen transformada inversa")
    plt.imshow(np.uint8(np.abs(I_f)), cmap='gray')
    plt.show()

I = imageio.imread("image.jpg")
idealHighPassFilter(I)
```

3.6. Filtro Gaussiano (Paso-Alto)

Código 13: Código para el método del Filtro Gaussiano (Paso-Alto).

```
clc;
clear;
close all;
pkg load image
#High pass filter using gauss
#Inputs - A: An image to filter
#Outputs - A_f: final image that applied a filter to A

A=imread('image.jpg');

A_o = im2double(A);
#Get the fourier transform
A = fftshift(fft2(A_o));
#Get the size
[m,n]=size(A);
fc=1;
H=zeros(m,n);
#Masc applied to image
for u=1:m
  for v=1:n

    D_uv=sqrt(u^2+v^2);

    H(u,v)=1-e^(-(D_uv**2/(2*fc**2)));
  endfor
endfor

#Complete the masc of the filter
for x=1:round(m/2)
  for y=1:round(n/2)

    H(m-x+1,n-y+1)=H(x,y);
    H(m-x+1,y)=H(x,y);
    H(x,n-y+1)=H(x,y);
```

```
  endfor
endfor
H = fftshift(H);

G = A.*H;

G=fftshift(G);

A_f=ifft2(G);

#Original Image
subplot(2,1,1)
imshow(A_o)
title('Imagen Original')
#Final image
A_f=im2uint8(real(A_f));
subplot(2,1,2)
imshow(A_f)
title('Imagen con el filtro de gauss paso alto')
```

3.7. Filtro Butterworth (Paso-Alto)

Código 14: Código para el método Filtro Butterworth (Paso-Alto).

```
import imageio
import matplotlib.pyplot as plt
import numpy as np


def butterWorthHighFilter(I):
    """
    ButterWorth High filter applying fourier transform
    :param I: image to filter
    :return: final image
    """
    #Get the size of the image
    M = I.shape[0]
    N = I.shape[1]
    #Get the Fourier Transform of the image
    fourierTransform = np.fft.fftshift(np.fft.fft2(I[:,:,1]))
    #Asign the cut-off frequency
    D0 = 10
    #Get Euclidean Distance
    D = np.zeros([M, N])
    for u in range(M):
        for v in range(N):
            #distance calculation
            D_temp = np.sqrt(u ** 2 + v ** 2)
            D[u,v] = 1/(1+(D0/(1+D_temp)**(2*1))) #Set order to 1
    H = D
    #Masc applied to image
    m_masc = H.shape[0]
    n_masc = H.shape[1]
```

```
    for x in range(int(m_masc / 2)):
        for y in range(int(n_masc / 2)):
            H[m_masc - x - 1, n_masc - y - 1] = H[x, y]
            H[m_masc - x - 1, y] = H[x, y]
            H[x, n_masc - y - 1] = H[x, y]

    H = np.fft.fftshift(H)

    G_T = fourierTransform * H

    G = np.fft.fftshift(G_T)

    I_f = np.fft.ifft2(G)
    plt.figure()
    # Original Image
    plt.subplot(1, 2, 1), plt.title("Imagen original")
    plt.imshow(I, cmap='gray')
    # Output image
    plt.subplot(1, 2, 2), plt.title("Imagen transformada inversa")
    plt.imshow(np.uint8(np.abs(I_f)), cmap='gray')
    plt.show()

I = imageio.imread("image.jpg")
butterWorthHighFilter(I)
```

4. Semana 10:Restauración de imágenes

4.1. Filtro Promedio

Código 15: Código para el filtro promedio.

```
import numpy as np
import imageio as im
import matplotlib.pyplot as plt

def filter(side,B,A_t,m,n):
    if side == "E1":
        W = B[0, 0] + B[0, 1] + B[1, 0] + B[1, 1]
        A_t[0, 0] = (1 / 4) * W
    elif side == "E2":
        W = B[0, n - 1] + B[0, n - 2] + B[1, n - 1] + B[1, n - 2]
        A_t[0, n - 1] = (1 / 4) * W
    elif side == "E3":
        W = B[m - 1, 0] + B[m - 1, 1] + B[m - 2, 0] + B[m - 2, 1]
        A_t[m - 1, 0] = (1 / 4) * W
    elif side == "E4":
        W = B[m - 1, n - 1] + B[m - 1, n - 2] + B[m - 2, n - 1] + B[m - 2, n - 2]
        A_t[m - 1, n - 1] = (1 / 4) * W
    elif side == "BU":
        for y in range(1, n - 1):
            Wf1 = B[0, y - 1] + B[0, y] + B[0, y + 1]
```

```python
            Wf2 = B[1, y - 1] + B[1, y] + B[1, y + 1]
            A_t[0, y] = (1 / 6) * (Wf1 + Wf2)
    elif side == "BD":
        for y in range(1, n - 1):
            Wf1 = B[m - 2, y - 1] + B[m - 2, y] + B[m - 2, y + 1]
            Wf2 = B[m - 1, y - 1] + B[m - 1, y] + B[m - 1, y + 1]
            A_t[m - 1, y] = (1 / 6) * (Wf1 + Wf2)
    elif side == "BR":
        for x in range(1, m - 1):
            Wc1 = B[x - 1, n - 2] + B[x, n - 2] + B[x + 1, n - 2]
            Wc2 = B[x - 1, n - 1] + B[x, n - 1] + B[x + 1, n - 1]
            A_t[x, n - 1] = (1 / 6) * (Wc1 + Wc2)

    elif side == "BL":
        for x in range(1, m - 1):
            Wc1 = B[x - 1, 0] + B[x, 0] + B[x + 1, 0]
            Wc2 = B[x - 1, 1] + B[x, 1] + B[x + 1, 1]
            A_t[x, 0] = (1 / 6) * (Wc1 + Wc2)
    elif side == "C":
        for x in range(1, m - 1):
            for y in range(1, n - 1):
                Wf1 = B[x - 1, y - 1] + B[x - 1, y] + B[x - 1, y + 1]
                Wf2 = B[x, y - 1] + B[x, y] + B[x, y + 1]
                Wf3 = B[x + 1, y - 1] + B[x + 1, y] + B[x + 1, y + 1]
                A_t[x, y] = (1 / 9) * (Wf1 + Wf2 + Wf3)
def promFilter(B):
    """
    Improve the quality of an image by delete some noice of an image with mean filter
    :param B: gets an black and white image with noice
    :return: an image without noice
    """

    (m, n) = B.shape
    A_t = np.zeros((m, n))

    filter("E1", B, A_t, m, n)
    filter("E2", B, A_t, m, n)
    filter("E3", B, A_t, m, n)
    filter("E4", B, A_t, m, n)

    filter("BU", B, A_t, m, n)
    filter("BD", B, A_t, m, n)


    filter("BR", B, A_t, m, n)
    filter("BL", B, A_t, m, n)

    filter("C", B, A_t, m, n)

    imgDT = np.iinfo(np.uint8)
    imax = A_t * imgDT.max
    imax[imax > imgDT.max] = imgDT.max
    imax[imax < imgDT.min] = imgDT.min
    return imax.astype(np.uint8)
```

```
#Open the image
I = im.imread("filename.jpg")

info = np.iinfo(I.dtype)
I = I.astype(np.float64) / info.max

#Pass the image to uint8
imgDT = np.iinfo(np.uint8)
imax = I * imgDT.max
imax[imax > imgDT.max] = imgDT.max
imax[imax < imgDT.min] = imgDT.min
A = imax.astype(np.uint8)


#Plot the original image
plt.figure(1)
plt.subplot(121)
plt.title("Imagen con ruido")
plt.imshow(A, cmap='gray', vmin = 0, vmax = 255, interpolation='none')


#Plot the final image
B = promFilter(I)

plt.subplot(122)
plt.title("Imagen Filtrada Promedio")
plt.imshow(B, cmap='gray', vmin = 0, vmax = 255, interpolation='none')


plt.show()
```

4.2. Filtro Promedio Geométrico

Código 16: Código para el filtro promedio geométrico .

```
import numpy as np
import imageio as im
import matplotlib.pyplot as plt

def filter(side,B,A_t,m,n):
    if side == "E1":
        W = B[0, 0] * B[0, 1] * B[1, 0] * B[1, 1]
        A_t[0, 0] = (1 / 4) * W
    elif side == "E2":
        W = B[0, n - 1] * B[0, n - 2] * B[1, n - 1] * B[1, n - 2]
        A_t[0, n - 1] = (1 / 4) * W
    elif side == "E3":
        W = B[m - 1, 0] * B[m - 1, 1] * B[m - 2, 0] * B[m - 2, 1]
        A_t[m - 1, 0] = (1 / 4) * W
    elif side == "E4":
```

```
        W = B[m - 1, n - 1] * B[m - 1, n - 2] * B[m - 2, n - 1] * B[m - 2, n - 2]
        A_t[m - 1, n - 1] = (1 / 4) * W
    elif side == "BU":
        for y in range(1, n - 1):
            Wf1 = B[0, y - 1] * B[0, y] * B[0, y + 1]
            Wf2 = B[1, y - 1] * B[1, y] * B[1, y + 1]
            A_t[0, y] = (1 / 6) * (Wf1 + Wf2)
    elif side == "BD":
        for y in range(1, n - 1):
            Wf1 = B[m - 2, y - 1] * B[m - 2, y] * B[m - 2, y + 1]
            Wf2 = B[m - 1, y - 1] * B[m - 1, y] * B[m - 1, y + 1]
            A_t[m - 1, y] = (1 / 6) * (Wf1 + Wf2)
    elif side == "BR":
        for x in range(1, m - 1):
            Wc1 = B[x - 1, n - 2] * B[x, n - 2] * B[x + 1, n - 2]
            Wc2 = B[x - 1, n - 1] * B[x, n - 1] * B[x + 1, n - 1]
            A_t[x, n - 1] = (1 / 6) * (Wc1 + Wc2)

    elif side == "BL":
        for x in range(1, m - 1):
            Wc1 = B[x - 1, 0] * B[x, 0] * B[x + 1, 0]
            Wc2 = B[x - 1, 1] * B[x, 1] * B[x + 1, 1]
            A_t[x, 0] = (1 / 6) * (Wc1 + Wc2)
    elif side == "C":
        for x in range(1, m - 1):
            for y in range(1, n - 1):
                Wf1 = B[x - 1, y - 1] * B[x - 1, y] * B[x - 1, y + 1]
                Wf2 = B[x, y - 1] * B[x, y] * B[x, y + 1]
                Wf3 = B[x + 1, y - 1] * B[x + 1, y] * B[x + 1, y + 1]
                A_t[x, y] = (1 / 9) * (Wf1 + Wf2 + Wf3)
def promGeoFilter(B):
    """
        Improve the quality of an image by delete some noice of an image with geometric mea
        :param B: gets an black and white image with noice
        :return: an image without noice
        """

    (m, n) = B.shape
    A_t = np.zeros((m, n))

    filter("E1", B, A_t, m, n)
    filter("E2", B, A_t, m, n)
    filter("E3", B, A_t, m, n)
    filter("E4", B, A_t, m, n)

    filter("BU", B, A_t, m, n)
    filter("BD", B, A_t, m, n)


    filter("BR", B, A_t, m, n)
    filter("BL", B, A_t, m, n)

    filter("C", B, A_t, m, n)
```

```
    imgDT = np.iinfo(np.uint8)
    imax = A_t * imgDT.max
    imax[imax > imgDT.max] = imgDT.max
    imax[imax < imgDT.min] = imgDT.min
    return imax.astype(np.uint8)



#Open the image
I = im.imread("filename.jpg")

info = np.iinfo(I.dtype)
I = I.astype(np.float64) / info.max

#Pass the image to uint8
imgDT = np.iinfo(np.uint8)
imax = I * imgDT.max
imax[imax > imgDT.max] = imgDT.max
imax[imax < imgDT.min] = imgDT.min
A = imax.astype(np.uint8)


#Plot the original image
plt.figure(1)
plt.subplot(121)
plt.title("Imagen con ruido")
plt.imshow(A, cmap='gray', vmin = 0, vmax = 255, interpolation='none')



B = promGeoFilter(I)
#Plot the final image
plt.subplot(122)
plt.title("Imagen Filtrada Promedio")
plt.imshow(B, cmap='gray', vmin = 0, vmax = 255, interpolation='none')


plt.show()
```

4.3. Filtro Promedio Armónico

Código 17: Código para el método del filtro Promedio Armónico.

```
clc;
clear;
close all;
pkg load image;

#Filter for the armonic mean filter that uses a window of 3x3
#Inputs - B: An image with noice
#Outputs - A_t: An image withouth noice
function A_t = filt_prom_arm(B)

```

```
B = double(B);
[m,n] = size(B);
A_t = zeros(m,n);

B_i = B.^ - 1;
mask = isinf(B_i);
B_i(mask) = 0.0;


W = B_i(1,1) + B_i(1,2) + B_i(2,1) + B_i(2,2);
A_t(1,1) = 4/W;

W = B_i(1,n) + B_i(1,n - 1) + B_i(2,n) + B_i(2,n - 1);
A_t(1,n) = 4/W;


W = B_i(m,1) + B_i(m,2) + B_i(m - 1,1) + B_i(m - 1,2);
A_t(m,1) = 4/W;


W = B_i(m,n) + B_i(m,n - 1) + B_i(m - 1,n) + B_i(m - 1,n - 1);
A_t(m,n) = 4/W;



for y = 2:n - 1
  Wnf1 = B_i(1,y - 1) + B_i(1,y) + B_i(1,y + 1);
  Wnf2 = B_i(2,y - 1) + B_i(2,y) + B_i(2,y + 1);
  A_t(1,y) = 6/(Wnf1 + Wnf2);

  Wnf1 = B_i(m - 1,y - 1) + B_i(m - 1,y) + B_i(m - 1,y + 1);
  Wnf2 = B_i(m,y - 1) + B_i(m,y) + B_i(m,y + 1);
  A_t(m,y) = 6/(Wnf1 + Wnf2);
endfor

for x = 2:m - 1
  Wnc1 = B_i(x - 1,n - 1) + B_i(x,n - 1) + B_i(x + 1,n - 1);
  Wnc2 = B_i(x - 1,n) + B_i(x,n) + B_i(x + 1,n);
  A_t(x,n) = 6/(Wnc1 + Wnc2);

  Wnc1 = B_i(x - 1,1) + B_i(x,1) + B_i(x + 1,1);
  Wnc2 = B_i(x - 1,2) + B_i(x,2) + B_i(x + 1,2);
  A_t(x,1) = 6/(Wnc1 + Wnc2);
endfor

for x = 2:m - 1
  for y = 2:n - 1
    Wf1 = B_i(x - 1,y - 1) + B_i(x - 1,y) + B_i(x - 1,y + 1);
    Wf2 = B_i(x,y - 1) + B_i(x,y) + B_i(x,y + 1);
    Wf3 = B_i(x + 1,y - 1) + B_i(x + 1,y) + B_i(x + 1,y + 1);
    A_t(x,y) = 9/(Wf1 + Wf2 + Wf3);
  endfor
endfor
```

```
  A_t = uint8(A_t);
endfunction




A = imread('gaussianNoise.jpg');
#Original image
subplot(1,2,1)
imshow(A)
title('Original Image')

A_t = filt_prom_arm(A);
#Final image
subplot(1,2,2)
imshow(A_t)
title('Mean Armonic Filter')
```

4.4. Filtro Contra - Armónico Promedio

Código 18: Código para el filtro contra armónico promedio.

```python
import numpy as np
import imageio as im
import matplotlib.pyplot as plt

def filter(side,B,A_t,m,n,R):
    BR1 = np.power(B, R + 1)
    BR = np.power(B, R)
    #Apply filter to corners of the image
    if side == "E1":
        Wn = BR1[0, 0] + BR1[0, 1] + BR1[1, 0] + BR1[1, 1]
        Wd = BR[0, 0] + BR[0, 1] + BR[1, 0] + BR[1, 1]
        A_t[0, 0] = Wn / Wd
    elif side == "E2":
        Wn = BR1[0, n - 1] + BR1[0, n - 2] + BR1[1, n - 1] + BR1[1, n - 2]
        Wd = BR[0, n - 1] + BR[0, n - 2] + BR[1, n - 1] + BR[1, n - 2]
        A_t[0, n - 1] = Wn / Wd
    elif side == "E3":
        Wn = BR1[m - 1, 0] + BR1[m - 1, 1] + BR1[m - 2, 0] + BR1[m - 2, 1]
        Wd = BR[m - 1, 0] + BR[m - 1, 1] + BR[m - 2, 0] + BR[m - 2, 1]
        A_t[m - 1, 0] = Wn / Wd
    elif side == "E4":
        Wn = BR1[m - 1, n - 1] + BR1[m - 1, n - 2] + BR1[m - 2, n - 1] + BR1[m - 2, n - 2]
        Wd = BR[m - 1, n - 1] + BR[m - 1, n - 2] + BR[m - 2, n - 1] + BR[m - 2, n - 2]
        A_t[m - 1, n - 1] = Wn / Wd
    #Apply filter to edges
    elif side == "BU":
        for y in range(1, n - 1):

            Wnf1 = BR1[0, y - 1] + BR1[0, y] + BR1[0, y + 1]
            Wnf2 = BR1[1, y - 1] + BR1[1, y] + BR1[1, y + 1]
            Wn = Wnf1 + Wnf2
```

```
            Wdf1 = BR[0, y - 1] + BR[0, y] + BR[0, y + 1]
            Wdf2 = BR[1, y - 1] + BR[1, y] + BR[1, y + 1]
            Wd = Wdf1 + Wdf2
            A_t[0, y] = Wn / Wd
    elif side == "BD":
        for y in range(1, n - 1):
            Wnf1 = BR1[m - 2, y - 1] + BR1[m - 2, y] + BR1[m - 2, y + 1]
            Wnf2 = BR1[m - 1, y - 1] + BR1[m - 1, y] + BR1[m - 1, y + 1]
            Wn = Wnf1 + Wnf2
            Wdf1 = BR[m - 2, y - 1] + BR[m - 2, y] + BR[m - 2, y + 1]
            Wdf2 = BR[m - 1, y - 1] + BR[m - 1, y] + BR[m - 1, y + 1]
            Wd = Wdf1 + Wdf2
            A_t[m - 1, y] = Wn / Wd
    elif side == "BR":
        for x in range(1, m - 1):

            Wnc1 = BR1[x - 1, n - 2] + BR1[x, n - 2] + BR1[x + 1, n - 2]
            Wnc2 = BR1[x - 1, n - 1] + BR1[x, n - 1] + BR1[x + 1, n - 1]
            Wn = Wnc1 + Wnc2

            Wdc1 = BR[x - 1, n - 2] + BR[x, n - 2] + BR[x + 1, n - 2]
            Wdc2 = BR[x - 1, n - 1] + BR[x, n - 1] + BR[x + 1, n - 1]
            Wd = Wdc1 + Wdc2
            A_t[x, n - 1] = Wn / Wd

    elif side == "BL":
        for x in range(1, m - 1):

            Wnc1 = BR1[x - 1, n - 2] + BR1[x, n - 2] + BR1[x + 1, n - 2]
            Wnc2 = BR1[x - 1, n - 1] + BR1[x, n - 1] + BR1[x + 1, n - 1]
            Wn = Wnc1 + Wnc2

            Wdc1 = BR[x - 1, n - 2] + BR[x, n - 2] + BR[x + 1, n - 2]
            Wdc2 = BR[x - 1, n - 1] + BR[x, n - 1] + BR[x + 1, n - 1]
            Wd = Wdc1 + Wdc2
            A_t[x, n - 1] = Wn / Wd

            Wnc1 = BR1[x - 1, 0] + BR1[x, 0] + BR1[x + 1, 0]
            Wnc2 = BR1[x - 1, 1] + BR1[x, 1] + BR1[x + 1, 1]
            Wn = Wnc1 + Wnc2

            Wdc1 = BR[x - 1, 0] + BR[x, 0] + BR[x + 1, 0]
            Wdc2 = BR[x - 1, 1] + BR[x, 1] + BR[x + 1, 1]
            Wd = Wdc1 + Wdc2
            A_t[x, 0] = Wn / Wd
    #Apply to center
    elif side == "C":
        for x in range(1, m - 1):
            for y in range(1, n - 1):

                Wnf1 = BR1[x - 1, y - 1] + BR1[x - 1, y] + BR1[x - 1, y + 1]
                Wnf2 = BR1[x, y - 1] + BR1[x, y] + BR1[x, y + 1]
                Wnf3 = BR1[x + 1, y - 1] + BR1[x + 1, y] + BR1[x + 1, y + 1]
```

```
                Wn = Wnf1 + Wnf2 + Wnf3

                Wdf1 = BR[x - 1, y - 1] + BR[x - 1, y] + BR[x - 1, y + 1]
                Wdf2 = BR[x, y - 1] + BR[x, y] + BR[x, y + 1]
                Wdf3 = BR[x + 1, y - 1] + BR[x + 1, y] + BR[x + 1, y + 1]
                Wd = Wdf1 + Wdf2 + Wdf3

                A_t[x, y] = Wn / Wd
def contrMeanFilter(B):
    """
    Apply contr armonic filter
    :param B: image to reconstruct
    :return: reconstructed image
    """
    R = 1
    (m, n) = B.shape
    A_t = np.zeros((m, n))

    filter("E1", B, A_t, m, n,R)
    filter("E2", B, A_t, m, n,R)
    filter("E3", B, A_t, m, n,R)
    filter("E4", B, A_t, m, n,R)

    filter("BU", B, A_t, m, n,R)
    filter("BD", B, A_t, m, n,R)


    filter("BR", B, A_t, m, n,R)
    filter("BL", B, A_t, m, n,R)

    filter("C", B, A_t, m, n,R)
    #Apply unint8 to image
    imgDT = np.iinfo(np.uint8)
    imax = A_t * imgDT.max
    imax[imax > imgDT.max] = imgDT.max
    imax[imax < imgDT.min] = imgDT.min
    return imax.astype(np.uint8)




I = im.imread("filename.jpg")

info = np.iinfo(I.dtype)
I = I.astype(np.float64) / info.max

#Convert image to uint8
imgDT = np.iinfo(np.uint8)
imax = I * imgDT.max
imax[imax > imgDT.max] = imgDT.max
imax[imax < imgDT.min] = imgDT.min
A = imax.astype(np.uint8)
```

```
#Original image
plt.figure(1)
plt.subplot(121)
plt.title("Imagen con ruido")
plt.imshow(A, cmap='gray', vmin = 0, vmax = 255, interpolation='none')



B = contrMeanFilter(I)

#Final image
plt.subplot(122)
plt.title("Imagen Filtrada Promedio")
plt.imshow(B, cmap='gray', vmin = 0, vmax = 255, interpolation='none')


plt.show()
```

4.5. Filtro Punto Medio

Código 19: Código para el método del filtro Punto Medio .

```
clc;
clear;
close all;
pkg load image;


#Midpoint filter that deletes salt and peper noice from image
#using a window of 3x3
#Inputs — B: image to filter
#Output — A_t: clean image of salt and pepper noice
function A_t=filt_punt_med(B)


  B=im2double(B);
  [m,n]=size(B);
  A_t=zeros(m,n);

  #First corner
  Wmax=max(max(B(1:2,1:2)));
  Wmin=min(min(B(1:2,1:2)));
  A_t(1,1)=(Wmax+Wmin)/2;

  #Second corner
  Wmax=max(max(B(1:2,n-1:n)));
  Wmin=min(min(B(1:2,n-1:n)));
  A_t(1,n)=(Wmax+Wmin)/2;

  #Third corner
  Wmax=max(max(B(m-1:m,1:2)));
  Wmin=min(min(B(m-1:m,1:2)));
  A_t(m,1)=(Wmax+Wmin)/2;
```

```
  #Fourth corner
  Wmax=max(max(B(m-1:m,n-1:n)));
  Wmin=min(min(B(m-1:m,n-1:n)));
  A_t(m,n)=(Wmax+Wmin)/2;


  #Superior edge
  for y=2:n-1
    Wmax=max(max(B(1:2,y-1:y+1)));
    Wmin=min(min(B(1:2,y-1:y+1)));
    A_t(1,y)=(Wmax+Wmin)/2;
  #Down edge
    Wmax=max(max(B(m-1:m,y-1:y+1)));
    Wmin=min(min(B(m-1:m,y-1:y+1)));
    A_t(m,y)=(Wmax+Wmin)/2;
  endfor
  #Right edge
  for x=2:m-1
    Wmax=max(max(B(x-1:x+1,n-1:n)));
    Wmin=min(min(B(x-1:x+1,n-1:n)));
    A_t(x,n)=(Wmax+Wmin)/2;
  #Left edge
    Wmax=max(max(B(x-1:x+1,1:2)));
    Wmin=min(min(B(x-1:x+1,1:2)));
    A_t(x,1)=(Wmax+Wmin)/2;
  endfor

  #Center
  for x=2:m-1
    for y=2:n-1
      Wmax=max(max(B(x-1:x+1,y-1:y+1)));
      Wmin=min(min(B(x-1:x+1,y-1:y+1)));
      A_t(x,y)=(Wmax+Wmin)/2.0;
    endfor
  endfor

  A_t=im2uint8(A_t);
endfunction



A=imread('imageSandP.jpg'); %lectura de imagen escala a grises
%Crear un ruido del tipo sal y pimienta con la funcion de Octave imnoise



subplot(1,2,1) %posicionamiento de imagen en el grafico
imshow(A)
title('Imagen con Ruido Sal y Pimienta')

A_t=filt_punt_med(A); %llamado de la funcion Filtro Punto Medio

subplot(1,2,2)
```

```
imshow(A_t)
title('Imagen Filtrada Punto Medio')
```

4.6. Filtro Ideal (Rechazo de Banda)

Código 20: Código para el filtro ideal RB.

```
import imageio
import matplotlib.pyplot as plt
import numpy as np


def idealRBFilter(I, f_c, w):
    """
    Ideal bandwith filter using fourier transformrs
    :param I: image to filter
    :return: filtered image
    """
    #Get the size of the image
    M = I.shape[0]
    N = I.shape[1]
    #Get the Fourier Transform of the image
    fourierTransform = np.fft.fftshift(np.fft.fft2(I[:, :]))
    #Asign the cut-off frequency
    D0 = f_c
    #Get Euclidean Distance
    D = np.zeros([M, N])
    for u in range(M):
        for v in range(N):
            # Calculo de distancias
            D[u, v] = np.sqrt(u ** 2 + v ** 2)

    H = np.ones([M, N])
    W = w
    indx = D0-W/2< D
    indy = D0+W/2 >= D
    index = np.logical_and(indx,indy)
    H[index] = 0

    #Applying the masc
    m_masc = H.shape[0]
    n_masc = H.shape[1]

    for x in range(int(m_masc / 2)):
        for y in range(int(n_masc / 2)):
            H[m_masc - x - 1, n_masc - y - 1] = H[x, y]
            H[m_masc - x - 1, y] = H[x, y]
            H[x, n_masc - y - 1] = H[x, y]

    H = np.fft.fftshift(H)

    G_T = fourierTransform * H
```

```
    G = np.fft.ifft2(G_T)
    plt.figure()
    # original image
    plt.subplot(1, 2, 1), plt.title("Original Image")
    plt.imshow(I, cmap='gray')
    # final image
    plt.subplot(1, 2, 2), plt.title("Idea RB Filter final image")
    plt.imshow(np.uint8(np.abs(G)), cmap='gray')
    plt.show()

I = imageio.imread("idealRB.jpg")
idealRBFilter(I, 58, 10)
```

4.7. Filtro Gaussiano (Rechazo de Banda)

Código 21: Código para el filtro rechazo de banda gaussiano

```
import imageio
import matplotlib.pyplot as plt
import numpy as np


def gaussRBFilter(I):
    #Get the size of the image
    M = I.shape[0]
    N = I.shape[1]
    #Get the Fourier Transform of the image
    fourierTransform = np.fft.fftshift(np.fft.fft2(I[:,:]))
    #Asign the cut-off frequency
    D0 = 58
    W = 10
    #Get Euclidean Distance
    D = np.zeros([M, N])
    for u in range(M):
        for v in range(N):
            # Distance compute
            D_temp = np.sqrt(u ** 2 + v ** 2)
            D[u, v] = 1-np.exp(-(1/2)*((D_temp**2-D0**2)/(D_temp*W))**2)

    H = D
    #Masc calculus
    m_masc = H.shape[0]
    n_masc = H.shape[1]

    for x in range(int(m_masc / 2)):
        for y in range(int(n_masc / 2)):
            H[m_masc - x - 1, n_masc - y - 1] = H[x, y]
            H[m_masc - x - 1, y] = H[x, y]
            H[x, n_masc - y - 1] = H[x, y]

    H = np.fft.fftshift(H)

    G_T = fourierTransform * H
```

```
    G = np.fft.fftshift(G_T)

    I_f = np.fft.ifft2(G)
    plt.figure()
    # Original image
    plt.subplot(1, 2, 1), plt.title("Imagen original")
    plt.imshow(I, cmap='gray')
    # Final image
    plt.subplot(1, 2, 2), plt.title("Imagen con filtro RB de Gauss")
    plt.imshow(np.uint8(np.abs(I_f)), cmap='gray')
    plt.show()

I = imageio.imread("image_gauss.png")
gaussRBFilter(I)
```

4.8. Filtro Butterworth (Rechazo de Banda)

Código 22: Código para el Filtro Butterworth (Rechazo de Banda).

```
import imageio
import matplotlib.pyplot as plt
import numpy as np


def butterWorthRBFilter(I,f_c, w):
    """
    ButterWorth RB filter filter applying fourier transform
    :param I: image to filter
    :return: final image
    """
    #Get the size of the image
    M = I.shape[0]
    N = I.shape[1]
    #Get the Fourier Transform of the image
    fourierTransform = np.fft.fftshift(np.fft.fft2(I[:,:]))
    #Asign the cut-off frequency
    D0 = f_c
    W = w
    #Get Euclidean Distance
    H = np.zeros([M, N])
    for u in range(M):
        for v in range(N):
            # Distance procedure
            Duv = np.sqrt(u ** 2 + v ** 2)
            #Masc Calculus
            H[u, v] = 1 / (1 + (Duv * W / (Duv ** 2 - D0 ** 2)) ** (2 * 1)) #Set order to 1

    #Masc applied to image
    m_masc = H.shape[0]
    n_masc = H.shape[1]

    for x in range(int(m_masc / 2)):
```

```
        for y in range(int(n_masc / 2)):
            H[m_masc - x - 1, n_masc - y - 1] = H[x, y]
            H[m_masc - x - 1, y] = H[x, y]
            H[x, n_masc - y - 1] = H[x, y]

    H = np.fft.fftshift(H)

    G_T = fourierTransform * H

    G = np.fft.fftshift(G_T)

    I_f = np.fft.ifft2(G)
    plt.figure()
    # Original Image
    plt.subplot(1, 2, 1), plt.title("Original Image")
    plt.imshow(I, cmap='gray')
    # Output image
    plt.subplot(1, 2, 2), plt.title("RB butterworth Filter")
    plt.imshow(np.uint8(np.abs(I_f)), cmap='gray')
    plt.show()

I = imageio.imread("image_gauss.png")
butterWorthRBFilter(I, 58, 10)
```

5. Semana 11:Procesamiento de imágenes morfológicas

5.1. Operaciones morfológicas básicas

Código 23: Complemento,unión,intersección y diferencia de imagenes binarias.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x

def basicOP(file = None,filename = None,file2 = None, filename2 = None):
    """
    Arguments:
             file: an image file that is going to be processed
             filename: a filename of the file to be processed
             file2: an image file that is going to be processed
             filename2: a filename of the file to be processed
    Returns:
            Different operations of binary matrices.
    """
```

```
    if file  is None and filename is None:
        return "Error, both arguments cant be None"
    if file is None:
        file = ImageOps.grayscale(Image.open(filename))

    if file2  is None and filename2 is None:
        return "Error, both arguments cant be None"
    if file2 is None:
        file2 = ImageOps.grayscale(Image.open(filename2))
    image = np.asarray(file)
    image2 = np.asarray(file2)
    image = to_binary(image)
    image2 = to_binary(image2)
    #complement
    complement = to_binary(~image)
    #union of images
    union = image | image2
    #intersection of images
    intersection = image & image2
    #difference of images
    difference = abs( union - intersection)
    fig = plt.figure(figsize=(10, 7))
    rows = 3
    columns = 2
    #plots all the images
    fig.add_subplot(rows, columns, 1)
    plt.imshow(image * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Original 1")
    plt.axis('off')

    fig.add_subplot(rows, columns, 2)
    plt.imshow(image2  * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Original 2")
    plt.axis('off')

    fig.add_subplot(rows, columns, 3)
    plt.imshow(complement * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Complement")
    plt.axis('off')

    fig.add_subplot(rows, columns, 4)
    plt.imshow(union*255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("union")
    plt.axis('off')

    fig.add_subplot(rows, columns, 5)
    plt.imshow(intersection * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("intersection")
    plt.axis('off')

    fig.add_subplot(rows, columns, 6)
    plt.imshow(difference * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("difference")
    plt.axis('off')
```

```
    plt.show()
```

5.2. Apertura y clausura de imágenes binarias

Código 24: Apertura y clausura de imágenes binarias.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x

def open_bin(image):
    """
    Applies the apperture effect to a binary image
    image: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(image))
    image = np.asarray(file)
    binary = to_binary(image)
    #plots the image
    cv.imshow("binary image", binary*255)
    #Gets the kernel to use
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (2, 2))
    #Applies the apperture effect
    binary = cv.morphologyEx(binary, cv.MORPH_OPEN, kernel)
    cv.imshow("open-result", binary*255)
    cv.waitKey(0)
    cv.destroyAllWindows()

def close_bin(image):
    """
    Applies the closure effect to a binary image
    image: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(image))
    image = np.asarray(file)
    binary = to_binary(image)
    #plots the image
    cv.imshow("binary image", binary*255)
    #Gets the kernel to use
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (2, 2))
    #Applies the closure effect
    binary = cv.morphologyEx(binary, cv.MORPH_CLOSE, kernel)
    cv.imshow("close-result", binary*255)
    cv.waitKey(0)
```

```
    cv.destroyAllWindows()
```

6. Semana 11 : Detección de Bordes

6.1. Extracción de bordes

Código 25: Extracción de borde interno y externo.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x


def erosion_dilation_gradient(imagename):
    """
    Applies the morphological dilation, erosion and gradient to the image given by imagenam
    imagename: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(imagename))
    img = np.asarray(file)
    img = to_binary(img)
    #Creates the kernel to apply the effects
    kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    #Applies erosion
    erosion = cv.erode(img, kernel, iterations = 1)
    #Applies dilation
    dilation = cv.dilate(img, kernel, iterations= 1)
    #Applies gradient
    gradient1 =  dilation&~ erosion

    #Plots the different images
    fig = plt.figure(figsize=(10, 7))
    rows = 2
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(img * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Original")
    plt.axis('off')

    fig.add_subplot(rows, columns, 2)
    plt.imshow(erosion * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("erosion")
    plt.axis('off')
```

```
    fig.add_subplot(rows, columns, 3)
    plt.imshow(dilation  * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("dilation")
    plt.axis('off')

    fig.add_subplot(rows, columns, 4)
    plt.imshow(gradient1 * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("gradient1")
    plt.axis('off')

def internal_border(imagename):
    """
    Gets the internal borders of a figure
    imagename: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(imagename))
    img = np.asarray(file)
    img = to_binary(img)
    #Gets the kernel to applie the effects
    kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    #Applies erosion
    erosion = cv.erode(img, kernel, iterations = 1)
    #Applies substraction to get the border
    result = img & (~erosion)
    #plots the images
    fig = plt.figure(figsize=(10, 7))
    rows = 1
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(img * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Original")
    plt.axis('off')

    fig.add_subplot(rows, columns, 2)
    plt.imshow(result * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Internal Border")
    plt.axis('off')
    plt.show()

def external_border(imagename):
    """
    Gets the external borders of a figure
    imagename: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(imagename))
    img = np.asarray(file)
    img = to_binary(img)
    #Gets the kernel to applie the effects
    kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    #Applies dilation
    dilate = cv.dilate(img, kernel, iterations = 1)
    #Applies substraction to get the borders
```

```
    result = (~img) & dilate
    #Plots the images
    fig = plt.figure(figsize=(10, 7))
    rows = 1
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(img * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("Original")
    plt.axis('off')

    fig.add_subplot(rows, columns, 2)
    plt.imshow(result * 255, cmap = 'gray',vmin = 0,vmax = 255)
    plt.title("External Border")
    plt.axis('off')
    plt.show()
```

6.2. Rellenar huecos de una Imagen

Código 26: Método para rellenar huecos en una imagen.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x


def fill_image(imagename,iterations=1):
    """
    Fills an image using the application of dilation the ammount of
    times given by the iteration parameters.
    imagename: The filename of the image to be processed.
    iterations: The ammount of times to be executed the dilation.
    """
    file = ImageOps.grayscale(Image.open(imagename))
    img = np.asarray(file)
    img = to_binary(img)
    kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    (m,n) = img.shape
    X= img*0
    m = int(m/2)
    n = int(n/2)
    X[m][n] = 1
    for i in range(0,iterations):
        X = cv.dilate(X, kernel, iterations= 1)
        X = X&~img
```

```
    cv.imshow("Filled Image", X*255)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

6.3. Esqueleto de una imagen

Código 27: Código para el método de obtención de esqueleto de una imagen.

```
clear;
clc;
close all;
pkg load image;

function skeleton(imagename)
  %Reads the image
  A= imread(imagename);
  A = im2bw (A);
  %applies the skeleton transformation
  skeleton = bwmorph(A,'skel',Inf);
  %plots the skeleton
  imshow(skeleton)
 endfunction

 skeleton("spooky.jpg")
```

6.4. Semana 12:Dilatación y erosión de una imagen a escala de grises

Código 28: Métodos para la aplicación de dilatación y erosión de una imagen a escala de grises.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x


def dilation_erosion_grey(imagename):
    """
    Applies the dilation and erosion to a grey image
    imagename: The filename of the image to be processed.
    """
    file = ImageOps.grayscale(Image.open(imagename))
    img = np.asarray(file)
    kernel = cv.getStructuringElement(cv.MORPH_RECT,(3,3))
    cv.imshow("binary image", img)
    #Applies the erosion effect
```

```
    erosion = cv.erode(img, kernel, iterations = 1)
    #Applies the dilation Effect
    dilation = cv.dilate(img, kernel, iterations= 1)
    #calculates the gradient
    gradient1 =  dilation - erosion

    #plots the images
    fig = plt.figure(figsize=(10, 7))
    rows = 2
    columns = 2

    fig.add_subplot(rows, columns, 1)
    plt.imshow(img , cmap = 'gray',vmin = 0,vmax = 256)
    plt.title("Original")
    plt.axis('off')

    fig.add_subplot(rows, columns, 2)
    plt.imshow(erosion , cmap = 'gray',vmin = 0,vmax = 256)
    plt.title("erosion")
    plt.axis('off')

    fig.add_subplot(rows, columns, 3)
    plt.imshow(dilation  , cmap = 'gray',vmin = 0,vmax = 256)
    plt.title("dilation")
    plt.axis('off')

    fig.add_subplot(rows, columns, 4)
    plt.imshow(gradient1 , cmap = 'gray',vmin = 0,vmax = 256)
    plt.title("gradient")
    plt.axis('off')
    plt.show()
```

6.5. Apertura y clausura de una imagen a escala de grises

Código 29: Métodos para apertura y clausura de una imagen a escala de grises.

```
import numpy as np
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import cv2 as cv

def to_binary(x):
    """
    Turn a array into a CV binary array to be processed and return it
    """
    f,x = cv.threshold(x, 0, 1, cv.THRESH_BINARY | cv.THRESH_OTSU)
    return x

def open_grey(image):
    """
    image: the filename of the image to  be processed
    """
    file = ImageOps.grayscale(Image.open(image))
```

```
    image = np.asarray(file)
    cv.imshow("Image", image)
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
    #Applies the morphology close operation
    binary = cv.morphologyEx(image, cv.MORPH_OPEN, kernel)
    cv.imshow("open-result", binary)
    cv.waitKey(0)
    cv.destroyAllWindows()

def close_grey(image):
    """
    image: the filename of the image to  be processed
    """
    file = ImageOps.grayscale(Image.open(image))
    image = np.asarray(file)
    cv.imshow("Image", image)
    kernel = cv.getStructuringElement(cv.MORPH_RECT, (2, 2))
    #Applies the morphology close operation
    binary = cv.morphologyEx(image, cv.MORPH_CLOSE, kernel)
    cv.imshow("close-result", binary)
    cv.waitKey(0)
    cv.destroyAllWindows()
```