MÓDULO 2: Mejora y procesamiento de imágenes

Objetivo: Utilizando Python se pretende que el alumno logre introducirse en el nuevo software implementando conceptos adquidos en el teórico.

Metodología: Repaso de conceptos teóricos. Uso de material bibliográfico específico (Libro "Programming Computer Vision with Python", http://programmingcomputervision.com/). Discuciones pertinentes.

Higiene y Seguridad: Antes de realizar el laboratorio es necesario haber firmado la planilla que certifica la lectura y aceptación del documento "PROCEDIMIENTO DE SEGURIDAD GENERAL DEL LABORATORIO DE SEÑALES" y "NORMAS DE PROCEDIMIENTO PARA EL TRABAJO CON EQUIPOS ENERGIZADOS".

Problema 1

Dada la imagen ***sunset.jpg*** calcular el histograma y explicar el gráfico del mismo.

Problema 2

Implementar y explicar el siguiente código. Además, graficar la imagen ***flores.jpg*** antes y después de su ecualización y sus histogramas correspondientes.

```
def histeq(im, nbr_bins=256):
  imhist, bins = histogram(im.flatten(), nbr_bins, normed=True)
  cdf = imhist.cumsum()
  cdf = 255 * cdf / cdf[-1]
  im2 = interp(im.flatten(), bins[:-1], cdf)
  return im2.reshape(im.shape), cdf
```

Repetir el proceso para la imagen ***placa.jpg***. Compare los histogramas. En este caso, ¿mejoró la visualización al aplicarle ecualización de histograma?

Problema 3

Dada las 3 (tres) imágenes de Bugs Bunny (***golf.jpg, golf-1-jpg, golf-2-jpg***), calcule el histograma por canal de color (RGB) para cada una.

¿Qué información sobre la imagen se ve claramente en los histogramas por canal?

Problema 4

Suponiendo que todas las imágenes son del mismo tamaño, calcular el promedio de las mismas mediante el siguiente código:

```
def compute_average(imlist):
    averageim = np.array(Image.open(imlist[0]),'f')/3
    for imname in imlist[1:]:
        try:
            averageim += np.array(Image.open(imname))/3
        except:
            print(imname + '..skiped')
    return np.array(averageim,'uint8')
```

a- Utilizar la serie de imágenes ***tigre 01, 02, 03***.
b- Guardar el resultado como tigre_promedio.jpg. Comentar.

Problema 5

Utilizando la librería *SciPy*, aplique un filtro difuminado gaussiano a la imagen ***empire.jpg***. ¿Cómo influye la selección del parámetro sigma en el efecto del filtro?

Problema 6

El ruido tipo sal y pimienta ("salt and pepper") es un tipo especial de ruido impulsivo, que presenta un punteado característico que lo hace fácilmente identificable y que se elimina mediante el empleo del filtro de mediana.
Este filtro presenta la ventaja de no eliminar los bordes como lo haría un filtro de media, aunque resulta computacionalmente costoso debido a que requiere la ordenación de los valores de los puntos de la vecindad para calcular la mediana y así sustituir el valor de cada píxel.

a- Utilice filtrado de mediana para eliminar el ruido de la imagen "***cerebro.jpg"***. ¿Puede encontrar algún otro tipo de filtro que mejore la imagen?. En caso afirmativo, compare los mismos, ¿cuál le ofrece mejor resultado?

b- Realice lo mismo que en el apartado anterior pero utilizando el operador del Laplaciano.

0	-1	0
-1	4	-1
0	-1	0

1- Aplicar el operador laplaciano pero utilizando el filtro Laplace de la librería Scipy.

Comparar con el manual.

Problema 7

Implemente la FFT y la DCT en python (CODIGO ADJUNTO) aplicada a las imágenes ***bugbunny1.jpg y bugbunny2.jpg***. ¿Explique las diferencias entre la DCT y la FFT?

```
from scipy import fftpack
from numpy.fft import fftshift, ifftshift , fft, ifft

def dct2d(im0):
    """ Get 2D Cosine Transform of Image """
    dct = fftpack.dct(fftpack.dct(im0.T, norm='ortho').T, norm='ortho')
    return dct

def idct2d(D):
    """ Get 2D Inverse Cosine Transform of Image """
    raw = fftpack.idct(fftpack.idct(D.T, norm='ortho').T, norm='ortho')
    return raw

def fft2d(ima):
    """ Get 2D Fourier Transform of Image """
    L=fftshift(fft(fftshift(ima)))
    return L

def ifft2d(ima):
    N=fftshift(ifft(fftshift(ima)))
    return N


conejo1=array(Image.open('bugsbunny1.jpg').convert('L'))
conejo2=array(Image.open('bugsbunny2.jpg').convert('L'))

#FFT
conejo1_fft=fft2d(conejo1)
conejo2_fft=fft2d(conejo2)

#IFFT
conejo1_ifft=ifft2d(conejo1_fft)
conejo2_ifft=ifft2d(conejo2_fft)

#DCT
conejo1_dct=dct2d(conejo1)
conejo2_dct=dct2d(conejo2)

#IDCT
conejo1_idct=idct2d(conejo1_dct)
conejo2_idct=idct2d(conejo2_dct)


fig , ax = plt.subplots(2,5, figsize=(25,10))
ax[0,0].imshow(conejo1,cmap='gray')
ax[0,0].set_title('bugsbunny1 original',fontsize=10);
ax[1,0].imshow(conejo2,cmap='gray')
```

```
ax[1,0].set_title('bugsbunny2 original',fontsize=10);

ax[0,1].imshow(log(abs(conejo1_fft)),cmap='Paired')
ax[0,1].set_title('bugsbunny1 modulo fft',fontsize=10);
ax[1,1].imshow(log(abs(conejo2_fft)),cmap='Paired')
ax[1,1].set_title('bugsbunny2 modulo fft',fontsize=10);

ax[0,2].imshow(log(abs(conejo1_dct)),cmap='Paired')
ax[0,2].set_title('bugsbunny1 modulo dct',fontsize=10);
ax[1,2].imshow(log(abs(conejo2_dct)),cmap='Paired')
ax[1,2].set_title('bugsbunny2 modulo dct',fontsize=10);

ax[0,3].imshow(abs(conejo1_ifft),cmap='gray')
ax[0,3].set_title('bugsbunny1 recuperada a partir de fft',fontsize=10);
ax[1,3].imshow(abs(conejo2_ifft),cmap='gray')
ax[1,3].set_title('bugsbunny2 recuperada a partir de fft',fontsize=10);

ax[0,4].imshow(abs(conejo1_idct),cmap='gray')
ax[0,4].set_title('bugsbunny1 recuperada a partir de dct',fontsize=10);
ax[1,4].imshow(abs(conejo2_idct),cmap='gray')
ax[1,4].set_title('bugsbunny2 recuperada a partir de dct',fontsize=10);
```

Problema 8

La segmentación de imágenes se refiere al proceso de dividir una imagen dada en varias partes. Corra los siguientes códigos y analícelos. (utilizar la imagen '***im4.png'***)

```
def algo_grabcut(img, bounding_box):
    seg = np.zeros(img.shape[:2],np.uint8)
    x,y,width,height = bounding_box
    seg[y:y+height, x:x+width] = 1
    background_mdl = np.zeros((1,65), np.float64)
    foreground_mdl = np.zeros((1,65), np.float64)

    cv2.grabCut(img, seg, bounding_box, background_mdl, foreground_mdl,
5,
    cv2.GC_INIT_WITH_RECT)

    mask_new = np.where((seg==2)|(seg==0),0,1).astype('uint8')
    img = img*mask_new[:,:,np.newaxis]
    cv2.imshow('Output', img)

def box_draw(click, x, y, flag_param, parameters):
    global x_pt, y_pt, drawing, topleft_pt, bottomright_pt, img

    if click == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        x_pt, y_pt = x, y

    elif click == cv2.EVENT_MOUSEMOVE:
        if drawing:
```

```
            topleft_pt, bottomright_pt = (x_pt,y_pt), (x,y)
            image[y_pt:y, x_pt:x] = 255 - img[y_pt:y, x_pt:x]
            cv2.rectangle(image, topleft_pt, bottomright_pt, (0,255,0),
2)

    elif click == cv2.EVENT_LBUTTONUP:
        drawing = False
        topleft_pt, bottomright_pt = (x_pt,y_pt), (x,y)
        image[y_pt:y, x_pt:x] = 255 - image[y_pt:y, x_pt:x]
        cv2.rectangle(image, topleft_pt, bottomright_pt, (0,255,0), 2)
        bounding_box = (x_pt, y_pt, x-x_pt, y-y_pt)

        algo_grabcut(img, bounding_box)


drawing = False
topleft_pt, bottomright_pt = (-1,-1), (-1,-1)

img = cv2.imread("im4.png")

img = cv2.resize(img ,(500,500))
image = img.copy()
cv2.namedWindow('Frame')
cv2.setMouseCallback('Frame', box_draw)

while True:
    cv2.imshow('Frame', image)
    if cv2.waitKey(22) & 0xFF == ord('q'):
            break

    #ch = cv2.waitKey(1)
    #if ch == 32:
      #     break

cv2.destroyAllWindows()


---------

img = cv2.imread('im4.png')
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
bound_lower = np.array([36, 25, 25])
bound_upper = np.array([70, 255,255 ])

mask_green = cv2.inRange(hsv_img, bound_lower, bound_upper)
kernel = np.ones((7,7),np.uint8)

mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_CLOSE, kernel)
mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_OPEN, kernel)

seg_img = cv2.bitwise_and(img, img, mask=mask_green)
contours, hier = cv2.findContours(mask_green.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
output = cv2.drawContours(seg_img, contours, -1, (0, 0, 255), 3)

cv2.imshow("Result", seg_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Problema 9

Correr los códigos y explicar el funcionamiento de los mismos.

Umbralizacion simple

Con un solo umbral para toda la imagen

i. BINARY aplica 255 al valor que pase el umbral y 0 para el que no la pase, BINARY_INV hace lo mismo pero invirtiendo el resultado.
ii. THRESH_TRUNC. Aplica de forma similar al binario pero no asigna un cero a los valores menores al umbral, no modifica dicho valores.
iii. THRESH_TOZERO. Respeta los valores que superen el umbral y asigna cero a aquellos que no cumplan, THRESH_TOZERO_INV hace lo lo mismo pero invertido.

```
img = cv.imread('gradiente.png',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles                              =                                  ['Original
Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Umbralizacion adaptada

A diferencia con el ejemplo anterior, en este caso umbraliza la imagen tomando por sectores la imagen y calculando el umbral apropiado para cada sector.

iv. El ADAPTIVE_THRESH_MEAN calcula la mediana de cada sector.
v. ADAPTIVE_THRESH_GAUSSIAN el valor umbral es la suma ponderada de los valores de la vecindad donde los pesos son una ventana gaussiana.

```
img = cv.imread('sudoku.jpg',0)
img = cv.medianBlur(img,5)
ret,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
th2                                                                    =
cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,1
1,2)
th3                                                                    =
cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINA
RY,11,2)
titles = ['Original Image', 'Global Thresholding (v = 127)', 'Adaptive
Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
for i in range(4):
    plt.subplot(2,2,i+1)
    plt.imshow(images[i],'gray')
    plt.title(titles[i], fontsize=10)
    plt.xticks([])
    plt.yticks([])
plt.show()
```

Umbralizacion de OTSU

Procedimiento no paramétrico que selecciona el umbral óptimo maximizando la varianza, tiene excelente respuesta frente a imágenes ruidosas, con histogramas planos, mal iluminadas.

```
img = cv.imread('noise.png',0)

# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1, img, 0, th2, blur, 0, th3]
titles   =   ['Original   Noisy   Image','Histogram','Global   Thresholding
(v=127)',
'Original Noisy Image','Histogram',"Otsu's Thresholding",
'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3], fontsize=8), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1], fontsize=8), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2], fontsize=8), plt.xticks([]), plt.yticks([])
plt.show()
#_____________________________________________________________________________
```

```
img = cv.imread('EinStein.jpg',0)

# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# plot all the images and their histograms
images = [img, 0, th1, img, 0, th2, blur, 0, th3]
titles   =   ['Original   Noisy   Image','Histogram','Global   Thresholding
(v=127)',
'Original Noisy Image','Histogram',"Otsu's Thresholding",
'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]

for i in range(3):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3], fontsize=8), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1], fontsize=8), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2], fontsize=8), plt.xticks([]), plt.yticks([])
plt.show()
```

Problema 10

La identificación de bordes (“edges”) o contornos, representa otro método de segmentación (clasificado como una “segmentación basada en frontera”), dado que un borde es toda región de la imagen donde aparece una fuerte variación del nivel de intensidad entre “pixeles” adyacentes.

A la hora de identificar un borde en una imagen, existen dos posibilidades matemáticas: aplicar un filtro basado en un operador gradiente o en un operador laplaciano. La diferencia estriba en que mientras que con el operador gradiente se obtienen los niveles de variación entre “píxeles”, con el laplaciano lo que se obtienen son los puntos de inflexión de las variaciones de intensidad, o sea, los cambios de tendencia.

Sin embargo, operacionalmente, la detección de bordes consiste en la aplicación de alguno de los núcleos de convolución (Sobel, Roberts, Laplaciano) diseñado a tal efecto. Lo que permite identificar las regiones en las que los “píxeles” experimentan fuertes variaciones en sus niveles de intensidad.

Uno de los operadores más usados es el de Sobel, cuyo kernel de convolucion varía según la dirección de detección de los bordes.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad o \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{[Sobel]}$$

A continuación, se adjunta el código para la detección de bordes de la imagen empire.jpg, utiliza el operador de Sobel (en la dirección X), comente el funcionamiento del mismo.

```
img = cv.imread('empire.jpg', cv.IMREAD_GRAYSCALE) #importación de imágen
en escala de grises
Sobelx = cv.Sobel(img, cv.CV_64F, 1, 0, ksize=1) #aplicación de Sobel en
eje X
Sobely = cv.Sobel(img, cv.CV_64F, 0, 1, ksize=1) #aplicación de Sobel en
eje Y
plt.figure()
plt.subplot(1, 3, 1)
plt.imshow(img,cmap='gray')
plt.title('Original')
plt.xticks([])
plt.yticks([])
plt.subplot(1, 3, 2)
plt.imshow(Sobelx, cmap='gray')
plt.title('Sobel en la dirección X')
plt.xticks([])
plt.yticks([])
plt.show()
plt.subplot(1, 3, 3)
plt.imshow(Sobely, cmap='gray')
plt.title('Sobel en la dirección Y')
plt.xticks([])
plt.yticks([])
plt.show()
```

Problema 11

Ejecute y explique el siguiente código.

```
import cv2
import numpy as np

# Cargar la imagen en blanco y negro
imagen = cv2.imread('bugsbunny1.jpg')  # Cambia 'imagen.jpg' por la ruta
de tu imagen

# Definir el kernel (elemento estructurante)
kernel = np.ones((7, 7), np.uint8) # Tamaño del kernel (puedes ajustarlo)

# Aplicar erosión
erosion = cv2.erode(imagen, kernel, iterations=1)

# Aplicar dilatación
dilatacion = cv2.dilate(imagen, kernel, iterations=1)
```

```
# Mostrar las imágenes original, erosionada y dilatada
cv2.imshow('Imagen Original', imagen)
cv2.imshow('Erosión', erosion)
cv2.imshow('Dilatación', dilatacion)

# Esperar a que se presione una tecla y luego cerrar las ventanas
cv2.waitKey(0)
cv2.destroyAllWindows()
```

PROPUESTOS (Puede canjearlos por actividad equivalente preacordada con cátedra)

Problema 12

"Esqueletonice" la imagen ***"horse.png".***

Buscar una figura de una persona y "esqueletonizarla".

Sugerencia: vea
http://scikit-image.org/docs/dev/auto_examples/applications/plot_morphology.html#skeletonize

Problema 13

Corra el ejemplo "Template-matching" adjuntado a continuación, y explique el funcionamiento del mismo.
(https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_template.html#sphx-glrauto-examples-features-detection-plot-template-py)

Fecha de entrega
15 de octubre – subir via Aula Virtual
Un archivo comprimido; puede trabajarse con .py o .ipynb.