Procesamiento de Imágenes Digitales - Entrega 1 **Gálvez Camacho,Dionisio**

1.

No se puede dar un código de longitud constante porque tenemos porcentajes dispersos , es decir , tenemos como máximo valor de probabilidad de veces que sale un tipo de gris de un 47% , mientras el valor más bajo es 3% , y los otros valores están entre estos dos que son 25%. Por tanto resulta imposible pensar de utilizar un código de longitud constante ya que el porcentaje más alto y bajo están muy alejados, y los del medio(25%) con el más bajo.

Por ello es mejor utilizar un código de longitud variable , es decir , se podría mejorar. Ya que como el gris que aparece con probabilidad del 47% podríamos asignarle un menor número de bits ya que es el que más se repite de los otros grises , y así optimizar la imagen. Por el otro lado al menor valor 3% se le asignaría un número mayor de bits ya que apenas se repiten.

2.

He escogido el lenguaje de programación **Python** puesto que estoy más familiarizado con él , ya que lo hemos impartido en otras asignaturas y me resulta más fácil programar con este lenguaje debido a su sencillez. A su vez me estuve documentando sobre la librería **OpenCV** para **Python** y es bastante intuitiva para los temas que vamos a trabajar en clase.

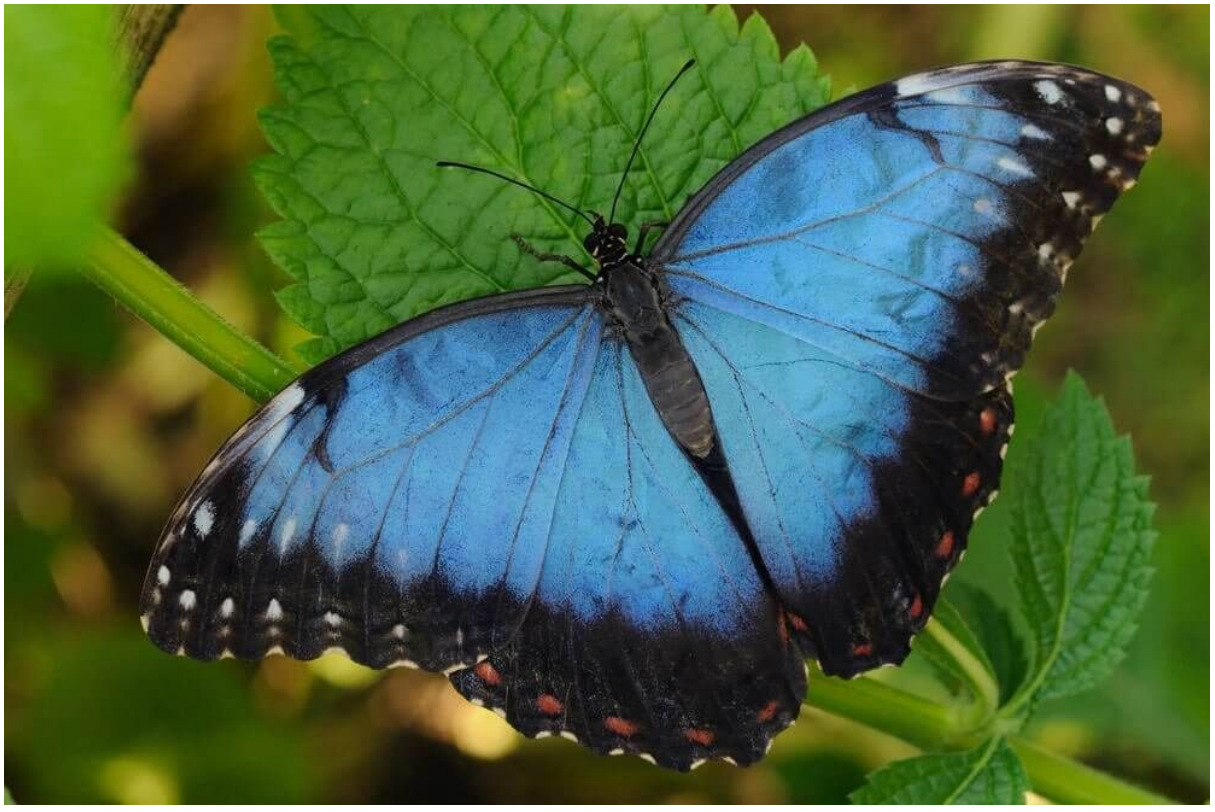
a)

```python
import cv2
import numpy as np

#Apartado 2
#a)Separa los tres canales RGB que componen la imagen y visualiza cada uno por separado como imágenes en escala de grises.
img = cv2.imread("mariposa-azul.jpg", 1) #Leemos la imagen a color

b,g,r = cv2.split(img) #hacemos split para separar r,g,b . Nótese que en OPENcv se separa como b,g,r
cv2.imshow("r", r) #para mostrar el red de la imagen
cv2.waitKey(0) #no se cierra la ventana hasta que le demos a cerrarla
cv2.imshow("g", g) #para mostrar el green de la imagen
cv2.waitKey(0)
cv2.imshow("b", b) #para mostrar el blue de la imagen
cv2.waitKey(0)
```

Imagen escogida a color:



Color r:

Color g:

Color b:

b)

```
#b)Pasa la imagen a color a escala de grises de dos formas: usando la fórmula de Y
i = 1/3*(r+g+b)
i=i.astype(np.uint8) #la matriz i la pasamos a tipo uint8 para no tener decimales
print(i)
cv2.imshow("i", i) #mostramos i
cv2.waitKey(0)

y = 0.299*r+0.587*g+0.144*b
y=y.astype(np.uint8)
print(y)
cv2.imshow("y", y)
cv2.waitKey(0)
```

Salidas del código aplicando la fórmula de **i**(modelo **HSI**) y la **“y**”(modelo **YIQ**):

En el modelo **YIQ** los colores de la escala de grises son más claros ya que la **“Y”** en este modelo denota la luminancia, medida de la cantidad de energía que un observador percibe procedente de una fuente luminosa(para televisiones) y en el **HSI** denota la Intensidad que

representa la iluminación percibida, por tanto en esta imagen saldrá más oscuro en el modelo **HS**I ya que la luz proviene del exterior , es una fotografía , y al mostrar la intensidad de la luz resalta la luz dada en ese momento en la escala de grises, que es alta.

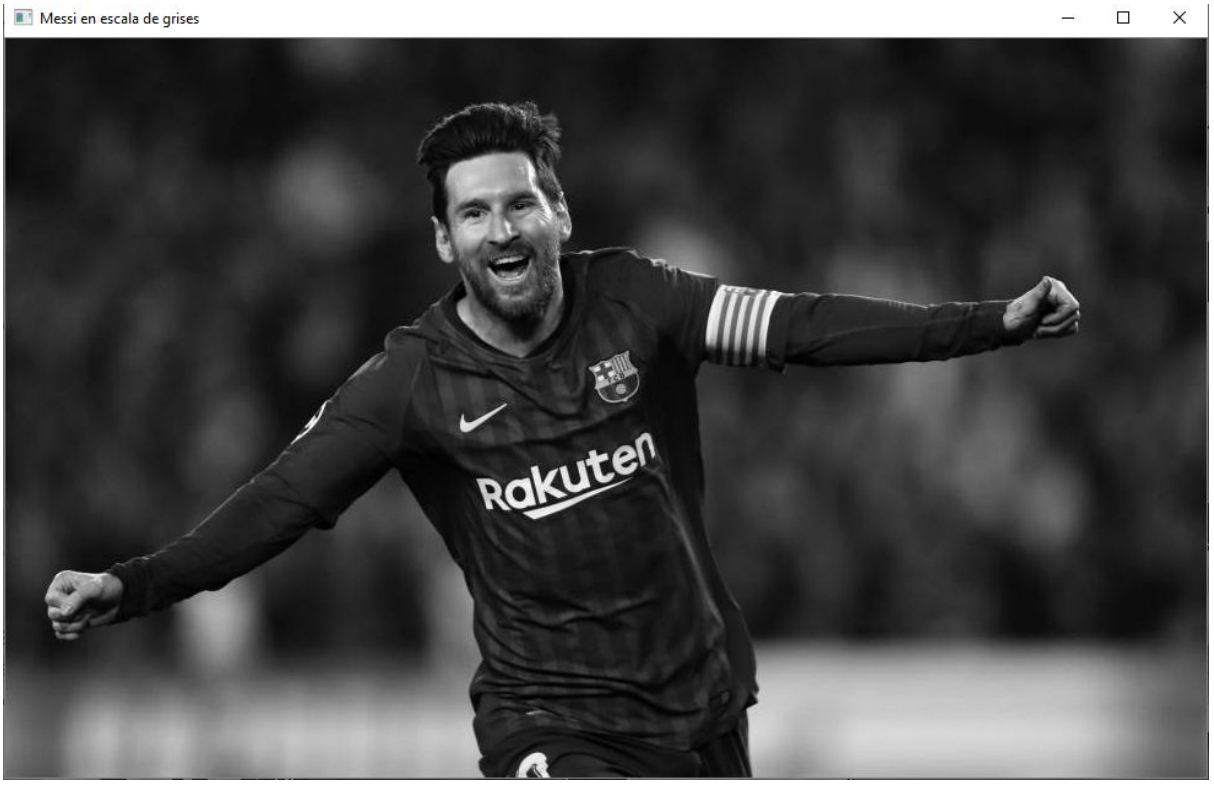
c)

La imagen que mejor representa la información que vemos es la del modelo **YIQ** ya que en este modelo la **I,Q** denotan la información de color , y la **Y** la luminancia , medida de la cantidad de energía que un observador percibe procedente de una fuente luminosa.
Por tanto si representamos **Y** , saldrá en escala de grises tal y como vemos la imagen con nuestros propios ojos pero sin color puesto que no representamos **I,Q** que es la ganancia de color.

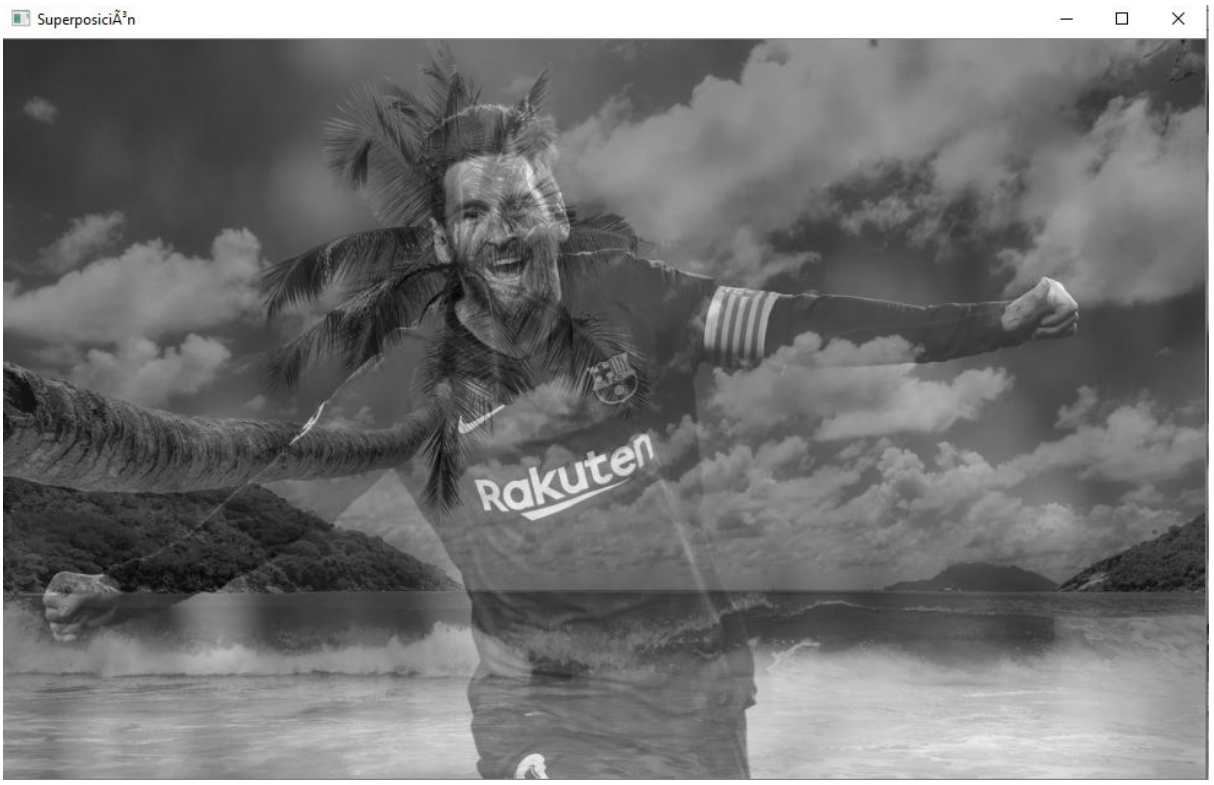
3)

Los valores realmente van entre 0 y 510 pero al sumar las matrices lo que hace es que cuando llega a 255 , empieza desde 0 hasta llegar a la diferencia del resultado de la suma - 255. Es decir si la suma es 310 , llegará hasta 255,empezará en 0 y llegará a 310-255= 55 , por tanto estamos perdiendo información.
Por tanto podemos truncar hasta 255 de una manera , cogiendo la mitad de escala de grises de cada pixel de una imagen y la de la otra y sumarla para obtener una imagen más clara en escala de grises, y luego sumar ambas matrices con el método ***add()*** de **numpy**, para así realizar la superposición. Aún así perdemos información pero menos que antes que se pasaba de 255. Y no se ha reescalado , sino que se ha truncado , como he explicado anteriormente.

Imágenes usadas:



Superposición:



Código implementado:

```
img2 = cv2.imread("playa.jpg", 0)

cv2.imshow("Messi en escala de grises", img)
cv2.imshow("Playa en escala de grises ", img2)

print(img.shape) #603x980
print(img2.shape) #678x1200

img2 = cv2.resize(img2,(980,603)) #redimensiono la imagen 2 , y la pongo mas pequeña para poder realizar la suma con la img1
img = img/2 #divido/2 para coger la mitad de escala de grises de los pixeles con la img2 , para sumarlas y que no pase de 255
img2 = img2/2
imgFinal = np.add(img,img2) #suma de las matrices de las imágenes
imgFinal = imgFinal.astype(np.uint8)
print(imgFinal)#comprobamos que no se pasa de 255
cv2.imshow("Superposición",imgFinal) #superposicion de las imágenes
cv2.waitKey(0)
```

Nota: Paso a uint8 la imgFinal para operar con números enteros , ya que con decimales no saldría bien la imagen.