Segmentación de imágenes usando RGB empleando el algoritmo MOG con ajuste de E.M.

A partir de una imagen RGB y proporcionando el valor de k (el número de gaussianas a mezclar). Obtener los parámetros de salida μ, ∑, λ con i=1, …, k gaussianas. Para generar la imagen de salida en cada pixel darle el color promedio μ, de la gaussiana mas cercana (la que de mayor probabilidad a ese color).

Selección de modelo optimo.

Segmentar la imagen desde k=2 hasta k=15, para cada k guardar la imagen de salida y encontrar la k optima K* usando el criterio de información AIC.

Para la realización de este trabajo se utilizaron dos metodologías diferentes para llevar a cabo la implementación del algoritmo MOG con ajuste de E.M., la primera fue programar todas las funciones y ecuaciones utilizadas para realizar el E.M., la segunda es el uso de la librería Scikitlearn que ya tiene implementada la función GMM, para finalmente comparar las imágenes resultantes de ambos métodos y ver qué resultados se pueden obtener.

Para programar las funciones, primero se obtienen 3 matrices del tamaño de la imagen, donde cada una representa un valor de la componente RGB, de ahí se generan μ, ∑ y λ aleatorios para cada k. Sin embargo, al sacar ∑ aleatoria se descubrió que algunas aleatorias no podían usarse porque son semidefinidas y su determinante es 0, dando error al querer calcular la normal con esa matriz de covarianza, para evitar eso se hace fueron guardando matrices aleatorias que no tuvieran ese problema y esas eran las que se utilizaron para entrenar. Posteriormente se empieza el algoritmo de E.M. para entrenar estos valores aleatorios, obteniendo las responsabilidades de cada k para cada pixel de la imagen y posteriormente actualizando los μ, ∑ y λ en cada iteración. Finalmente se actualiza la imagen, para esto se compara la responsabilidad de todas las k en cada pixel y se selecciona la que tiene mayor valor, y se cambia el color de ese pixel al de la media de la k escogida y se guarda la imagen resultante.

Para la segunda metodología se hace uso de la librería scikitlearn, la cual tiene implementado GMM en una función llamada GaussianMixture a la cual se le pasa un DataFrame 2D y el numero de K deseado y realiza la clasificación de cada pixel y obtiene las μ correspondientes, la limitante que se encontró con esta función es que solo acepta 2D, por lo que no se pueden usar valores RGB, para esto la imagen se convierte a escala de grises con ayuda de Opencv asi cada pixel paso de tener valores (R,G,B) entre 0 y 255 a tener solo un valor (grayscale) con valor en la misma escala de 0 a 255, para la segunda dimensión

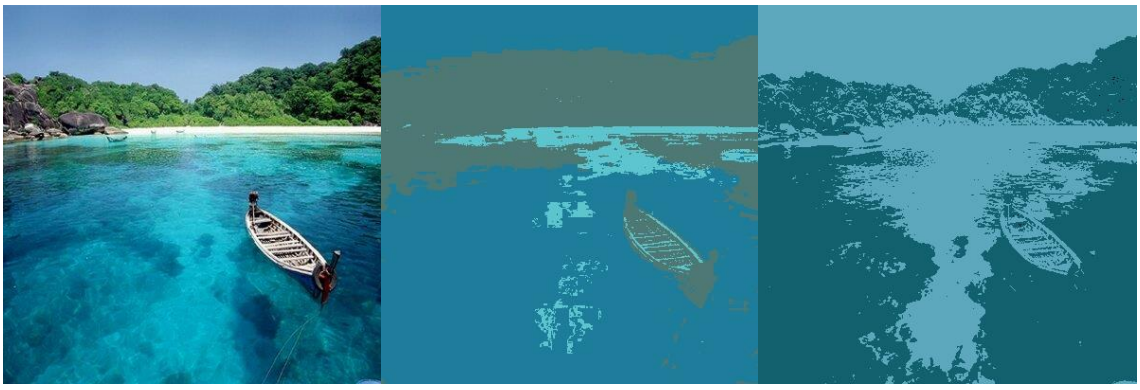
se pone el valor de 0, asiendo una coordenada (grayscale,0) y así se genera el DataFrame que será usado con la función GaussianMixture, la cual nos regresa una matriz donde cada índice representa un pixel y va a tener como valor la clase k a la que pertenece, con esta matriz se calcula en la imagen original los valores de las μ en RGB, para posteriormente cambiar el color de cada pixel al de la μ de su clase correspondiente. También la propia de función de GaussianMixture nos da el AIC y BIC para cada valor de K y así poder obtener las K optima.

A continuación, se presentan las imágenes obtenidas para cada valor de K con las dos metodologías implementadas:

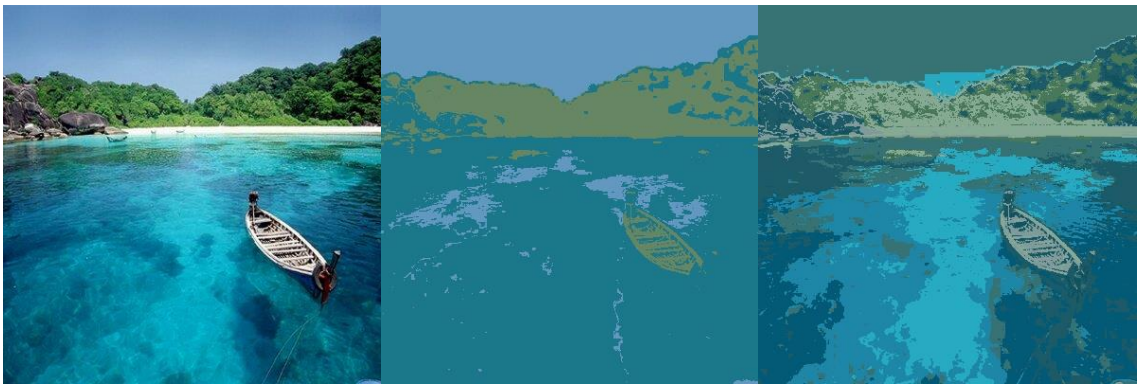
ORIGINAL EM Y GMM EN PYTHON SCIKITLEARN

K=2

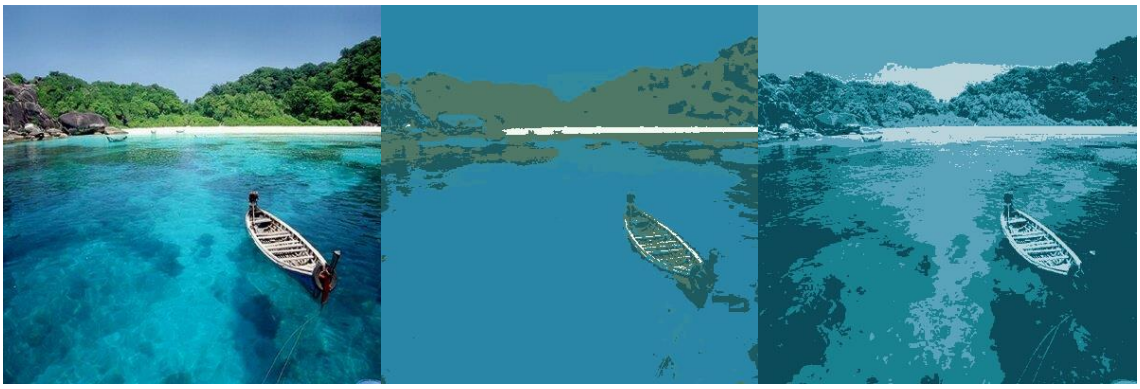
K=3



K=4



K=5



K=6

K=7

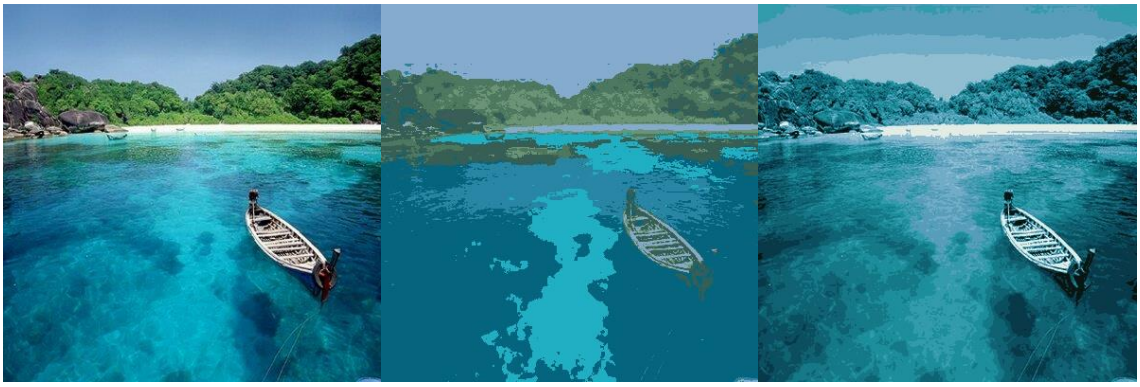
K=8

K=9

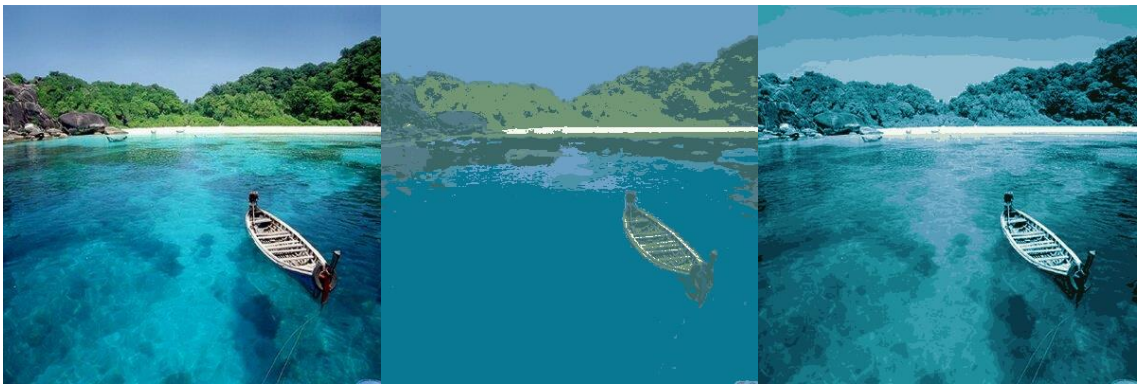
K=10

K=11

K=12

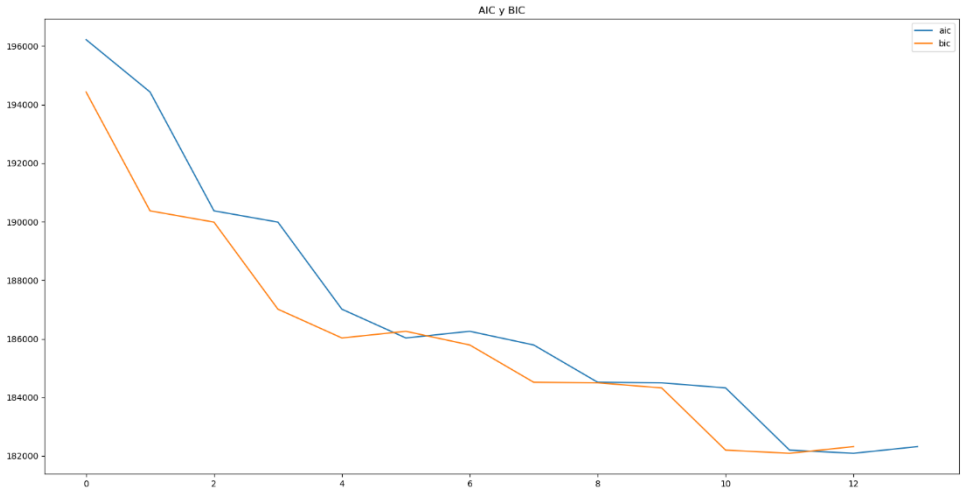


K=13



K=14

K=15



Conclusión:

El primer método resulto ser muy lento, por lo que no se incluyó el cálculo del AIC para cada k, y así no alentar más el procesamiento de la imagen, ya que para k=2 el tiempo de ejecución es de aproximadamente 3 minutos, pero para k=15 el tiempo rebasa los 40 minutos, siendo esta su mayor limitante. Mientras que la ventaja del segundo método es primero que nada el tiempo, el cual obtiene todas las imágenes de k=2 a k=15 en menos de 5 minutos, lo cual es una mejora muy significativa con respecto al otro, además de dar los valores de AIC y BIC para obtener la K optima sin necesidad de aumentar el tiempo de ejecución. Sin embargo, ambos métodos logran realizar de manera correcta la segmentación de imágenes que es lo que se buscaba obtener, aunque si se comparan, el modelo que no usa librerías es capaz de recuperar de mejor manera los colores de la imagen, lo cual se podría mejorar en el programa con scikitlearn encontrando la manera de trabajar con las tres dimensiones del RGB.

Código Python usando librería scikitlearn:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import numpy as np
import cv2

#Carga de imagen
imagen = cv2.imread('imagen1.jpg')
#cv2.imshow('imagen',imagen)
resolucionx=imagen.shape[1]
```

```
resoluciony=imagen.shape[0]

aic=np.array([[0]])
bic=np.array([[0]])

for k in range(2,16):
    imagen = cv2.imread('imagen1.jpg')
    gray=cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
    aux=np.asarray(gray)
    list=aux.reshape(1,resoluciony*resolucionx)
    list2=np.zeros((1,resoluciony*resolucionx))
    matrix=np.append(list,list2)
    data=pd.DataFrame(matrix)
    gmm=GaussianMixture(k,covariance_type='full',random_state=0).fit(data)
    medias=gmm.means_
    lmedias,cmedias=medias.shape
    label=gmm.predict(data)
    etiqueta=label.reshape(2,resoluciony*resolucionx)
    labels=np.delete(etiqueta,1,axis=0)
    mlabels=labels.reshape(resoluciony,resolucionx)

    mu=np.zeros((lmedias,3))
    cont=np.zeros((lmedias,1))
    for y in range(resoluciony):
            for x in range(resolucionx):
                for z in range(lmedias):
                    if mlabels[y][x]==z:
                        (b, g, r) = imagen[y, x]
                        mu[z][0]+=b
                        mu[z][1]+=g
                        mu[z][2]+=r
                        cont[z]+=1

    mus=mu/cont

    for y in range(resoluciony):
            for x in range(resolucionx):
                for z in range(lmedias):
                    if mlabels[y][x]==z:
                        imagen[y][x]=(mus[z][0],mus[z][1],mus[z][2])

    naic=np.array([[gmm.aic(data)]])
    aic=np.append(aic,naic,axis=0)
    nbic=np.array([[gmm.bic(data)]])
    bic=np.append(bic,nbic,axis=0)

    name='imagenn'+str(k)+'.jpg'
    print(name)
```

```
    cv2.imwrite(name,imagen)

aic=np.delete(aic,0,0)
bic=np.delete(aic,0,0)
plt.plot(aic,label='aic')
plt.title('AIC y BIC')
plt.plot(bic,label='bic')
plt.legend()
plt.show()
```

Código Python implementando EM y GMM sin uso de librería:

```
from email.base64mime import header_length
from email.mime import image
from re import M
from sys import maxunicode
import numpy as np
import cupy as cp
import cv2
import random
import math
from mpl_toolkits.mplot3d import Axes3D
import time
from scipy.special import expit
from scipy.stats import multivariate_normal
import numba
from numba import njit, prange
import numba_scipy
inicio=time.time()

#funcion que cambia color de imagen original
@njit(parallel=True)
def acimagen(imagen,mrik1,mrik2,mrik3,mu1,mu2,mu3):

    global resoluciony
    global resolucionx

    for y in prange(resoluciony):
            for x in prange(resolucionx):

                n1=mrik1[y][x]
                n2=mrik2[y][x]
                n3=mrik3[y][x]
                '''n4=mrik4[y][x]
                n5=mrik5[y][x]
                n6=mrik6[y][x]
                n7=mrik7[y][x]
```

```
                n8=mrik8[y][x]
                n9=mrik9[y][x]
                n10=mrik10[y][x]
                n11=mrik11[y][x]
                n12=mrik12[y][x]
                n13=mrik13[y][x]
                n14=mrik14[y][x]
                n15=mrik15[y][x]'''

                if n1>n2 and n1>n3:
                    imagen[y, x] = (mu1[2]*255, mu1[1]*255, mu1[0]*255)

                elif n2>n1 and n2>n3:
                    imagen[y, x] = (mu2[2]*255, mu2[1]*255, mu2[0]*255)

                elif n3>n1 and n3>n2:
                    imagen[y, x] = (mu3[2]*255, mu3[1]*255, mu3[0]*255)

                '''elif n4>n1 and n4>n2 and n4>n3 and n4>n5 and n4>n6 and n4>n7
and n4>n8 and n4>n9 and n4>n10 and n4>n11 and n4>n12 and n4>n13 and n4>n14 and
n4>n15:
                    imagen[y, x] = (mu4[2]*255, mu4[1]*255, mu4[0]*255)

                elif n5>n1 and n5>n2 and n5>n3 and n5>n4 and n5>n6 and n5>n7 and
n5>n8 and n5>n9 and n5>n10 and n5>n11 and n5>n12 and n5>n13 and n5>n14 and
n5>n15:
                    imagen[y, x] = (mu5[2]*255, mu5[1]*255, mu5[0]*255)

                elif n6>n1 and n6>n2 and n6>n3 and n6>n4 and n6>n5 and n6>n7 and
n6>n8 and n6>n9 and n6>n10 and n6>n11 and n6>n12 and n6>n13 and n6>n14 and
n6>n15:
                    imagen[y, x] = (mu6[2]*255, mu6[1]*255, mu6[0]*255)

                elif n7>n1 and n7>n2 and n7>n3 and n7>n4 and n7>n5 and n7>n6 and
n7>n8 and n7>n9 and n7>n10 and n7>n11 and n7>n12 and n7>n13 and n7>n14 and
n7>n15:
                    imagen[y, x] = (mu7[2]*255, mu7[1]*255, mu7[0]*255)

                elif n8>n1 and n8>n2 and n8>n3 and n8>n4 and n8>n5 and n8>n6 and
n8>n7 and n8>n9 and n8>n10 and n8>n11 and n8>n12 and n8>n13 and n8>n14 and
n8>n15:
                    imagen[y, x] = (mu8[2]*255, mu8[1]*255, mu8[0]*255)

                elif n9>n1 and n9>n2 and n9>n3 and n9>n4 and n9>n5 and n9>n6 and
n9>n7 and n9>n8 and n9>n10 and n9>n11 and n9>n12 and n9>n13 and n9>n14 and
n9>n15:
                    imagen[y, x] = (mu9[2]*255, mu9[1]*255, mu9[0]*255)
```

```
                elif n10>n1 and n10>n2 and n10>n3 and n10>n4 and n10>n5 and
n10>n6 and n10>n7 and n10>n8 and n10>n9 and n10>n11 and n10>n12 and n10>n13 and
n10>n14 and n10>n15:
                    imagen[y, x] = (mu10[2]*255, mu10[1]*255, mu10[0]*255)

                elif n11>n1 and n11>n2 and n11>n3 and n11>n4 and n11>n5 and
n11>n6 and n11>n7 and n11>n8 and n11>n9 and n11>n10 and n11>n12 and n11>n13 and
n11>n14 and n11>n15:
                    imagen[y, x] = (mu11[2]*255, mu11[1]*255, mu11[0]*255)

                elif n12>n1 and n12>n2 and n12>n3 and n12>n4 and n12>n5 and
n12>n6 and n12>n7 and n12>n8 and n12>n9 and n12>n10 and n12>n11 and n12>n13 and
n12>n14 and n12>n15:
                    imagen[y, x] = (mu12[2]*255, mu12[1]*255, mu12[0]*255)

                elif n13>n1 and n13>n2 and n13>n3 and n13>n4 and n13>n5 and
n13>n6 and n13>n7 and n13>n8 and n13>n9 and n13>n10 and n13>n11 and n13>n12 and
n13>n14 and n13>n15:
                    imagen[y, x] = (mu13[2]*255, mu13[1]*255, mu13[0]*255)

                elif n14>n1 and n14>n2 and n14>n3 and n14>n4 and n14>n5 and
n14>n6 and n14>n7 and n14>n8 and n14>n9 and n14>n10 and n14>n11 and n14>n12 and
n14>n13 and n14>n15:
                    imagen[y, x] = (mu13[2]*255, mu13[1]*255, mu13[0]*255)

                elif n15>n1 and n15>n2 and n15>n3 and n15>n4 and n15>n5 and
n15>n6 and n15>n7 and n15>n8 and n15>n9 and n15>n10 and n15>n11 and n15>n12 and
n15>n13 and n15>n14:
                    imagen[y, x] = (mu13[2]*255, mu13[1]*255, mu13[0]*255)'''

    return imagen

#funcion que extrae matriz de colores
@njit(parallel=True)
def colorr(v):

    global resoluciony
    global resolucionx

    for y in prange(resoluciony):
        for x in prange(resolucionx):
            (b, g, r) = imagen[y, x]
            v[y][x]=r/255

    return v

#funcion que extrae matriz de colores
@njit(parallel=True)
```

```
def colorg(v):

    global resoluciony
    global resolucionx

    for y in prange(resoluciony):
        for x in prange(resolucionx):
            (b, g, r) = imagen[y, x]
            v[y][x]=g/255

    return v

#funcion que extrae matriz de colores
@njit(parallel=True)
def colorb(v):

    global resoluciony
    global resolucionx

    for y in prange(resoluciony):
        for x in prange(resolucionx):
            (b, g, r) = imagen[y, x]
            v[y][x]=b/255

    return v

#Carga de imagen
imagen = cv2.imread('image1.jpg')
#cv2.imshow('imagen',imagen)
resolucionx=imagen.shape[1]
resoluciony=imagen.shape[0]

#Creacion de arrays r,g,b
vr=np.zeros((resoluciony,resolucionx))
vg=np.zeros((resoluciony,resolucionx))
vb=np.zeros((resoluciony,resolucionx))

vr=colorr(vr)
vg=colorg(vg)
vb=colorb(vb)

#creación de vector medio aleatorio y matriz de covarianza aleatoria y lambdas
mu1=np.array([random.random(), random.random(), random.random()])
mu2=np.array([random.random(), random.random(), random.random()])
mu3=np.array([random.random(), random.random(), random.random()])
'''mu4=np.array([random.random(), random.random(), random.random()])
mu5=np.array([random.random(), random.random(), random.random()])
mu6=np.array([random.random(), random.random(), random.random()])
```

```
mu7=np.array([random.random(), random.random(), random.random()])
mu8=np.array([random.random(), random.random(), random.random()])
mu9=np.array([random.random(), random.random(), random.random()])
mu10=np.array([random.random(), random.random(), random.random()])
mu11=np.array([random.random(), random.random(), random.random()])
mu12=np.array([random.random(), random.random(), random.random()])
mu13=np.array([random.random(), random.random(), random.random()])
mu14=np.array([random.random(), random.random(), random.random()])
mu15=np.array([random.random(), random.random(), random.random()])'''

lambda1=1/3+(random.randint(-10,10)/100)
lambda2=1/3+(random.randint(-10,10)/100)
'''lambda3=1/15+(random.randint(-1,1)/100)
lambda4=1/15+(random.randint(-1,1)/100)
lambda5=1/15+(random.randint(-1,1)/100)
lambda6=1/15+(random.randint(-1,1)/100)
lambda7=1/15+(random.randint(-1,1)/100)
lambda8=1/15+(random.randint(-1,1)/100)
lambda9=1/15+(random.randint(-1,1)/100)
lambda10=1/15+(random.randint(-1,1)/100)
lambda11=1/15+(random.randint(-1,1)/100)
lambda12=1/15+(random.randint(-1,1)/100)
lambda13=1/15+(random.randint(-1,1)/100)
lambda14=1/15+(random.randint(-1,1)/100)'''
lambda3=1-lambda1-lambda2

sigma1=np.array([
    [0.35390629, 0.09883398, 0.22479398],
    [0.09883398, 0.88912981, 0.64110369],
    [0.22479398, 0.64110369, 0.58122366],
])
sigma2=np.array([
    [0.93572708, 0.37072349, 0.17011359],
    [0.37072349, 0.87168998, 0.2964515 ],
    [0.17011359, 0.2964515,  0.51052869]
])
sigma3=np.array([
    [0.9500897, 0.29943573, 0.68852249],
    [0.29943573, 0.0980334, 0.24556276],
    [0.68852249, 0.24556276, 0.76094257]
])
'''sigma4=np.array([
    [0.81236315, 0.7184991,  0.46977035],
    [0.7184991,  0.81202962, 0.35962525],
    [0.46977035, 0.35962525, 0.910352  ]
])
sigma5=np.array([
    [0.91981405, 0.26085458, 0.06493828],
```

```
    [0.26085458, 0.69458422, 0.04014827],
    [0.06493828, 0.04014827, 0.56398507]
])
sigma6=np.array([
    [0.41690341, 0.12760576, 0.22878992],
    [0.12760576, 0.48307079, 0.59769438],
    [0.22878992, 0.59769438, 0.79346432]
])
sigma7=np.array([
    [0.9367735,  0.86801361, 0.20031797],
    [0.86801361, 0.88819186, 0.33937805],
    [0.20031797, 0.33937805, 0.57412754]
])
sigma8=np.array([
    [0.1321427,  0.28145001, 0.09547122],
    [0.28145001, 0.98235465, 0.14665779],
    [0.09547122, 0.14665779, 0.76489439]
])
sigma9=np.array([
    [0.94283664, 0.26213996, 0.28602818],
    [0.26213996, 0.75949994, 0.59983561],
    [0.28602818, 0.59983561, 0.78484585]
])
sigma10=np.array([
    [0.91470067, 0.24633126, 0.17247505],
    [0.24633126, 0.95370694, 0.52494829],
    [0.17247505, 0.52494829, 0.58601692]
])
sigma11=np.array([
    [0.26348295, 0.31790601, 0.17556696],
    [0.31790601, 0.4907126,  0.07685114],
    [0.17556696, 0.07685114, 0.85846835]
])
sigma12=np.array([
    [0.77197374, 0.11722366, 0.21972152],
    [0.11722366, 0.35455645, 0.09792693],
    [0.21972152, 0.09792693, 0.5809836 ]
])
sigma13=np.array([
    [0.3871499,  0.05695883, 0.0791909 ],
    [0.05695883, 0.85239819, 0.81380946],
    [0.0791909,  0.81380946, 0.83593968]
])
sigma14=np.array([
    [0.68145857, 0.04970968, 0.03388924],
    [0.04970968, 0.33219746, 0.07143189],
    [0.03388924, 0.07143189, 0.10066652]
])
```

```
sigma15=np.array([
    [0.8794686,  0.22298366, 0.4095341 ],
    [0.22298366, 0.36216902, 0.19958971],
    [0.4095341,  0.19958971, 0.71157573]
])'''

#prints
print(mu1)
print(mu2)
print(mu3)
'''print(mu4)
print(mu5)
print(mu6)
print(mu7)
print(mu8)
print(mu9)
print(mu10)
print(mu11)
print(mu12)
print(mu13)
print(mu14)
print(mu15)'''

print(lambda1)
print(lambda2)
print(lambda3)
'''print(lambda4)
print(lambda5)
print(lambda6)
print(lambda7)
print(lambda8)
print(lambda9)
print(lambda10)
print(lambda11)
print(lambda12)
print(lambda13)
print(lambda14)
print(lambda15)'''

print(sigma1)
print(sigma2)
print(sigma3)
'''print(sigma4)
print(sigma5)
print(sigma6)
print(sigma7)
print(sigma8)
print(sigma9)
```

```
print(sigma10)
print(sigma11)
print(sigma12)
print(sigma13)
print(sigma14)
print(sigma15)'''

#inicio de iteraciones
for i in range(5):

    mrik1=np.zeros((resoluciony,resolucionx))
    mrik2=np.zeros((resoluciony,resolucionx))
    mrik3=np.zeros((resoluciony,resolucionx))
    '''mrik4=np.zeros((resoluciony,resolucionx))
    mrik5=np.zeros((resoluciony,resolucionx))
    mrik6=np.zeros((resoluciony,resolucionx))
    mrik7=np.zeros((resoluciony,resolucionx))
    mrik8=np.zeros((resoluciony,resolucionx))
    mrik9=np.zeros((resoluciony,resolucionx))
    mrik10=np.zeros((resoluciony,resolucionx))
    mrik11=np.zeros((resoluciony,resolucionx))
    mrik12=np.zeros((resoluciony,resolucionx))
    mrik13=np.zeros((resoluciony,resolucionx))
    mrik14=np.zeros((resoluciony,resolucionx))
    mrik15=np.zeros((resoluciony,resolucionx))'''

    sumrik1=0
    sumrik2=0
    sumrik3=0
    '''sumrik4=0
    sumrik5=0
    sumrik6=0
    sumrik7=0
    sumrik8=0
    sumrik9=0
    sumrik10=0
    sumrik11=0
    sumrik12=0
    sumrik13=0
    sumrik14=0
    sumrik15=0'''

    sumriks=0

    sumrik1x=0
    sumrik2x=0
    sumrik3x=0
    '''sumrik4x=0
```

```
sumrik5x=0
sumrik6x=0
sumrik7x=0
sumrik8x=0
sumrik9x=0
sumrik10x=0
sumrik11x=0
sumrik12x=0
sumrik13x=0
sumrik14x=0
sumrik15x=0'''

sumrik1xr=0
sumrik1xg=0
sumrik1xb=0

sumrik2xr=0
sumrik2xg=0
sumrik2xb=0

sumrik3xr=0
sumrik3xg=0
sumrik3xb=0

'''sumrik4xr=0
sumrik4xg=0
sumrik4xb=0

sumrik5xr=0
sumrik5xg=0
sumrik5xb=0

sumrik6xr=0
sumrik6xg=0
sumrik6xb=0

sumrik7xr=0
sumrik7xg=0
sumrik7xb=0

sumrik8xr=0
sumrik8xg=0
sumrik8xb=0

sumrik9xr=0
sumrik9xg=0
sumrik9xb=0
```

```
sumrik10xr=0
sumrik10xg=0
sumrik10xb=0

sumrik11xr=0
sumrik11xg=0
sumrik11xb=0

sumrik12xr=0
sumrik12xg=0
sumrik12xb=0

sumrik13xr=0
sumrik13xg=0
sumrik13xb=0

sumrik14xr=0
sumrik14xg=0
sumrik14xb=0

sumrik15xr=0
sumrik15xg=0
sumrik15xb=0'''

#calculo de rik
for y in range(resoluciony):
    for x in range(resolucionx):
        rgb=np.array([vr[y][x],vg[y][x],vb[y][x]])
        n1=lambda1*multivariate_normal.pdf(rgb,mu1,sigma1)
        n2=lambda2*multivariate_normal.pdf(rgb,mu2,sigma2)
        n3=lambda3*multivariate_normal.pdf(rgb,mu3,sigma3)
        '''n4=lambda4*multivariate_normal.pdf(rgb,mu4,sigma4)
        n5=lambda5*multivariate_normal.pdf(rgb,mu5,sigma5)
        n6=lambda6*multivariate_normal.pdf(rgb,mu6,sigma6)
        n7=lambda7*multivariate_normal.pdf(rgb,mu7,sigma7)
        n8=lambda8*multivariate_normal.pdf(rgb,mu8,sigma8)
        n9=lambda9*multivariate_normal.pdf(rgb,mu9,sigma9)
        n10=lambda10*multivariate_normal.pdf(rgb,mu10,sigma10)
        n11=lambda11*multivariate_normal.pdf(rgb,mu11,sigma11)
        n12=lambda12*multivariate_normal.pdf(rgb,mu12,sigma12)
        n13=lambda13*multivariate_normal.pdf(rgb,mu13,sigma13)
        n14=lambda14*multivariate_normal.pdf(rgb,mu14,sigma14)
        n15=lambda15*multivariate_normal.pdf(rgb,mu15,sigma15)'''

        sumn=n1+n2+n3

        rik1=n1/sumn
        rik2=n2/sumn
```

```
rik3=n3/sumn
'''rik4=n4/sumn
rik5=n5/sumn
rik6=n6/sumn
rik7=n7/sumn
rik8=n8/sumn
rik9=n9/sumn
rik10=n10/sumn
rik11=n11/sumn
rik12=n12/sumn
rik13=n13/sumn
rik14=n14/sumn
rik15=n15/sumn'''

mrik1[y][x]=rik1
mrik2[y][x]=rik2
mrik3[y][x]=rik3
'''mrik4[y][x]=rik4
mrik5[y][x]=rik5
mrik6[y][x]=rik6
mrik7[y][x]=rik7
mrik8[y][x]=rik8
mrik9[y][x]=rik9
mrik10[y][x]=rik10
mrik11[y][x]=rik11
mrik12[y][x]=rik12
mrik13[y][x]=rik13
mrik14[y][x]=rik14
mrik15[y][x]=rik15'''

sumrik1+=rik1
sumrik2+=rik2
sumrik3+=rik3
'''sumrik4+=rik4
sumrik5+=rik5
sumrik6+=rik6
sumrik7+=rik7
sumrik8+=rik8
sumrik9+=rik9
sumrik10+=rik10
sumrik11+=rik11
sumrik12+=rik12
sumrik13+=rik13
sumrik14+=rik14
sumrik15+=rik15'''

sumriks+=rik1+rik2+rik3
```

```
                sumrik1xr+=(rik1*(vr[y][x]))
                sumrik1xg+=(rik1*(vg[y][x]))
                sumrik1xb+=(rik1*(vb[y][x]))

                sumrik2xr+=(rik2*(vr[y][x]))
                sumrik2xg+=(rik2*(vg[y][x]))
                sumrik2xb+=(rik2*(vb[y][x]))

                sumrik3xr+=(rik3*(vr[y][x]))
                sumrik3xg+=(rik3*(vg[y][x]))
                sumrik3xb+=(rik3*(vb[y][x]))

                '''sumrik4xr+=(rik4*(vr[y][x]))
                sumrik4xg+=(rik4*(vg[y][x]))
                sumrik4xb+=(rik4*(vb[y][x]))

                sumrik5xr+=(rik5*(vr[y][x]))
                sumrik5xg+=(rik5*(vg[y][x]))
                sumrik5xb+=(rik5*(vb[y][x]))

                sumrik6xr+=(rik6*(vr[y][x]))
                sumrik6xg+=(rik6*(vg[y][x]))
                sumrik6xb+=(rik6*(vb[y][x]))

                sumrik7xr+=(rik7*(vr[y][x]))
                sumrik7xg+=(rik7*(vg[y][x]))
                sumrik7xb+=(rik7*(vb[y][x]))

                sumrik8xr+=(rik8*(vr[y][x]))
                sumrik8xg+=(rik8*(vg[y][x]))
                sumrik8xb+=(rik8*(vb[y][x]))

                sumrik9xr+=(rik9*(vr[y][x]))
                sumrik9xg+=(rik9*(vg[y][x]))
                sumrik9xb+=(rik9*(vb[y][x]))

                sumrik10xr+=(rik10*(vr[y][x]))
                sumrik10xg+=(rik10*(vg[y][x]))
                sumrik10xb+=(rik10*(vb[y][x]))

                sumrik11xr+=(rik11*(vr[y][x]))
                sumrik11xg+=(rik11*(vg[y][x]))
                sumrik11xb+=(rik11*(vb[y][x]))

                sumrik12xr+=(rik12*(vr[y][x]))
                sumrik12xg+=(rik12*(vg[y][x]))
                sumrik12xb+=(rik12*(vb[y][x]))
```

```
            sumrik13xr+=(rik13*(vr[y][x]))
            sumrik13xg+=(rik13*(vg[y][x]))
            sumrik13xb+=(rik13*(vb[y][x]))

            sumrik14xr+=(rik14*(vr[y][x]))
            sumrik14xg+=(rik14*(vg[y][x]))
            sumrik14xb+=(rik14*(vb[y][x]))

            sumrik15xr+=(rik15*(vr[y][x]))
            sumrik15xg+=(rik15*(vg[y][x]))
            sumrik15xb+=(rik15*(vb[y][x]))'''

    #calculo de lambdas
    lambda1=sumrik1/sumriks
    lambda2=sumrik2/sumriks
    lambda3=sumrik3/sumriks
    '''lambda4=sumrik4/sumriks
    lambda5=sumrik5/sumriks
    lambda6=sumrik6/sumriks
    lambda7=sumrik7/sumriks
    lambda8=sumrik8/sumriks
    lambda9=sumrik9/sumriks
    lambda10=sumrik10/sumriks
    lambda11=sumrik11/sumriks
    lambda12=sumrik12/sumriks
    lambda13=sumrik13/sumriks
    lambda14=sumrik14/sumriks
    lambda15=sumrik15/sumriks'''

    #calculo de mus
    mu1[0]=sumrik1xr/sumrik1
    mu1[1]=sumrik1xg/sumrik1
    mu1[2]=sumrik1xb/sumrik1

    mu2[0]=sumrik2xr/sumrik2
    mu2[1]=sumrik2xg/sumrik2
    mu2[2]=sumrik2xb/sumrik2

    mu3[0]=sumrik3xr/sumrik3
    mu3[1]=sumrik3xg/sumrik3
    mu3[2]=sumrik3xb/sumrik3

    '''mu4[0]=sumrik4xr/sumrik4
    mu4[1]=sumrik4xg/sumrik4
    mu4[2]=sumrik4xb/sumrik4

    mu5[0]=sumrik5xr/sumrik5
    mu5[1]=sumrik5xg/sumrik5
```

```
mu5[2]=sumrik5xb/sumrik5

mu6[0]=sumrik6xr/sumrik6
mu6[1]=sumrik6xg/sumrik6
mu6[2]=sumrik6xb/sumrik6

mu7[0]=sumrik7xr/sumrik7
mu7[1]=sumrik7xg/sumrik7
mu7[2]=sumrik7xb/sumrik7

mu8[0]=sumrik8xr/sumrik8
mu8[1]=sumrik8xg/sumrik8
mu8[2]=sumrik8xb/sumrik8

mu9[0]=sumrik9xr/sumrik9
mu9[1]=sumrik9xg/sumrik9
mu9[2]=sumrik9xb/sumrik9

mu10[0]=sumrik10xr/sumrik10
mu10[1]=sumrik10xg/sumrik10
mu10[2]=sumrik10xb/sumrik10

mu11[0]=sumrik11xr/sumrik11
mu11[1]=sumrik11xg/sumrik11
mu11[2]=sumrik11xb/sumrik11

mu12[0]=sumrik12xr/sumrik12
mu12[1]=sumrik12xg/sumrik12
mu12[2]=sumrik12xb/sumrik12

mu13[0]=sumrik13xr/sumrik13
mu13[1]=sumrik13xg/sumrik13
mu13[2]=sumrik13xb/sumrik13

mu14[0]=sumrik14xr/sumrik14
mu14[1]=sumrik14xg/sumrik14
mu14[2]=sumrik14xb/sumrik14

mu15[0]=sumrik15xr/sumrik15
mu15[1]=sumrik15xg/sumrik15
mu15[2]=sumrik15xb/sumrik15'''

sumrik1xmu1=0
sumrik2xmu2=0
sumrik3xmu3=0
'''sumrik4xmu4=0
sumrik5xmu5=0
sumrik6xmu6=0
```

```
sumrik7xmu7=0
sumrik8xmu8=0
sumrik9xmu9=0
sumrik10xmu10=0
sumrik11xmu11=0
sumrik12xmu12=0
sumrik13xmu13=0
sumrik14xmu14=0
sumrik15xmu15=0'''

#calculo de sigmas
for y in range(resoluciony):
    for x in range(resolucionx):
        rgb=np.array([vr[y][x],vg[y][x],vb[y][x]])
        rgb_mu1=rgb-mu1
        rgb_mu2=rgb-mu2
        rgb_mu3=rgb-mu3
        '''rgb_mu4=rgb-mu4
        rgb_mu5=rgb-mu5
        rgb_mu6=rgb-mu6
        rgb_mu7=rgb-mu7
        rgb_mu8=rgb-mu8
        rgb_mu9=rgb-mu9
        rgb_mu10=rgb-mu10
        rgb_mu11=rgb-mu11
        rgb_mu12=rgb-mu12
        rgb_mu13=rgb-mu13
        rgb_mu14=rgb-mu14
        rgb_mu15=rgb-mu15'''

        rgb_mu1t=np.transpose(rgb_mu1.reshape(1,3))
        rgb_mu2t=np.transpose(rgb_mu2.reshape(1,3))
        rgb_mu3t=np.transpose(rgb_mu3.reshape(1,3))
        '''rgb_mu4t=np.transpose(rgb_mu4.reshape(1,3))
        rgb_mu5t=np.transpose(rgb_mu5.reshape(1,3))
        rgb_mu6t=np.transpose(rgb_mu6.reshape(1,3))
        rgb_mu7t=np.transpose(rgb_mu7.reshape(1,3))
        rgb_mu8t=np.transpose(rgb_mu8.reshape(1,3))
        rgb_mu9t=np.transpose(rgb_mu9.reshape(1,3))
        rgb_mu10t=np.transpose(rgb_mu10.reshape(1,3))
        rgb_mu11t=np.transpose(rgb_mu11.reshape(1,3))
        rgb_mu12t=np.transpose(rgb_mu12.reshape(1,3))
        rgb_mu13t=np.transpose(rgb_mu13.reshape(1,3))
        rgb_mu14t=np.transpose(rgb_mu14.reshape(1,3))
        rgb_mu15t=np.transpose(rgb_mu15.reshape(1,3))'''

        rik1xmu1=mrik1[y][x]*rgb_mu1*rgb_mu1t
        rik2xmu2=mrik2[y][x]*rgb_mu2*rgb_mu2t
```

```
            rik3xmu3=mrik3[y][x]*rgb_mu3*rgb_mu3t
            '''rik4xmu4=mrik4[y][x]*rgb_mu4*rgb_mu4t
            rik5xmu5=mrik5[y][x]*rgb_mu5*rgb_mu5t
            rik6xmu6=mrik6[y][x]*rgb_mu6*rgb_mu6t
            rik7xmu7=mrik7[y][x]*rgb_mu7*rgb_mu7t
            rik8xmu8=mrik8[y][x]*rgb_mu8*rgb_mu8t
            rik9xmu9=mrik9[y][x]*rgb_mu9*rgb_mu9t
            rik10xmu10=mrik10[y][x]*rgb_mu10*rgb_mu10t
            rik11xmu11=mrik11[y][x]*rgb_mu11*rgb_mu11t
            rik12xmu12=mrik12[y][x]*rgb_mu12*rgb_mu12t
            rik13xmu13=mrik13[y][x]*rgb_mu13*rgb_mu13t
            rik14xmu14=mrik14[y][x]*rgb_mu14*rgb_mu14t
            rik15xmu15=mrik15[y][x]*rgb_mu15*rgb_mu15t'''

            sumrik1xmu1+=rik1xmu1
            sumrik2xmu2+=rik2xmu2
            sumrik3xmu3+=rik3xmu3
            '''sumrik4xmu4+=rik4xmu4
            sumrik5xmu5+=rik5xmu5
            sumrik6xmu6+=rik6xmu6
            sumrik7xmu7+=rik7xmu7
            sumrik8xmu8+=rik8xmu8
            sumrik9xmu9+=rik9xmu9
            sumrik10xmu10+=rik10xmu10
            sumrik11xmu11+=rik11xmu11
            sumrik12xmu12+=rik12xmu12
            sumrik13xmu13+=rik13xmu13
            sumrik14xmu14+=rik14xmu14
            sumrik15xmu15+=rik15xmu15'''

    sigma1=sumrik1xmu1/sumrik1
    sigma2=sumrik2xmu2/sumrik2
    sigma3=sumrik3xmu3/sumrik3
    '''sigma4=sumrik4xmu4/sumrik4
    sigma5=sumrik5xmu5/sumrik5
    sigma6=sumrik6xmu6/sumrik6
    sigma7=sumrik7xmu7/sumrik7
    sigma8=sumrik8xmu8/sumrik8
    sigma9=sumrik9xmu9/sumrik9
    sigma10=sumrik10xmu10/sumrik10
    sigma11=sumrik11xmu11/sumrik11
    sigma12=sumrik12xmu12/sumrik12
    sigma13=sumrik13xmu13/sumrik13
    sigma14=sumrik14xmu14/sumrik14
    sigma15=sumrik15xmu15/sumrik15'''

#prints
print("Datos nuevos")
```

```
print(mu1)
print(mu2)
print(mu3)
'''print(mu4)
print(mu5)
print(mu6)
print(mu7)
print(mu8)
print(mu9)
print(mu10)
print(mu11)
print(mu12)
print(mu13)
print(mu14)
print(mu15)'''

print(lambda1)
print(lambda2)
print(lambda3)
'''print(lambda4)
print(lambda5)
print(lambda6)
print(lambda7)
print(lambda8)
print(lambda9)
print(lambda10)
print(lambda11)
print(lambda12)
print(lambda13)
print(lambda14)
print(lambda15)'''

print(sigma1)
print(sigma2)
print(sigma3)
'''print(sigma4)
print(sigma5)
print(sigma6)
print(sigma7)
print(sigma8)
print(sigma9)
print(sigma10)
print(sigma11)
print(sigma12)
print(sigma13)
print(sigma14)
print(sigma15)'''
```

```
acimagen(imagen,mrik1,mrik2,mrik3,mu1,mu2,mu3)

fin=time.time()
print("El tiempo de ejecución es:")
print(fin-inicio)
cv2.namedWindow('imagen')
cv2.imwrite('image3.jpg',imagen)
cv2.imshow('imagen',imagen)
cv2.waitKey(0)
```