INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Visión Artificial

Práctica 2

’Cálculo de centro de masa’

Profesor:

Anzueto Rios Álvaro

Grupo:

3BM5

Presenta:

Aguilar Gómez Pedro Nicolás

Sánchez Sánchez Luis Alberto

Fecha

29 de marzo del 2021

Contenido

Marco teórico

Algoritmo de elementos conectados

Un componente conectado o simplemente un componente de un gráfico no dirigido es un subgráfico en el que cada par de nodos está conectado entre sí a través de una ruta.

Un conjunto de nodos forma un componente conectado en un gráfico no dirigido si cualquier nodo del conjunto de nodos puede alcanzar cualquier otro nodo atravesando bordes. El punto principal aquí es la accesibilidad. En los componentes conectados, todos los nodos son siempre accesibles entre sí.

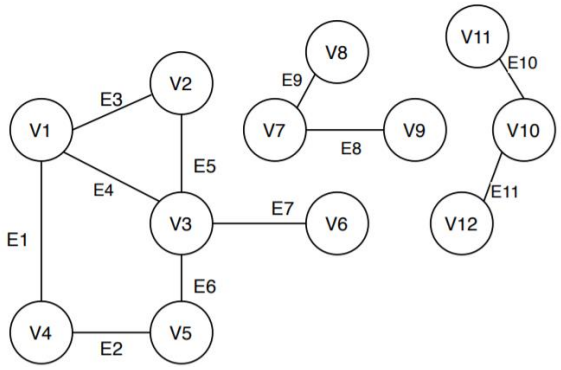

Ilustración 1 Representación de elementos conectados

Algoritmo de código cadena

Es un algoritmo de compresión con pérdida para imágenes monocromáticas. El principio básico de chain code es codificar por separado cada componente conectado, o "blot", en la imagen. Para cada región, se selecciona un punto de su límite y se transmiten sus coordenadas. El codificador se mueve a lo largo del límite de la imagen y, en cada paso, transmite un símbolo representando la dirección de su movimiento. Esto continua hasta que el codificador retorna a la posición inicial, punto en el que el blot ha sido descrito completamente, y la codificación continua con el siguiente blot de la imagen.

Este método de codificación es particularmente efectivo para imágenes consistentes en un número razonablemente alto de componentes conectados.

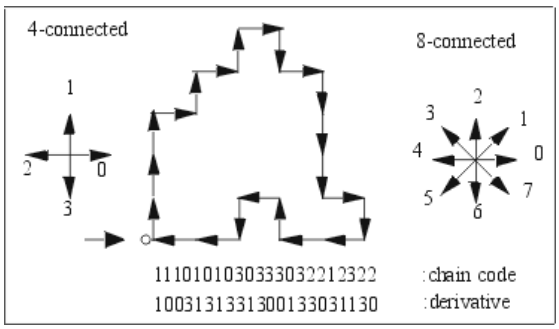

Ilustración 2 Representación del código cadena

Desarrollo

El programa se encuentra dividido en su ejecución en tres secciones. La primera es el clasificador por color que utiliza el modelo de color CIE-LAB con un segmentado por k-means que posteriormente es clasificado por el algoritmo de labeling. La segunda parte es un clasificado por figura que utiliza el código cadena para segmentar bordes y a partir de ahí identificar la figura deseada, el algoritmo de labeling cumple con la función de contar cuantas figuras hay en la imagen. Finalmente, la última sección consta de un clasificador por color y forma, el cual utiliza en conjunto el código de las secciones anteriores.

```
from skimage import data, color, io
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import cv2
from skimage.color import rgb2gray
from dtaidistance import dtw
plt.close('all')
def                                                                                    labelling(ima):
##########################################################################################
##

  ai=np.zeros((ima.shape[0]+2,ima.shape[1]+2)) #se añaden dos bordes a la imagen para no
                                  #tener tantas condicionales de frontera
  for i in range(ima.shape[0]):
      ai[i+1,1:ai.shape[1]-1]=ima[i,:]

  # se realiza el primer ciclo
  sal1=np.zeros(ai.shape)
  c=2

  for i in range (1,ai.shape[0]-1,1):
    for j in range (1,ai.shape[1]-1,1):
      if ai[i,j]==1:
        vec=np.array([sal1[i-1,j-1],sal1[i-1,j],sal1[i-1,j+1],  #aqui preguntamos por los 8 vecinos
                  sal1[i  ,j-1],          sal1[i  ,j+1],  #del pixel para asignar etiqueta
                  sal1[i+1,j-1],sal1[i+1,j],sal1[i+1,j+1]])
        vec=np.where(vec<=0,1000,vec) #se sustituyen los ceros por un valor muy grande
diferente
        #print(vec)
        mn=np.min(vec) #se saca la etiqueta minima de los vecinos si es que hay
        if mn ==1000:
          sal1[i,j]=c #si el minimo es igual 100 se asigna una nueva etiqueta
        else:
          sal1[i,j]=mn #si un vecino ya tiene etiqueta, se asigna la menor
      else:
        if ai[i,j-1]!=0: #condicion que permite solo un incremento
```

```
            c=c+1

   #se realiza el segundo ciclo

   sal2=np.zeros(ai.shape)
   vp=[]
   for i in range (1,ai.shape[0]-1,1):
      for j in range (1,ai.shape[1]-1,1):
         if sal1[i,j]!=0:   #funcion solo aplicamos a los que sean diferente de cero
            vec=np.array([sal1[i-1,j-1],sal1[i-1,j],sal1[i-1,j+1],  #aqui preguntamos por los 8 vecinos
                     sal1[i  ,j-1],          sal1[i  ,j+1],  #del pixel para asignar etiqueta
                     sal1[i+1,j-1],sal1[i+1,j],sal1[i+1,j+1]])
            for k in range(vec.shape[0]): #preguntamos para cada vecino
               if sal1[i,j]!=vec[k] and vec[k]!=0: # si la etiqueta del vecino no es 0 y es diferente
                  sal1=np.where(sal1==vec[k],sal1[i,j],sal1) #se sustituyen las etiquetas los vecinos
que coincidan

   for i in range (1,ai.shape[0]-1,1):
      for j in range (1,ai.shape[1]-1,1):
         if sal1[i,j]!=0:
            if sal1[i,j] not in vp:
               vp.append(sal1[i,j])
   vp=np.array(vp)
   sal1=sal1[1:sal1.shape[0]-1,1:sal1.shape[1]-1]
   #print('sal'+str(vp.shape[0]))

   return sal1,vp
##########################################################################################
###################################

def                                                                                  chainc
(Ima):#################################################################################
###################
   bordes = np.zeros((Ima.shape[0], Ima.shape[1]))

   # Buscar contornos
   for i in range(1, Ima.shape[0] - 1, 1):
      for j in range(1, Ima.shape[1] - 1, 1):
         if Ima[i,j] == 1 and (Ima[i-1,j] == 0 or Ima[i,j-1] == 0 or Ima[i,j+1] == 0 or Ima[i+1,j] == 0):
            # Para que el ruido no lo considere como bordes
            if Ima[i-1,j] == 0 and Ima[i,j-1] == 0 and Ima[i,j+1] == 0 and Ima[i+1,j] == 0:
               bordes[i,j] = 0
            else:
               bordes[i,j] = 1
   # plt.figure()
   # plt.title('Bordes')
   # plt.imshow(bordes, cmap = 'gray')
```

```
#  Variable auxiliar
bordes2 = bordes
coord_bordes = []

# Contador de objetos
obj = 1

for i in range(bordes2.shape[0]):
   for j in range(bordes2.shape[1]):
      # Encontrar el primer objeto
      if bordes2[i,j] == 1:
         # Variables para no modificar el recorrido normal de la imagen, para el
         # subprograma
         fil = i
         col = j
         # Registrar posiciones
         pixel = 0
         # Hacer cero la posición anterior
         bordes2[fil,col] = 0

         #Ciclo para preguntar por los siguientes pixeles
         final = 1
         while final == 1:
            if bordes2[fil,col+1] == 1:
               coord_bordes.append(np.array([fil,col,0,obj]))
               pixel += 1
               bordes2[fil,col] = 0
               fil = fil
               col = col + 1
            elif bordes2[fil+1,col+1] == 1:
               coord_bordes.append(np.array([fil,col,1,obj]))
               pixel += 1
               bordes2[fil,col] = 0
               fil = fil + 1
               col = col + 1
            elif bordes2[fil+1,col] == 1:
               coord_bordes.append(np.array([fil,col,2,obj]))
               pixel += 1
               bordes2[fil,col] = 0
               fil = fil + 1
               col = col
            elif bordes2[fil+1,col-1] == 1:
               coord_bordes.append(np.array([fil,col,3,obj]))
               pixel += 1
               bordes2[fil,col] = 0
               fil = fil + 1
               col = col - 1
            elif bordes2[fil,col-1] == 1:
```

```
                coord_bordes.append(np.array([fil,col,4,obj]))
                pixel += 1
                bordes2[fil,col] = 0
                fil = fil
                col = col - 1
            elif bordes2[fil-1,col-1] == 1:
                coord_bordes.append(np.array([fil,col,5,obj]))
                pixel += 1
                bordes2[fil,col] = 0
                fil = fil - 1
                col = col - 1
            elif bordes2[fil-1,col] == 1:
                coord_bordes.append(np.array([fil,col,6,obj]))
                pixel += 1
                bordes2[fil,col] = 0
                fil = fil - 1
                col = col
            elif bordes2[fil-1,col+1] == 1:
                coord_bordes.append(np.array([fil,col,7,obj]))
                pixel += 1
                bordes2[fil,col] = 0
                fil = fil - 1
                col = col + 1
            else:
                bordes2[fil,col] = 0
                pixel += 1
                final = 0
                coord_bordes.append(np.array([fil,col,0,obj]))
                obj += 1

# Calcular centroide de la imágen
suma_fil = 0
suma_col = 0

for i in range(pixel):

    suma_fil += coord_bordes[i][0]

    suma_col += coord_bordes[i][1]

cx = suma_col / pixel
cy = suma_fil / pixel

# Calcular distancia euclidiana del centroide al borde
firma = []
for i in range(pixel):
    d = np.sqrt((cx - coord_bordes[i][1])**2 + (cy - coord_bordes[i][0])**2)
    firma.append(d)
```

```
    firma=np.array(firma/np.max(firma))
    return firma

def extr(ent,vp,nf):

    ext=np.where(ent==vp[nf],1,0) #extraccion de la figura deseada

    return ext
def excol(name):
    ima=io.imread(name)
    plt.figure(0)
    plt.title('Da Click el color que deseas aislar y espera...')
    plt.imshow(ima)
    co=np.int32(plt.ginput(1))

    #--------------------Modelo de color RGB Kmeans--------------------
    ima33=io.imread(name)

    ima33=io.imread(name)
    ima3=color.rgb2lab(ima33)


    l=(ima3[:,:,0]) #convierte la matriz en un vector
    a=(ima3[:,:,1])
    b=(ima3[:,:,2])
    L=l.reshape((-1,1))
    A=a.reshape((-1,1))
    B=b.reshape((-1,1))
    datos3=np.concatenate((L,A,B),axis=1)
    clases=4
    salida3=KMeans(n_clusters=clases)
    salida3.fit(datos3)

    centros3=salida3.cluster_centers_
    aa2=color.lab2rgb(centros3[np.newaxis,:])
    etiquetas3=salida3.labels_ #volver a reconstruir como imagen

    for i in range (L.shape[0]): #asignar un color a cada posicion segun la etiqueta
        L[i]=aa2[0][etiquetas3[i]][0]
        A[i]=aa2[0][etiquetas3[i]][1]
        B[i]=aa2[0][etiquetas3[i]][2]

    L.shape=l.shape #redimencionar un vector a matriz
    A.shape=a.shape
    B.shape=b.shape

    L=L[:,:,np.newaxis]
    A=A[:,:,np.newaxis]
```

```
    B=B[:,:,np.newaxis]

    new3=np.concatenate((L,A,B),axis=2)
    gris = rgb2gray(new3)
    bina = np.where(gris == gris[co[0][1],co[0][0]],1,0)

    return bina
def excol1(name):
    ima=io.imread(name)
    plt.figure(0)
    plt.title('Introduce la forma que deseas aislar\n una vez que aparezca la opcion en la consola')
    plt.imshow(ima)
    plt.pause(1)
    #--------------------Modelo de color RGB Kmeans--------------------
    ima33=io.imread(name)

    ima33=io.imread(name)
    ima3=color.rgb2lab(ima33)

    l=(ima3[:,:,0]) #convierte la matriz en un vector
    a=(ima3[:,:,1])
    b=(ima3[:,:,2])
    L=l.reshape((-1,1))
    A=a.reshape((-1,1))
    B=b.reshape((-1,1))
    datos3=np.concatenate((L,A,B),axis=1)
    clases=4
    salida3=KMeans(n_clusters=clases)
    salida3.fit(datos3)

    centros3=salida3.cluster_centers_
    aa2=color.lab2rgb(centros3[np.newaxis,:])
    etiquetas3=salida3.labels_ #volver a reconstruir como imagen

    for i in range (L.shape[0]): #asignar un color a cada posicion segun la etiqueta
        L[i]=aa2[0][etiquetas3[i]][0]
        A[i]=aa2[0][etiquetas3[i]][1]
        B[i]=aa2[0][etiquetas3[i]][2]

    L.shape=l.shape #redimencionar un vector a matriz
    A.shape=a.shape
    B.shape=b.shape

    L=L[:,:,np.newaxis]
    A=A[:,:,np.newaxis]
    B=B[:,:,np.newaxis]

    new3=np.concatenate((L,A,B),axis=2)
```

```
    gris = rgb2gray(new3)
    #plt.figure()
    #plt.imshow(gris)
    bina = np.where(gris!=gris[0,0],1,0)

    return bina

########################################################################
#Se cargan las imagenes de referencia#################################
ref=[]
for i in range (3):
    imb=color.rgb2gray(io.imread('im'+str(i+1)+'.bmp'))
    salb,nob=labelling(imb)
    ext=extr(salb,nob,0)
    firm=chainc(ext)
    ref.append(firm)
########################################################################
deci=input('Segmentacion por: 1)color 2)forma 3) ambos)\n')
name2='imm7.jpeg'

if deci=='1':
    imab=excol(name2)
    imab =np.array(np.uint8(imab))
    kernel = np.ones((2,2),np.uint8)
    imab = cv2.erode(imab,kernel,iterations = 1)
    sale,noe=labelling(imab)
    print('figuras aisladas:'+str(noe.shape[0]))
    ssal=imab

elif deci=='2':
    imab=excol1(name2)
    imab =np.array(np.uint8(imab))
    kernel = np.ones((2,2),np.uint8)
    imab = cv2.erode(imab,kernel,iterations = 1)
    sale,noe=labelling(imab)
    print('numero de figuras:'+str(noe.shape[0]))
    # plt.figure(1)
    # plt.imshow(imab)
    # plt.pause(1)

    fs=input('que figura deseas aislar?\ntriangulo,cuadrado,circulo  \n')
    comp=[]
    ais=np.zeros(imab.shape)
    if fs=='cuadrado':
        comp=ref[0]
    elif fs=='triangulo':
        comp=ref[1]
    elif fs=='circulo':
```

```
      comp=ref[2]

   cc=0
   for i in range (noe.shape[0]):
      exte=extr(sale,noe,i)
      fire=chainc(exte)
      dis=dtw.distance(fire,comp)
      if dis<1.5:
         ais=ais+exte
         cc=cc+1
         print(dis)
   print('figuras aisladas:'+str(cc))
   ssal=ais

elif deci=='3':
   imab=excol(name2)
   imab =np.array(np.uint8(imab))
   kernel = np.ones((2,2),np.uint8)
   imab = cv2.erode(imab,kernel,iterations = 1)
   sale,noe=labelling(imab)
   print('numero de figuras:'+str(noe.shape[0]))
   #plt.figure(1)
   #plt.imshow(imab)
   #plt.pause(1)

   fs=input('que figura deseas aislar?\ntriangulo,cuadrado,circulo  \n')
   comp=[]
   ais=np.zeros(imab.shape)
   if fs=='cuadrado':
      comp=ref[0]
   elif fs=='triangulo':
      comp=ref[1]
   elif fs=='circulo':
      comp=ref[2]

   cc=0
   for i in range (noe.shape[0]):
      exte=extr(sale,noe,i)
      fire=chainc(exte)
      dis=dtw.distance(fire,comp)
      if dis<1.5:
         ais=ais+exte
         cc=cc+1
         print(dis)
   ssal=ais
   print('figuras aisladas:'+str(cc))

im=io.imread(name2)
```

```
im[:,:,0]=im[:,:,0]*ssal
im[:,:,1]=im[:,:,1]*ssal
im[:,:,2]=im[:,:,2]*ssal

plt.figure(3)
plt.imshow(im)
```

Resultados

Prueba 1

Clasificador por color

```
Segmentacion por: 1)color 2)forma 3) ambos)
1
figuras aisladas:2
```

Ilustración 3 Opción seleccionada y figuras aisladas

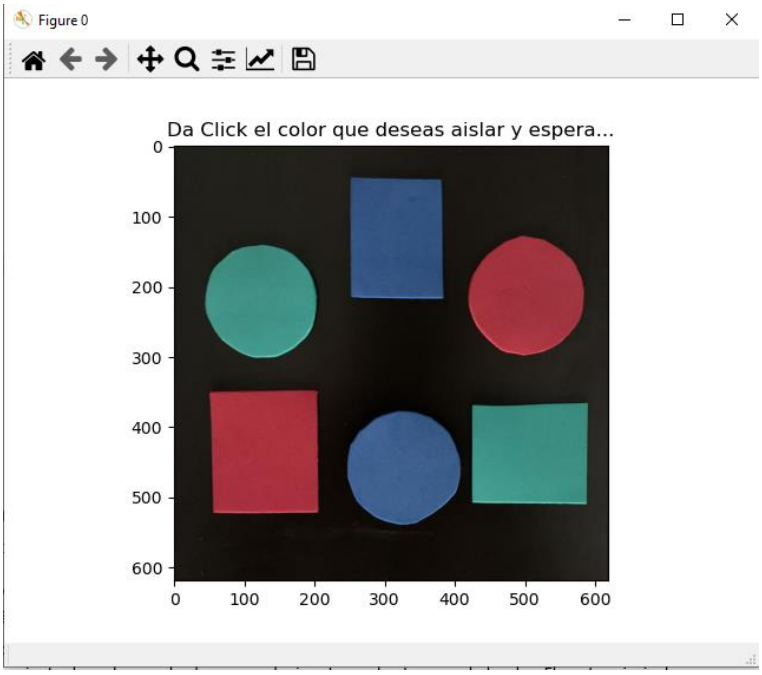

Ilustración 4 Selección de color deseado (color rojo)

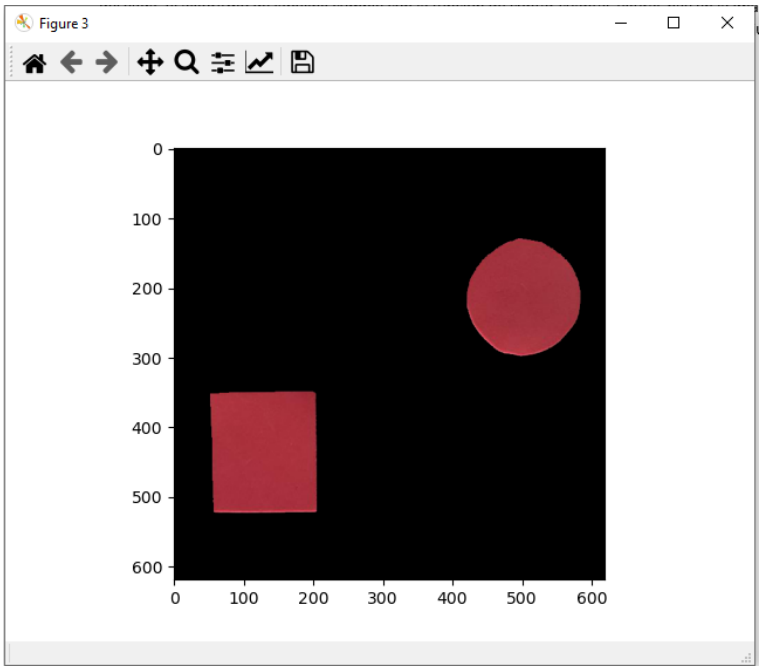

Ilustración 5 Segmentado del color rojo

Clasificador por forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
2
numero de figuras:6

que figura deseas aislar?
triangulo,cuadrado,circulo
circulo
1.0754552151817887
0.9442525660906931
0.8187445042257213
figuras aisladas:3
```

Ilustración 6 Opción seleccionada y figuras aisladas

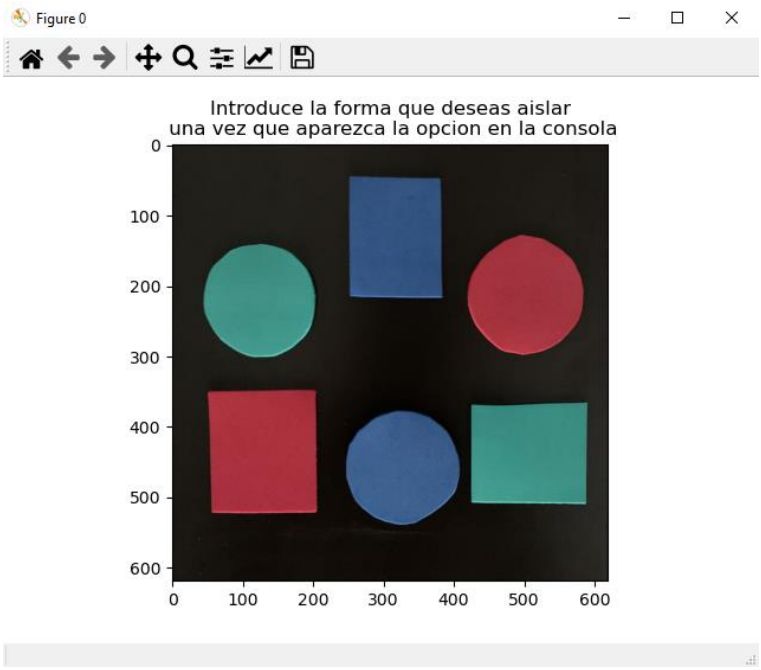

Ilustración 7 Formas disponibles para segmentar

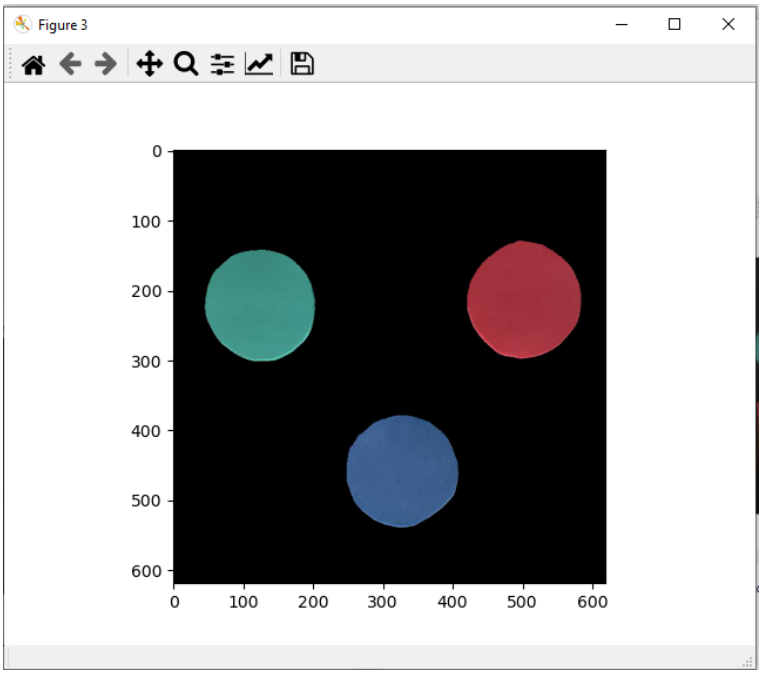

Ilustración 8 Círculos segmentados

Clasificador por color y forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
3
numero de figuras:2

que figura deseas aislar?
triangulo,cuadrado,circulo
cuadrado
1.1580742862883218
figuras aisladas:1
```

Ilustración 9 Opción seleccionada y figura aislada

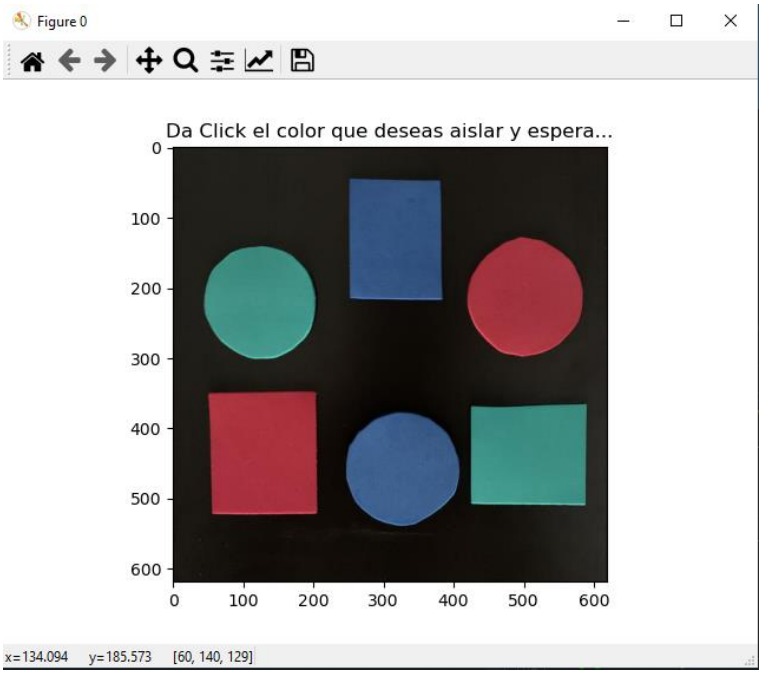

Ilustración 10 Selección del color para aislar

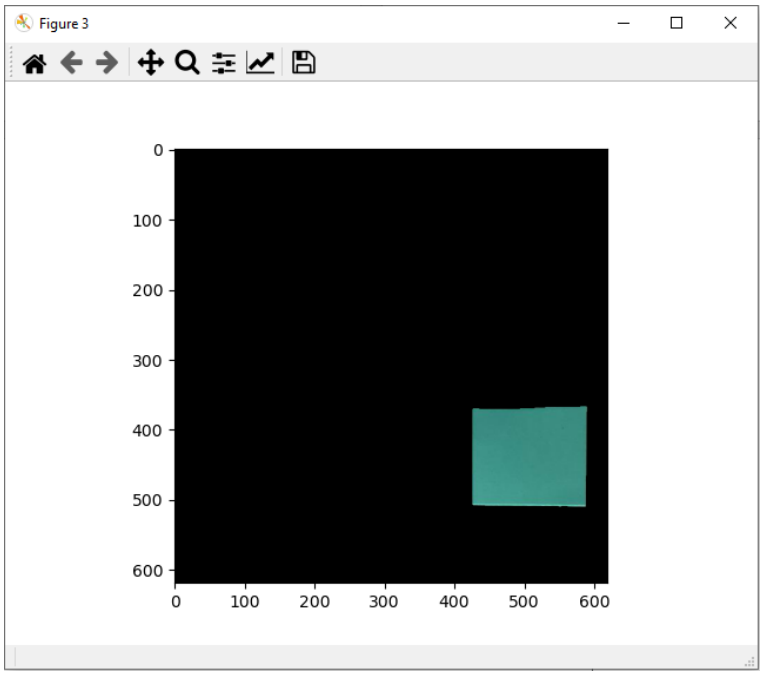

Ilustración 11 Figura segmentada con el color deseado

Prueba 2

Clasificador por color

```
Segmentacion por: 1)color 2)forma 3) ambos)
1
figuras aisladas:2
```

Ilustración 12 Opción seleccionada

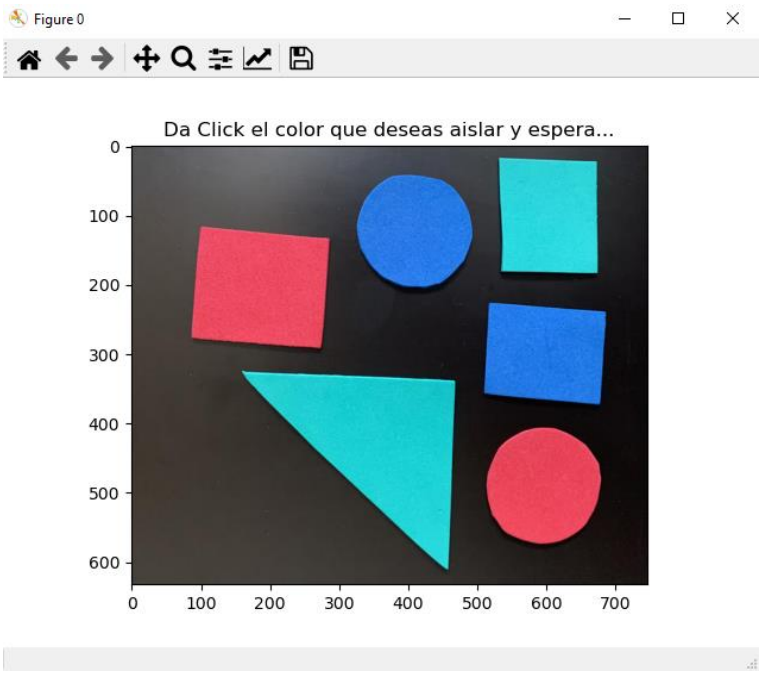

Ilustración 13 Selección del color a segmentar (azul)

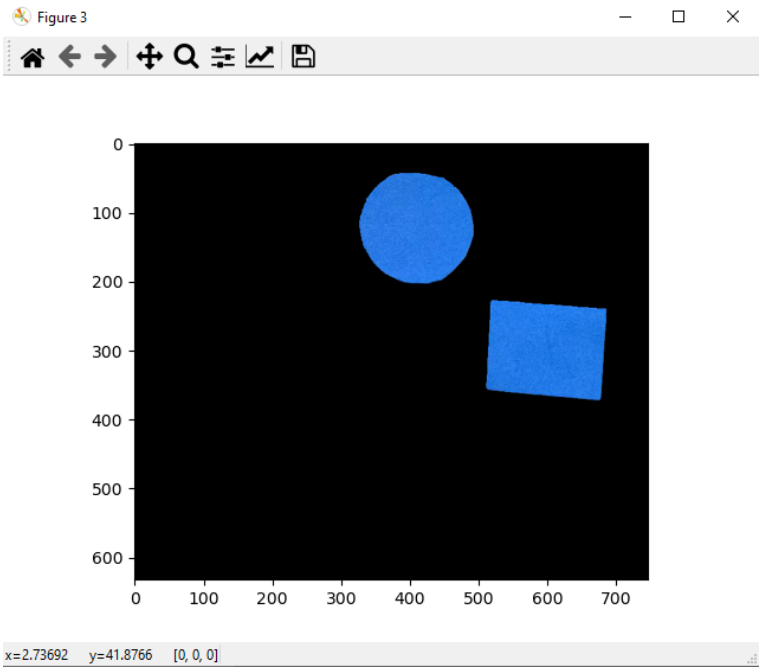

Ilustración 14 Segmentado de color azul

Clasificador por forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
2
numero de figuras:6

que figura deseas aislar?
triangulo,cuadrado,circulo
triangulo
1.1965362515511204
figuras aisladas:1
```

Ilustración 15 Opción seleccionada y figuras aisladas

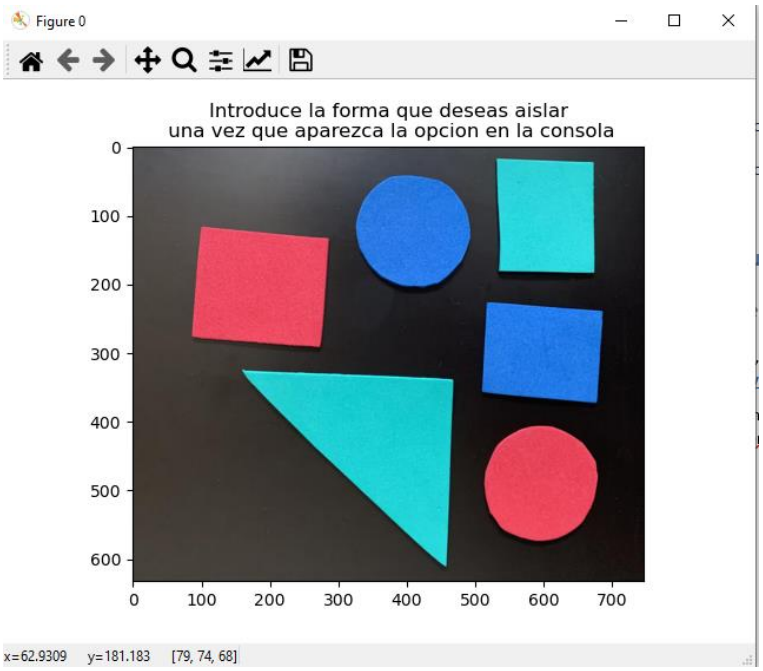

Ilustración 16 Formas disponibles para segmentar

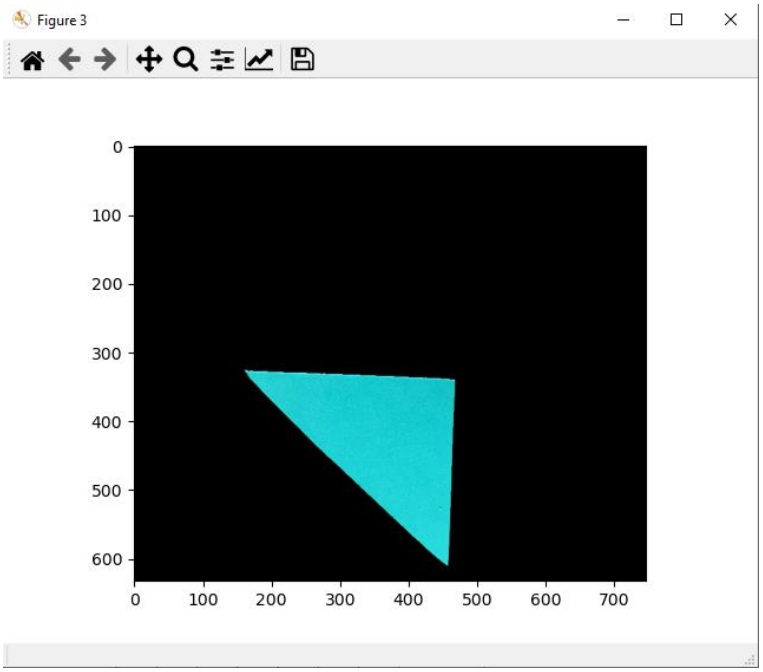

Ilustración 17 Triángulo segmentado

Clasificador por color y forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
3
numero de figuras:2

que figura deseas aislar?
triangulo,cuadrado,circulo
circulo
0.9973256236383109
figuras aisladas:1
```

Ilustración 18 Opción seleccionada y figura aislada

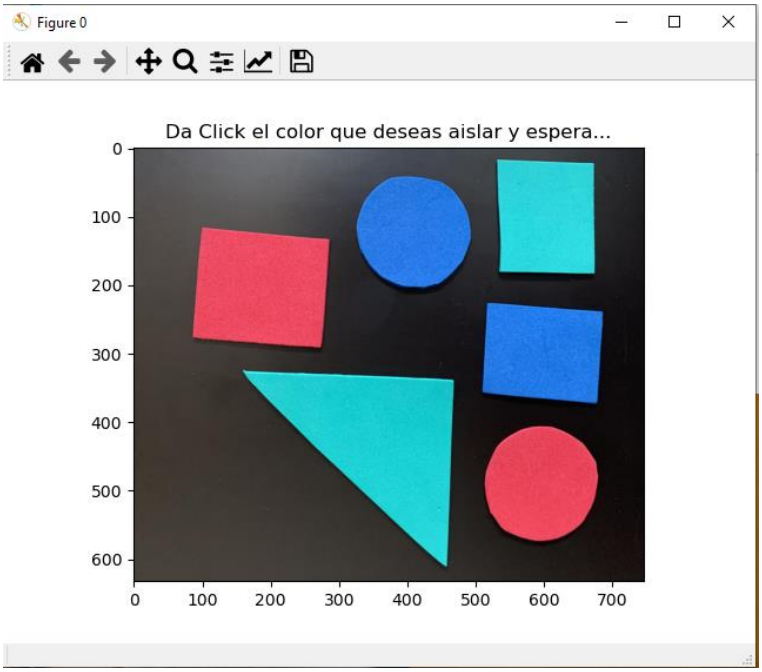

Ilustración 19 Selección de color para aislar

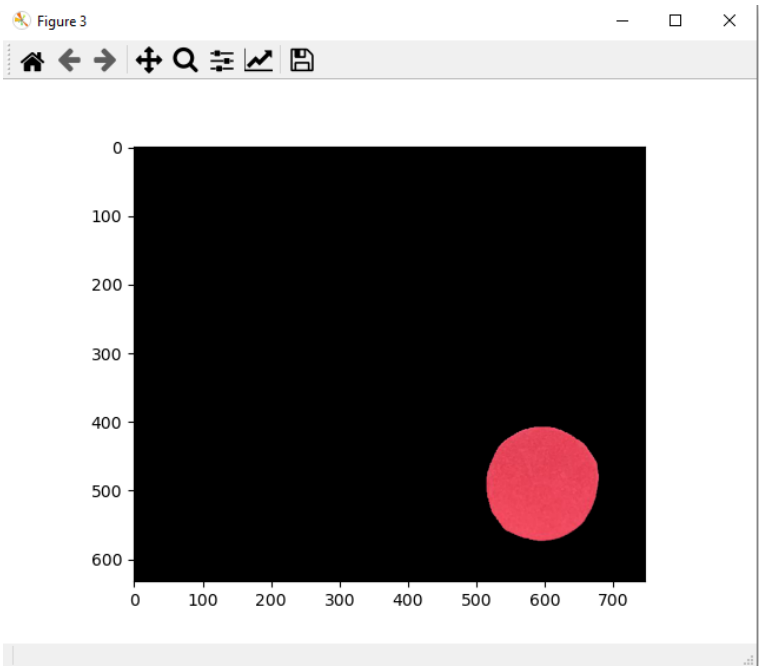

Ilustración 20 Figura aislada con el color deseado

Prueba 3

Clasificador por color

```
Segmentacion por: 1)color 2)forma 3) ambos)
1
figuras aisladas:2
```

Ilustración 21 Opción seleccionada y figuras aisladas

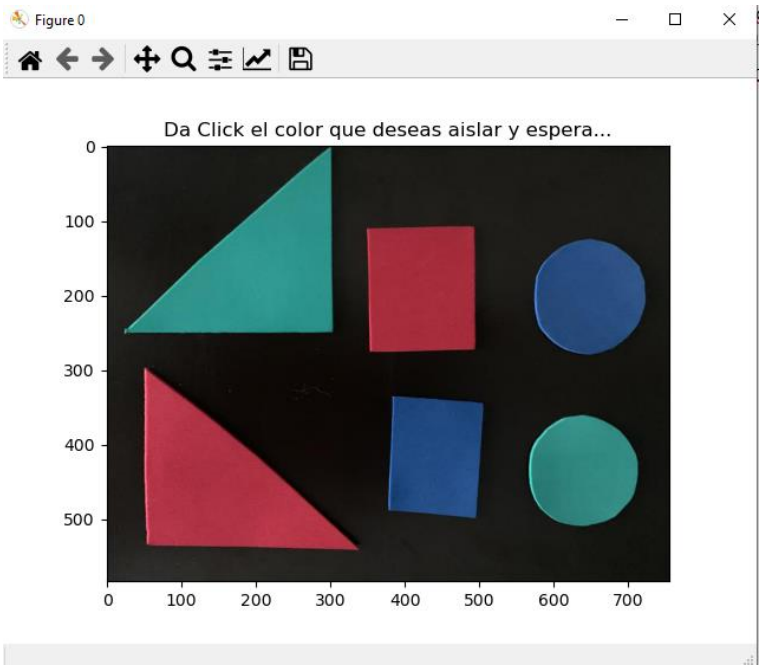

Ilustración 22 Selección de color a segmentar (verde)

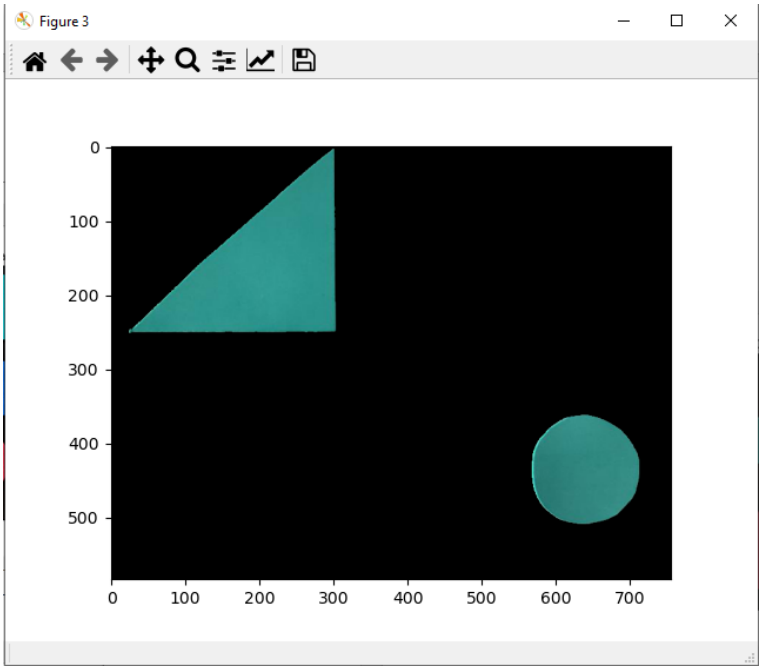

Ilustración 23 Segmentado del color verde

Clasificador por forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
2
numero de figuras:6

que figura deseas aislar?
triangulo,cuadrado,circulo
cuadrado
C:\Users\PEDRO\Desktop\Octavo\Vision\Practica2\p
in true_divide
  firma=np.array(firma/np.max(firma))
0.46076354962583244
1.0912615046078558
figuras aisladas:2
```

Ilustración 24 Opción seleccionada y figuras aisladas

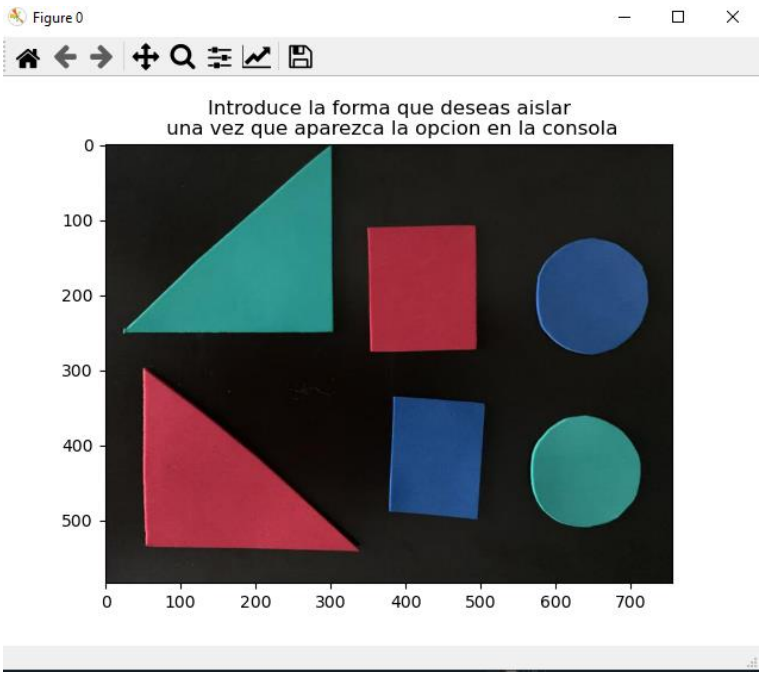

Ilustración 25 Formas disponibles para segmentar

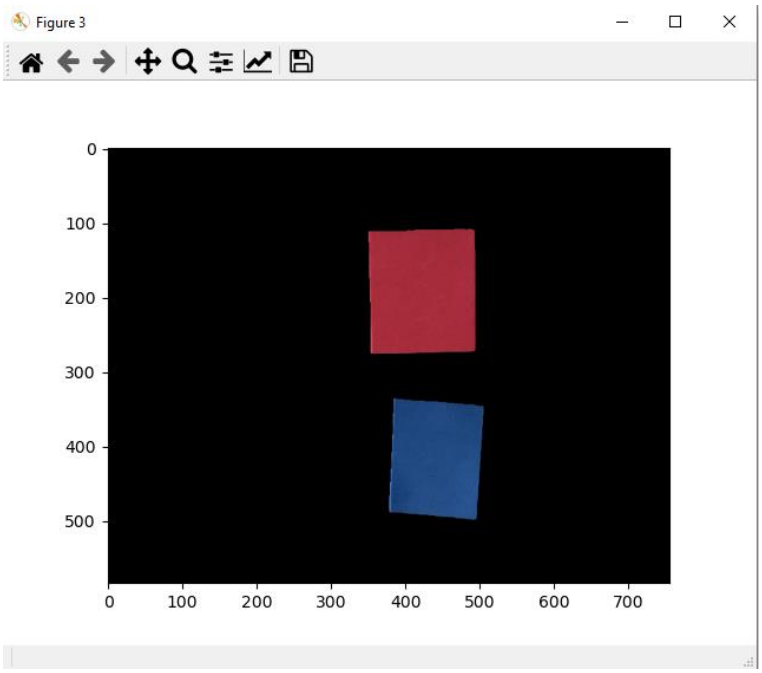

Ilustración 26 Cuadrados segmentados

Clasificador por color y forma

```
Segmentacion por: 1)color 2)forma 3) ambos)
3
numero de figuras:2

que figura deseas aislar?
triangulo,cuadrado,circulo
circulo
1.005914459087219
figuras aisladas:1
```

Ilustración 27 Opción seleccionada y figura aislada

Ilustración 28 Selección de color a aislar

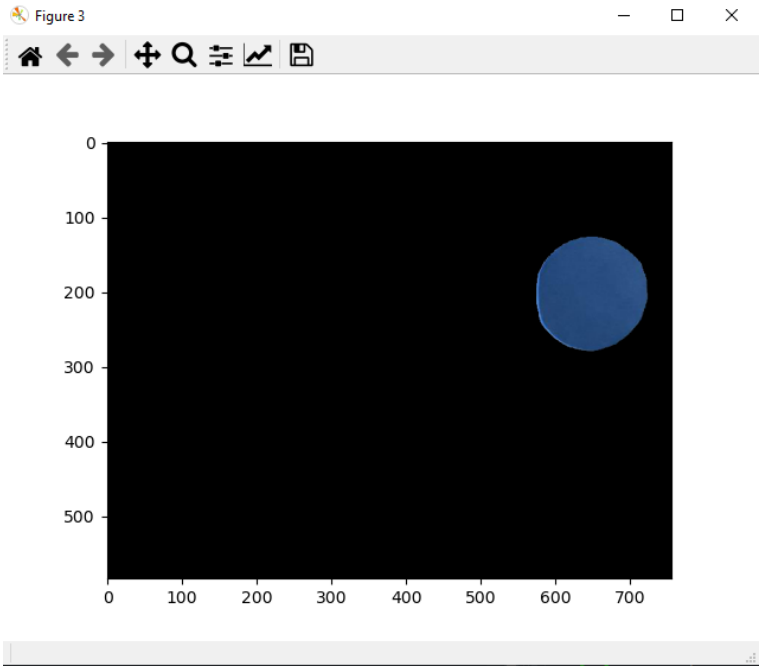

Ilustración 29 Figura aislada con el color deseado

Conclusiones

La práctica entregó resultados muy buenos ante distintas pruebas realizadas con diferentes formas de colocar las figuras, por el momento se encuentra limitado a tres figuras, sin embargo, se puede aumentar la capacidad de figuras a reconocer. Además, del uso de solo tres colores para que la segmentación por k-means tuviera la menor cantidad de confusión posible, hay que tener en cuenta el número de clases a utilizar debido a que pueden generar problemas en el procesamiento. Las imágenes recibieron un tratamiento de erosión, ya que a la hora de ser procedas arrojaban ruido que era imperceptible al ojo humano, pero el sistema lo detectaba generando conflicto en el procesamiento. El uso del código cadena nos permite detectar un elemento por la forma que tiene debido a que recorre el elemento mediante un sistema de pasos ya preestablecidos, esto funciona si los pixeles son pixeles frontera, por lo cual el uso de elementos cerrados es necesario como de la eliminación de ruido que pueda existir; mientras que el algoritmo de labelling nos permite tener en cuenta cuantos grupos distintos de elementos tenemos en nuestra imagen. La identificación de elementos mediante firmas obtenidas mediante el código cadenas es una forma muy confiable de discernir entre dos elementos, podemos usar la firma y procesarla mediante DTW para comparar el grado de parecido entre ambas señales obtenidos y poder segmentar el elemento. El programa considero que quedó lo suficientemente robusto para lograr clasificar distintos objetos en el apartado de figuras, se puede seguir trabajando en la sección de color y en el poder de procesado, ya que imágenes muy grandes pueden generar que le procesamiento lleve cierto tiempo

-Aguilar Gómez Pedro Nicolás

Por medio de esta práctica tuvimos la oportunidad de implementar métodos para la identificación de objetos como es el código cadena o la conexión de componentes “labelling”, primero que nada, cabe denotar que ambos métodos nos sirven para aislar los objetos dentro de una imagen y a su vez saber cuántos objetos hay, no obstante, ambos tienen sus ventajas y desventajas, en el caso del labelling con este algoritmo nosotros podemos agrupar todos los pixeles correspondientes a un mismo objeto y etiquetarlos para posteriormente segmentarlo, de igual forma el número de etiquetas es equivalente al número de objetos dentro de la imagen, no obstante, uno de los inconvenientes se presenta al obtener la firma del objeto, ya que la firma toma formas oscilantes debido a que se obtiene el cálculo de la distancia en forma de barrido, esta situación no se presenta en el Código cadena ya que este algoritmo nos permite almacenar las posiciones de los pixeles consecutivos, es decir, nos da información sobre la consecutividad de sus pixeles proporcionándonos una firma más adecuada para clasificar. Para el caso del código cadena, como se mencionó la obtención de la firma es mejor, pero para segmentar la imagen requiere de un proceso más robusto. En este trabajo se implementaron ambos códigos de forma complementaria obteniendo resultados bastante buenos para la segmentación de objetos por su forma, que complementándolo con las propiedades de color podemos hacer un sistema muy selectivo para aislar objetos dentro de una imagen.

-Sánchez Sánchez Luis Alberto

Referencias

B. (2020, 19 octubre). Connected Components in a Graph. Baeldung on Computer Science. https://www.baeldung.com/cs/graph-connected-components

H. Freeman. On the encoding of arbitrary geometric configurations, IRE Transactions on Electronic Computers EC- 10(1961) 260-268.