Visión por Computador

Grado en Informática Industrial y Robótica (GIIROB)

Práctica 4:

Procesamiento de imágenes binarias

Contenido:

1. Técnicas de umbralización
   1.1. Umbralización global
   1.2. Umbralización local
   1.3. Umbralización por color
2. Etiquetado y cuenta de objetos
3. Coloreado de los objetos
4. Entrega de resultados

Autor:

José Miguel Valiente González

Bibliografía:

- J. Howse & J. Minichino. “Learning OpenCV 4 Computer Vision with Python 3” (third Ed.) Packt Publishing Ltd. 2020.
- Sandigan Dei, “Python Image Processing. Cookbook”. Packt Publishing Ltd. 2020.
- B.R. Hunt; R.L. Lipsman, J.M. Rosenberg. “A guide to MATLAB”. Cambridge University Press. 2001.
- Rafael Gonzalez, Richard Woods, Steven Eddins. ”Digital Image Processing Using Matlab”. Second Edition, McGraw Hill, 2010
- Official documentation for Python 3.10 https://docs.python.org/es/3/tutorial/index.html
- https://es.mathworks.com/academia/books.html

Esta práctica se introducen las técnicas de segmentación que trabajan con imágenes binarias. En esta parte se describen las técnicas de umbralización global, basadas en la detección automática del umbral, y local basadas en el análisis de los vecinos de cada pixel. Estas técnicas también se pueden emplear sobre las componentes de color. Otras técnicas de segmentación se mostrarán mediante el uso de Apps específicas. Finalmente se emplean las imágenes binarias para realizar operaciones de etiquetado y cuenta de objetos, así como para obtener descriptores de esos objetos.

Objetivos

- Describir las funciones (Python) que tiene OpenCV para hacer operaciones de segmentación basadas en umbralización global y local, y también basadas en las componentes de color.
- Emplear funciones de componentes conectadas de objetos binarios.
- Utilizar la APP de procesamiento y segmentación de imágenes en color: **App Color Thresholder** de MATLAB.

Material

- Scripts de PYTHON: *umbralizar_global.py* y *umbralizar_color.py.*
- Archivos de imagen disponibles en PoliformaT: Cartas de póker.

1. Técnicas de umbralización

La umbralización de una imagen de niveles de gris – o una componente de una imagen de color – consiste en obtener la imagen binaria correspondiente mediante un valor umbral – *threshold* – reemplazando los pixeles por encima del umbral con un valor '1' y el resto con un valor '0'.

La umbralización es un método muy utilizado en las aplicaciones de visión, pues permite segmentar la imagen en objetos que comparten un valor similar de gris o de componente de color. La imagen binaria resultante es más fácil de procesar, para individualizar y medir dichos objetos, o para seguir procesándolos recortándolos de la imagen original. Dependiendo de la forma como se obtiene ese valor umbral se definen métodos de umbralización global o local.

1.1. Umbralización global

Cuando el valor umbral es un valor que se obtiene en a partir del histograma global de la imagen, tenemos los métodos de umbralización global con selección automática del umbral. En ocasiones el umbral es un valor fijo que se obtiene manualmente mediante prueba y error. En otros casos, como en el método de Otsu o el de Triangle, estos valores umbral se obtienen automáticamente. Las funciones de **OpenCV** para umbralizar una imagen son:

- retval, dst = cv2.threshold(src, thresh, maxval, type[, dst])
- Parámetros:
  - src: array. Imagen fuente (multichannel – uint8 o float32).
  - thresh: Umbral para binarizar.
  - maxval: Valor máximo de la salida binaria.
  - type: Tipo de umbralización: cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2. THRESH_TOZERO , cv2.THRESH_TOZERO_INV , cv2.THRESH_OTSU , cv2.THRESH_TRIANGLE , cv2.THRESH_MASK.
- Devuelve:
  - retval: Umbral calculado en los métodos de Otsu y Triangle.

- dst: Imagen binaria/gris de salida.

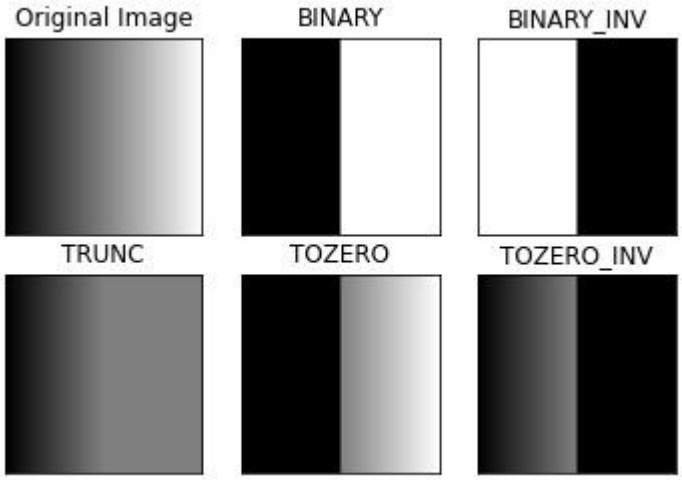

Figura 1 – Imagen original y salidas de distintos métodos de umbralización global.

En los métodos de umbralización global, el valor umbral se establece manualmente. En cambio, en los métodos de Otsu y del Triángulo ese umbral se obtiene automáticamente. En el método de Otsu, el umbral '*t*' se obtiene por el valor que minimiza la varianza intra-clase, dada por:

$$\sigma_w = q_1(t) \cdot \sigma_1^2(t) + q_2(t) \cdot \sigma_2^2(t)$$

donde la imagen se divide en dos partes, con sumas (q_1 y q_2) y varianzas (σ_1^2 *y* σ_2^2).

En el método del triángulo, el umbral se obtiene buscando el valor '*d*' que maximiza la distancia de la recta que une los valores mínimo y máximo del histograma, con el valor del histograma en perpendicular. El valor de gris 'b_0' que se corresponde con lo anterior es el umbral buscado. La figura 2 muestra esta idea.

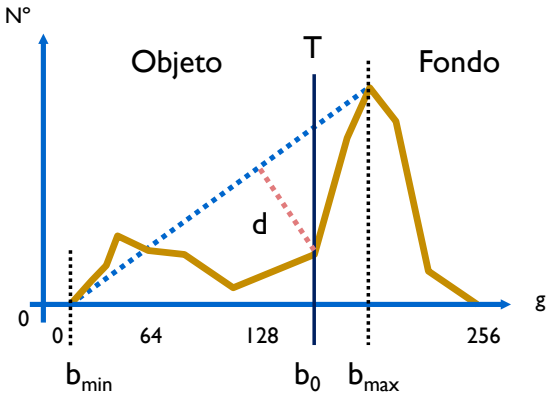

Figura 2 – Método de umbralización del triángulo.

El siguiente script muestra la forma de actuar en Python.

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

```
img = cv2.imread('gradient.png', cv2.IMREAD_GRAYSCALE)
ret,thresh1 = cv2.threshold(img,127,255,cv.THRESH_BINARY)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1]
for i in range(2):
        plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
        plt.title(titles[i])
        plt.xticks([]),plt.yticks([])
plt.show()
```

► *Ejercicio 1 – Umbralización global*

1. En la carpeta <Recursos/Imágenes VxC/Baraja póke 1/> de PoliformaT hay un conjunto de imágenes de cartas de póker que vamos a utilizar a partir de ahora. Busque esas imágenes y póngalas en una carpeta propia.

2. Introduzca el código anterior con el nombre '*umbralización.py*' y pruebe el resultado con distintas imágenes de la carpeta de cartas.

3. Añada los otros métodos de umbralización indicados en la lista '*type*'.

Observe que la función cv2.threshold(…) utiliza imágenes de niveles de gris. Por este motivo la imagen se lee con cv2.imread(filename, cv2.IMREAD_GRAYSCALE). Si leemos la imagen en color habrá que convertirla a nivel de gris con cv2.cvtColor(img, cv2.BGR2GRAY).

► *Ejercicio 2 – Umbralización global de las cartas*

1. Con el script anterior hay que modificar el valor del archivo y/o los valores de umbral para cada imagen. Para tener una versión más interactiva tome el archivo *umbralización_global.py* de PoliformaT. Lea el código y estudie cómo se lee una carpeta y se listan las imágenes. También cómo se utiliza un deslizador (Trackbar) para definir el valor del umbral.

2. Aplique la umbralización global de tipos cv2.THRESH_BINARY y cv2.THRESH_BINARY_INV, a diversas imágenes de las cartas.
   ¿Qué objeto es el más indicado para segmentar en este caso? La idea es que, a veces, es conveniente extraer un objeto que nos delimite las partes de la imagen que queremos observar. En este caso el objeto a discriminar es la carta, aunque ese objeto tenga agujeros en su interior.

3. ¿Qué umbral sería apropiado para la umbralización global funcione bien con todas las imágenes?

4. Pruebe también los métodos de Otsu y Triangle, para ver si obtienen un umbral apropiado para todas las imágenes de conjunto de cartas.

1.2. Umbralización local

Estos métodos se basan en calcular un estadístico de primer orden en base a los píxeles vecinos alrededor de cada pixel de la imagen y obtener así un umbral local para dicho punto. Así podemos construir una matriz de MxN umbrales, para todos los puntos de la imagen, donde ese umbral varía para obtener mejores resultados en zonas de iluminación variable. Dependiendo de cómo es el estadístico local se tienen distintos

métodos de umbralización local. Los valores de media, mínimo, máximo o mediana se obtienen en una ventada de vecinos del pixel de la imagen.

- Método de la mediana:
  pixel = (pixel > mediana – c) ? 0 : 1; c = 0
- Método MidGrey:
  pixel = (pixel > ((max-min)/2) – c) ? 0 : 1; c = 0
- Método de la media:
  pixel = (pixel > mean – c) ? 0 : 1; c = 0
- Método Niblack:
  Pixel = (pixel > mean + k*stdev – c) ? 0 : 1; k = 0.2; c = 0
- Método Phansalkar:
  pixel = (pixel > mean*(1+p*exp(-q*mean)+k*((stdev/r)-1)) ? 0 : 1;
  p = 2; q = 10; k = 0.25, r = 0.5
- Método de Sauvola:
  pixel = (pixel > mean *(1+k*(stdev/r-1))) ? 0 : 1; k = 0.5; r = 128
- Método de Gaussiano:
  pixel = correlación con un kernel Gaussiano menos C

Los valores de la imagen resultante pueden ser 0 o 1, como se indica arriba, o bien 0 y 255.

La función de OpenCV utilizada para este propósito es:

- dst = cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst])
- Parámetros:
  - src: array. Imagen fuente (multichannel – uint8).
  - maxValuel: Valor no cero a asignar a los píxeles que cumplen la condición.
  - adaptiveMethod: Método de umbralización adaptativa:
    ADAPTIVE_THRESH_MEAN_C o ADAPTIVE_THRESH_GAUSSIAN_C.
  - thresholdType: Tipo de umbralización binaria a aplicar a las imágenes cuando son de niveles de gris: cv2.THRESH_BINARY o cv2.THRESH_BINARY_INV.
  - blockSize: Tamaño del área de vecindad: 3,5,7,..
  - C: Constante a restar del valor calculado.
- Devuelve:
  - dst: Imagen umbralizada de salida.

Observe que OpenCV sólo implementa la umbralización local con dos métodos: media y Gaussiano. Para los otros métodos habría que implementar las funciones en Python.

► *Ejercicio 3 – Umbralización local*

1. Tome el script del ejercicio anterior y cree uno nuevo para umbralización local, denominado '*umbralizacion_local.py*'. Este código debe definir una función '*myAdaptiveThreshols()*' que llame a la función de OpenCV para los métodos 'Media' y 'Gaussiano'. Recuerde que esta función utiliza imágenes de 2 dimensiones (grises), por lo que habrá que pasar las imágenes en color a grises o leerlas directamente en gris.

2. Pruebe el script anterior con distintas imágenes y con distintos métodos.

3. Complete los métodos 'MidGrey' y 'Niblack' en la función *myAdaptivethreshold()* y pruébelos.

1.3. Umbralización por color

Puesto que las imágenes que se utilizan son de color, podemos recurrir a discriminar los objetos de interés empleando las coordenadas de color. La idea consiste en definir un rango de valores para las coordenadas de color en el cual se discriminan los objetos o partes de interés. Se pueden emplear coordenadas RGB, HSV, L*a*b, etc. El problema consiste en que conviene tener una aplicación que nos permita seleccionar, de forma interactiva, los tipos de coordenadas de color y los rangos de valores. En Python este proceso es complicado. Sin embargo, en Matlab la operación es más fácil. Para ello vamos a utilizar una APP disponible en el entorno de Matlab, llamada '***Color Thresholder***' que podemos encontrar en la pestaña 'APPS' de la ventana principal del entorno.

Si desplegamos esa pestaña nos encontraremos con una gran cantidad de aplicaciones predefinidas que nos pueden ayudar en nuestra tarea. En particular, dentro del grupo de apps llamado '*Image Processing and Computer Vision*' encontramos la aplicación '***Color Thresholder***' que podemos seleccionar para su ejecución.

Nos aparecerá una ventana de entrada de la aplicación donde, tras seleccionar una imagen desde un archivo, podemos seleccionar en qué espacio de color vamos a trabajar. La imagen original, de tipo jpg, siempre está en coordenadas de color **RGB**. Pero tenemos otros espacios de color, como **HSV**, **YCbCr** o **La*b***, que también podemos utilizar - Véase el anexo: *Espacios de color*, disponible en PoliformaT. El aspecto de la ventana indicada se muestra en la figura 3.

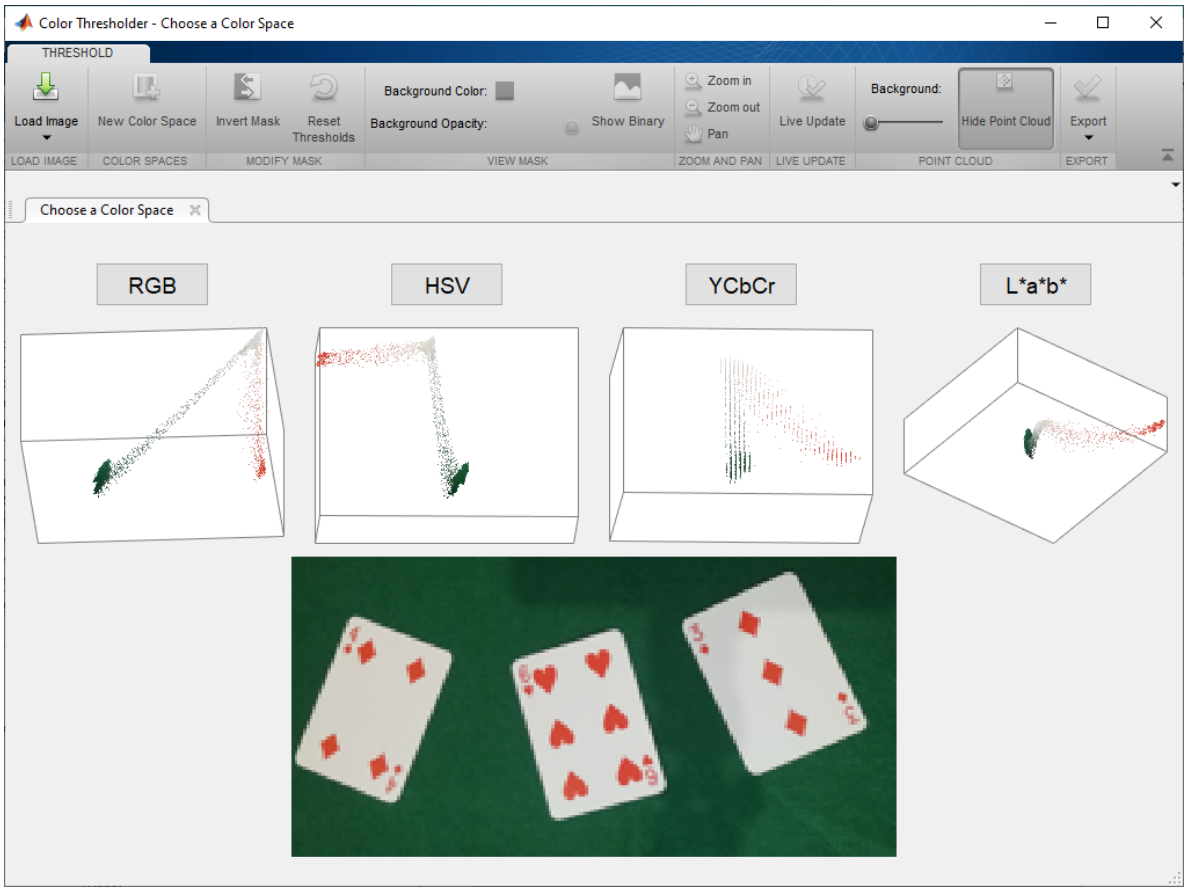

Figura 3 – Ventana de entrada de la aplicación ***Color Thresholder*** de Matlab.

Tras seleccionar el espacio de color en que vamos a trabajar (**HSV**), se crea una nueva vista en la aplicación donde nos aparecen los histogramas o gráficos de las tres componentes de color del espacio seleccionado y donde podemos establecer, interactivamente, los valores superior e inferior para umbralizar, así como el resultado que se obtiene. El aspecto de esa segunda vista se muestra en la figura 4.

Se observa que aquellos puntos de color que están dentro de los umbrales marcados quedan intactos en la imagen original, mientras que los que caen fuera de umbrales se ponen a negro. Se puede obtener justo lo contrario activando la opción *'Invert Mask'*.

En definitiva, de lo que se trata es de buscar unos umbrales para las componentes de color entre los que se pueden segmentar las regiones de interés. También es posible buscar lo contrario, es decir, buscar los umbrales para seleccionar sólo el fondo. Este es el caso de las cartas de póker, pues el fondo tiene un color verde bastante distintivo y que se repite en las distintas imágenes, aunque haya cambios de intensidad. Observe que cambiando los umbrales de la componente **H** (hue: color) y de la componente **S** (Saturation) se puede discriminar el fondo.

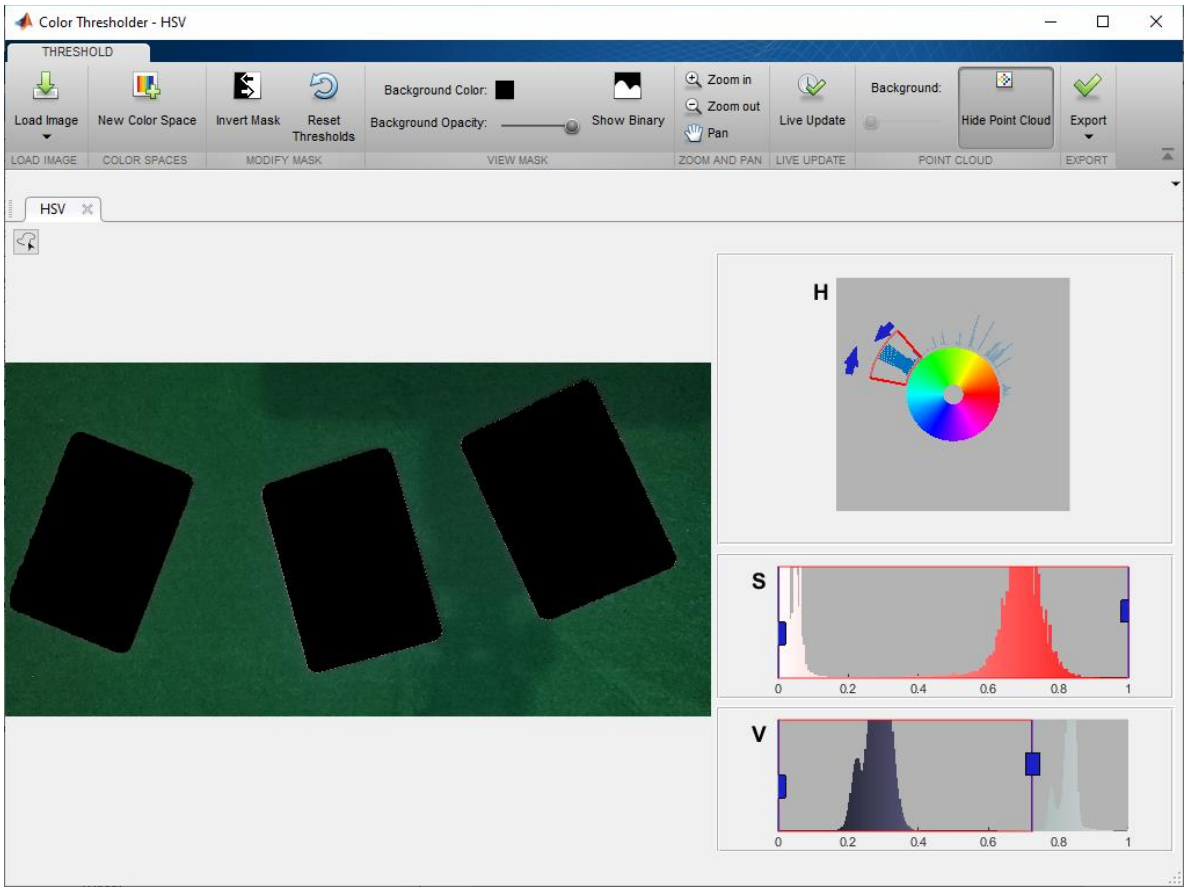

Figura 4 – Vista de coordenadas de color de la aplicación ***Color Thresholder*** y resultado de la umbralización pro rango de color en HSV.

Una vez obtenidos los umbrales se deben probar con otras imágenes del conjunto de imágenes disponible, para verificar que dichos umbrales son generales. Esto es algo complicado, pues la APP no visualiza los valores umbrales y, con cada imagen, hay que repetir el proceso de forma aproximada.

Para evitar estos inconvenientes, la APP dispone de una opción '*Export->Export Function*' que crea automáticamente un script de Matlab con la función "createMask(RGB)" que podemos usar para aplicarla a nuestras imágenes. Esta función devuelve una máscara de segmentación, que no es otra cosa sino una imagen binaria – de tipo *logical* - con los objetos segmentados. Pero nosotros no vamos a usar esa función. En su lugar vamos a crear nuestra propia función en Python.

► *Ejercicio 4 – Umbralización por color*

1. Tome de PoliformaT el script '*umbralizacion_color.py*'. Revise el programa para entender lo que está haciendo y complete las funciones que faltan para convertir de BGR a HSV y busque la información de **OpenCV** de la función cv2.inRange(.....) e inserte ésta en el código.

2. Pruebe con distintas imágenes de la carpeta de cartas de póker.

3. Establezca los umbrales de H y S para segmentar el fondo verde y verifique que funciona bien para todas las cartas.

2. Etiquetado y cuenta de objetos

En todos los apartados anteriores se completa una operación de umbralización obteniendo una imagen binaria – BW – que contiene los objetos (1) y el fondo (0). A veces lo que segmentamos es el fondo, con lo cual si invertimos el resultado obtenemos las zonas donde estarán los objetos de interés. Luego podemos centrarnos en esas zonas, y hacer una umbralización sólo en dichas zonas, para discriminar a su vez los objetos que contengan.

Tomemos el ejemplo de la umbralización global del ejercicio 2. La figura 5 muestra el resultado de la umbralización de la carta 10 de picas. El resultado es la imagen de la derecha, una imagen BW invertida para mostrar los objetos en negro (0) sobre fondo blanco (255). Los objetos de interés están en el objeto obtenido es el fondo de la carta.

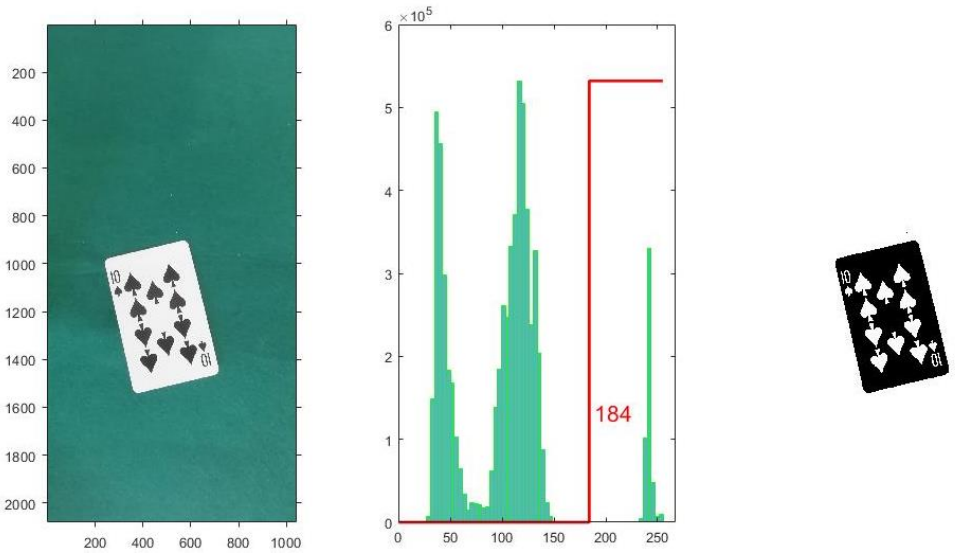

Figura 5 – Resultado del ejercicio de umbralización global.

La cuestión es si podemos obtener los objetos que hay dentro de la carta que, como se aprecia, son parte del fondo, pero son objetos individuales – figuras de picas, el número y posiblemente algún punto o zona aislada de ruido. Para esto seguimos usando el truco de trabajar con la imagen invertida – cv2.THRESH_BINARY_INV – donde todo lo que era fondo son ahora objetos diferentes.

Para Python, las funciones que calculan las componentes conectadas de una imagen binaria son:

- retval, labels = cv2.connectedComponents(image[, labels[, connectivity[, ltype]]]
- retval, labels=cv2.connectedComponentsWithAlgorithm(image, connectivity, ltype, ccltype[, labels])
  Parametros:
  - image: Imagen de 8 bits a ser etiquetada.
  - labels: Imagen de salida etiquetada.
  - connectivity: Conectividad 8 o 4.
  - ltype: Tipo de la imagen de etiquetas. CV_32S o CV_16U.
  - ccltype: Tipo de algoritmo para obtener las componentes conectadas. cv.CCL_DEFAULT, cv2.CCL_WU, cv2.CCL_GRANA, cv2.CCL_BOLELLI, cv2.CCL_SAUF, cv2.CCL_BBDT, cv2.CCL_SPAGHETT).

  Devuelve:
  - retval: Número total de etiquetas [0,N-1], donde 0 representa el fondo.
  - labels: Imagen de salida etiquetada.

  Existen funciones adicionales que devuelven más valores estadísticos de los objetos encontrados.

- retval, labels, stats, centroids = cv2.connectedComponentsWithStats(image[, labels[, stats[, centroids[, connectivity[, ltype]]]]])
- retval, labels, stats, centroids = cv2.connectedComponentsWithStatsWithAlgorithm(image, connectivity, ltype, ccltype[, labels[, stats[, centroids]]])
  Devuelve:
  - retval: Número total de etiquetas [0,N-1], donde 0 representa el fondo.
  - labels: Imagen de salida etiquetada.
  - stats: Estadísticas de cada objeto. Fila: Nº de objeto, Columnas:
    - 0: cv2.CC_STAT_LEFT: Coordenada X de inicio del Bounding Box.
    - 1: cv2.CC_STAT_TOP: Coordenada Y de inicio del Bounding Box.
    - 2: cv2.CC_STAT_WIDTH: Tamaño horizontal del Bounding Box.
    - 3: cv2.CC_STAT_HEIGHT: Tamaño vertical del Bounding Box.
    - 4: cv2.CC_STAT_AREA: Área del Bounding Box.
  - centroids: Centroide de cada objeto (X,Y).

Para hacer uso de estas funciones hay que añadir la siguiente biblioteca en el IDE de Python:

pip install opencv-contrib-python

A modo de ejemplo de estas funciones podemos introducir el siguiente código:

```
import cv2
import numpy as np
import tkinter as tk
from tkinter import filedialog
import numpy as np
import os

window_name = 'Original image'
window_filtered_name = 'Filterd objects'
cv2.namedWindow(window_name, flags=cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO )
cv2.namedWindow(window_filtered_name,  flags=cv2.WINDOW_NORMAL | cv2.WINDOW_KEEPRATIO)

areaMinima = 500
umbral = 155
folders = './output1/'
folder_name = filedialog.askdirectory(initialdir=folders)

f2Name = folder_name + '/'
list_files = os.scandir(f2Name)
for ent in list_files:
```

```
    if ent.is_file() and ent.name.endswith('.jpg'):
        filename = f2Name + ent.name
        img = cv2.imread(filename)
        gray_img = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)

        threshold = cv2.threshold(gray_img, umbral, 255, cv2.THRESH_BINARY_INV)[1]
           # Aplica la función de componentes conectadas
        analysis = cv2.connectedComponentsWithStats(threshold, 4, cv2.CV_32S)
        (totalLabels, label_ids, values, centroid) = analysis

        output = np.zeros(gray_img.shape, dtype="uint8")

        for i in range(1, totalLabels):  # Para todos los objetos encontrados
               # Área de los componentes
            area = values[i, cv2.CC_STAT_AREA]
            if (area > areaMinima):   # Filtrado por área mínima
                componentMask = (label_ids == i).astype("uint8") * 255
                output = cv2.bitwise_or(output, componentMask)

        cv2.imshow(window_name, img)
        cv2.imshow(window_filtered_name, output)
        key = cv2.waitKey(0)
        if key == ord('q') or key == 27:    # 'q' o ESC para acabar
            break

cv2.destroyAllWindows()
```

Observe que la imagen de salida (output) tiene el valor 255 para blanco y 0 para negro. Los objetos encontrados por la función de componentes conectadas se filtran por un tamaño mínimo de área

► *Ejercicio 5 – Etiquetado y cuenta de objetos*

1. Tome de script anterior y cree un nuevo archivo denominado '*componentes_conectadas.py*'. Revise el programa para entender lo que está haciendo y pruebe con distintas imágenes de la carpeta de cartas de póker.
2. Cambie el código para que funcione con 8 conexión. Prueve cambiando el área mínima.

► *Ejercicio 6 – Recuadro de los objetos*

1. Añada al código anterior la obtención del recuadro (Bounding Box) de cada objeto que supera el filtro de tamaño y muéstrelo en la imagen original mediante cv2.rectangle(image, start_point, end_point, color, thickness). Utilice al color azul claro (255,255,0) y grosor de 10 px.
2. Ídem que lo anterior, pero mostrando el centroide mediante cv2.circle(image, center_coord, radius, color, thickness). Utilice el color amarillo (255,255,0), radio 4 y grosor 5.
3. Pruebe con distintas imágenes de la carpeta de cartas de póker.
4. ¿Qué ocurre si en vez de aplicar `cv2.THRESH_BINARY_INV` utilizamos `cv2.THRESH_BINARY`?

3. Coloreado de los objetos

Pero también queremos visualizar los objetos encontrados. Para esto utilizamos una función de visualización auxiliar que llamaremos:

- `RGB = label2rgb(label_image)`

De esta forma podemos visualizar la imagen de etiquetas mediante cv2.imshow(label2rgb(label_image)). La idea consiste en considerar las etiquetas como el valor de color (Hue) de una nueva imagen HSV, pero

escalando los valores de las mismas al rango 0::127, pues 128 es el valor máximo del canal HUE. Las otras dos componentes (S y V) se pondrán al máximo valor (255). Luego se convierte esa imagen a BGR y la retornamos, para mostrarla en la ventana con cv2.imshow(). El código sería el siguiente:

```
def label2rgb(label_img):
    label_hue = np.uint8(179*(label_img)/np.max(label_img))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
      # Convertir a BGR
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
      # Poner el fondo a negro. El fondo son los píxeles con etiqueta 0
    labeled_img[label_ids==0] = 0
    return labeled_img
```

El resultado se muestra en la figura 6.

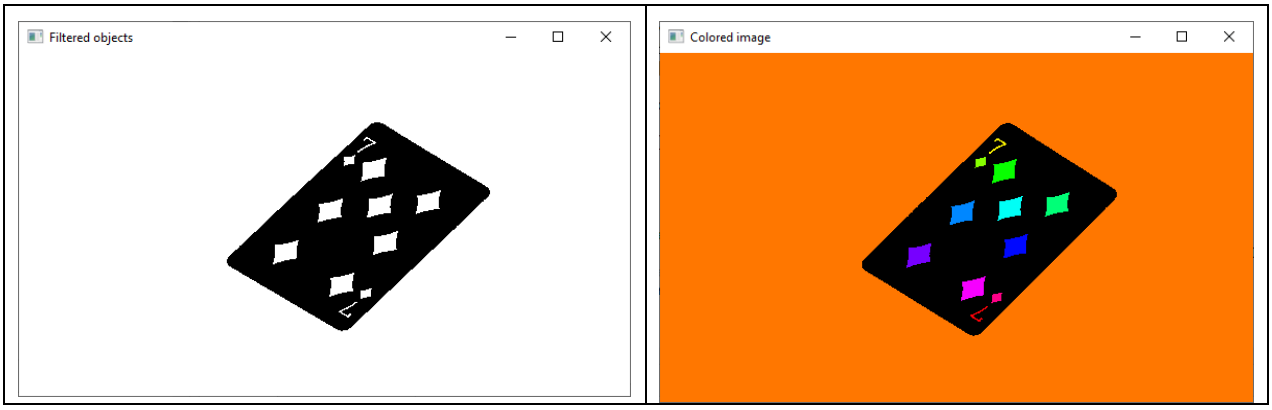

Figura 6 – (a) Imagen binaria (b) Resultado de colorear los objetos.

► *Ejercicio 7 – Visualización de etiquetas en color*

1. Añada la función anterior al archivo '*componentes_conectadas.py*' y muestre la imagen de etiquetas coloreadas en una nueva ventana, además de las anteriores.

2. Pruebe con distintas imágenes de la carpeta de cartas de póker.

La conclusión de todo lo visto anteriormente es que podemos INDIVIDUALIZAR los objetos obtenidos en la segmentación y los tendremos en una imagen de etiquetas, o una imagen binaria, que usaremos más adelante para extraer características.

4. Entrega de resultados

Como en las prácticas anteriores, para entregar los resultados haga un documento (Word o similar) con los distintos ejercicios incluyendo, en cada uno, lo siguiente:

- Enunciado del ejercicio
- Script de Python (copy – paste)
- Imágenes de los resultados (menú *Edit->Copy figure* en la ventana y paste en el Word)
- Comentarios personales sobre el ejercicio y los resultados (si procede)

Incluya, en la página inicial, el título de la práctica y el nombre del/los alumnos implicados. Convierta el archivo a pdf y súbalo a PoliformaT al espacio compartido de cada alumno implicado.