Visión por Computador (VxC)

Grado en Informática Industrial y Robótica (GIROB)

Práctica 6:

Extracción de características

Contenido:

1. Introducción
2. Extracción de las cartas
3. Orientación de las cartas
4. Extracción de los objetos de cada carta: Motivos
5. Características de los motivos
6. Sobre el trabajo2D

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

En esta práctica vamos a estudiar las funciones de Python para extracción de características, con especial interés en las características geométricas obtenidas de objetos en imágenes binarias. Este tipo de características son las que vamos a utilizar en la aplicación básica para el trabajo de Visión 2D sobre la identificación de las cartas de póker. El código desarrollado en esta práctica servirá para el citado trabajo y no será necesario realizar un documento de respuestas de esta práctica.

Objetivos

- Segmentar las imágenes de las cartas de póker para individualizar cada carta.
- Describir las funciones que tiene **OpenCV** para hacer operaciones de extracción de características en imágenes binarias.
- Preparación del código básico para el trabajo 2D.

Material

- Scripts de **OpenCV**: *segmentar_imagen.py.*
- Archivos de imagen de las cartas de póker, utilizadas en las prácticas anteriores.

1. Introducción

En la práctica anterior se estudiaron varias técnicas de umbralización y de segmentación por color, cuyo resultado final era una imagen BW – 0: fondo 255: objeto – , que representaba el resultado de la segmentación. Al finalizar dicha práctica, se describió cómo contar y etiquetar los objetos encontrados, mediante las funciones cv2.connectedComponents() y cv2.connectedComponentsWithStats(), y cómo mostrar el resultado mediante label2rgb().

Ahora vamos a poner todo junto en una función para segmentar una imagen, que contendrá una o varias cartas de póker, y devolver una matriz con todos los objetos encontrados. El esqueleto de dicha función lo tenemos en el script ***segmentar_imagen.py*** disponible en PoliformaT.

Para este código se emplea una técnica de umbralización global (ejercicios 2 y 5 - Práctica 4). En dicha técnica se obtenía un umbral global, mediante el histograma, que permitía umbralizar la imagen completa. Lo que realmente hacíamos es aprovechar el tipo de imágenes sobre las que trabajamos – cartas de póker – donde las cartas se disponen sobre un fondo verde – tapete – y cada carta tiene unos objetos de color – negro o rojo - sobre el fondo blanco de la carta. En realidad, lo que hacíamos era distinguir el fondo blanco de la carta del resto, pero al utilizar la opción `cv2.THRESH_BINARY_INV`, lo que obtenemos son el fondo global y todos los objetos del interior de la carta a blanco (255), y la carta en sí en negro (0), como se aprecia en la figura 1-a.

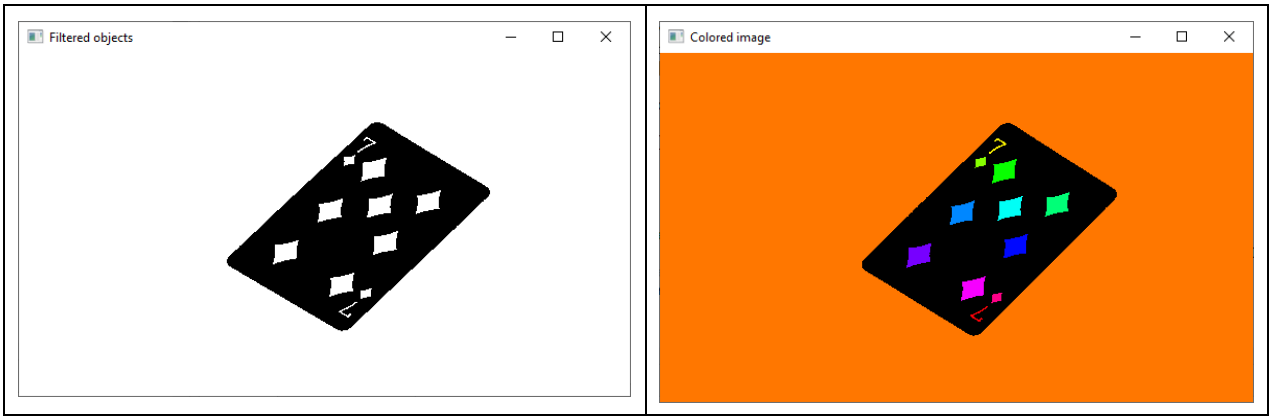

Figura 1 – (a) Imagen binaria invertida (b) Resultado de colorear los objetos.

La figura 1.6 muestra el resultado de colorear todos los objetos encontrados. En este contexto, podríamos eliminar el fondo de la imagen, por ser un objeto de área muy grande, y quedarnos sólo con los objetos del interior (rombos y números) que es lo que buscamos. Pero esto funcionaría bien si sólo hubiera una carta en la imagen, pero en realidad puede haber varias cartas. Por este motivo, nuestro primer propósito es encontrar e individualizar las cartas (array Cartas) y, a ser posible, colocarlas todas en la misma posición (vertical). Luego nos preocuparemos en encontrar los objetos (array Motivos) que contiene en su interior.

► *Ejercicio 1 – Segmentar la imagen*

1. Tome de PoliformaT el archivo ***segmentar_imagen.py***. Lea el código para entender lo que hace.

2. Ejecute al script con el conjunto de imágenes de cartas de póker y compruebe cómo funciona para imágenes con varias cartas.

2. Extracción de las cartas

El script anterior funciona bien extrayendo el fondo como un objeto (255), más los demás motivos dentro de la carta que también aparecen como el fondo. Pero nosotros queremos, en primer lugar, extraer las cartas. Para ello tenemos dos opciones:

- Utilizar la umbralización con `cv2.THRESH_BINARY`, de forma que las componentes conectadas serán las de los objetos carta.
- Utilizar la misma opción del scritp anterior (`cv2.THRESH_BINARY_INV`), pero utilizar la imagen umbralizada (255-thresh1) para obtener sólo las cartas.

► *Ejercicio 2 – Segmentar las cartas*

1. Copie el script de ***segmentar_imagen.py*** en otro archivo denominado ***segmentar_cartas.py.***

2. Cambie la opción de umbralilzación por `cv2.THRESH_BINARY` y compruebe el resultado.

3. Pruebe con la opción de umbralización inicial, pero obtenga las componentes conectadas de 255-thresh1. Compruebe el resultado con todas las cartas de póker.

En la primera opción umbralizamos el blanco de la carta y la salida es: fondo negro=0, objeto blanco=255. En el segundo caso umbralizamos el fondo y obtenemos: fondo blanco = 255 y objeto negro = 0, pero esta versión es más apropiada pues los objetos que se incluyen son aquellos que han superado el filtro de tamaño, por lo que sólo aparecen los objetos muy grandes, sin los detalles interiores. Por esto emplearemos esta segunda opción. En general, si ponemos un valor mínimo de área de 300000 es muy probable que no aparezcan objetos interiores de la carta. Es conveniente imprimir las áreas de los objetos encontrados para ver si el valor de 300000 píxeles es apropiado.

Ahora vamos a individualizar las cartas. Para ello vamos a trabajar con la imagen '*output'*, creada con sólo cada objeto individual (label_ids == i) que ha superado el área mínima:

```
for i in range(1, totalLabels):
            # Área del objeto
        area = values[i, cv2.CC_STAT_AREA]

        if (area > 300000):  # Filtro de tamaño   NUEVA CARTA
           output = np.zeros(img_gray.shape, dtype="uint8")
           componentMask = (label_ids == i).astype("uint8") * 255
          output = cv2.bitwise_or(output, componentMask)
           print(area)
```

A continuación, en dicho código vamos a obtener los contornos, mediante la función cv2.findContour(), así como el *bounding box* del objeto, mediante la tupla 'values', como sigue:

```
# Contornos del objeto 'i' con área mayor que el mínimo indicado
        contours, jerarquia = cv2.findContours(output, cv2.RETR_EXTERNAL ,  cv2.CHAIN_APPROX_SIMPLE)

# Bounding box del objeto 'i'
          x1 = values[i, cv2.CC_STAT_LEFT]
          y1 = values[i, cv2.CC_STAT_TOP]
          w  = values[i, cv2.CC_STAT_WIDTH]
          h  = values[i, cv2.CC_STAT_HEIGHT]

          roi = img_gray[int(y1):int(y1+h), int(x1):int(x1+w)].copy()   # ROI de la imagen de gris del objeto
          rows, cols = roi.shape[:2]
```

► *Ejercicio 3 – Mostrar cada carta segmentada*

1. Del script de ***segmentar_cartas.py*** añada el código anterior y cambie la ventana *window_label* por la ventana *window_roi*. Muestre en dicha ventana la imagen *roi*.

2. Compruebe el resultado con distintas cartas.

Observe que al poner la opción cv2.RETR_EXTERNAL, sólo aparecerá el contorno más exterior del objeto, que será el contorno de la carta (contours[0]). Al obtener de la imagen de gris del trozo indicado por ese contorno sólo aparecerá la imagen de una carta individual (*roi*). Se hace una copia porque en caso contrario el *roi* es un puntero al mismo espacio en la imagen original de gris (img_gray). Cualquier cambio en el *roi* produce el mismo en el original.

Es posible que necesitemos la imagen origina (img) para obtener alguna característica de color. Por esto se propone el siguiente ejercicio.

► *Ejercicio 4 – Mostrar la carta segmentada en color*

1. Del script de anterior obtenga el mismo fragmento (*roi*) pero de la imagen original en color (*img*), y llámelo *roi_color*. Muestre esa imagen en alguna de las ventanas disponibles para comprobar que funciona bien para una carta.

2. Compruebe el resultado con distintas cartas.

Para poder manejar el conjunto variado de cartas que existen en la carpeta debemos poder hacer una lista con todas las cartas, definiendo unos datos a almacenar para cada una de ellas. Para ello vamos a definir un tipo de datos 'Card' que nos permita asignar campos con las variables pertinentes. Tenemos dos opciones: un 'Structured Array' de NumPy o una clase con miembros y métodos para manejar esas variables.

De momento optaremos por la segunda alternativa, definiendo la siguiente clase, que también incluye algunas constantes que podremos necesitar en el futuro:

```
class Card:
    # Suits. Palos de las cartas de póker
     DIAMONDS = 'Diamonds'   # Rombos
     SPADES = 'Spades'      # Picas
     HEARDS = 'Hearts'      # Corazones
     CLUBS ='Clubs'         # Tréboles
     # Figuras y cifras de las cartas de póker
     FIGURES = ('0','A','2','3','4','5','6','7','8','9','J','Q','K') # Se accede mediantge Carta.FIGURES[i]

     def __init__(self):  # Constructor
      self.cardId = 0
      self.realSuit = ''
      self.realFigure = ''
      self.predictedSuit = ''
       self.predictedFigure = ''
       bboxType = [('x', np.intc),('y',np.intc),('width',np.intc),('height',np.intc)]
       self.boundingBox = np.zeros(1, dtype=bboxType).view(np.recarray)
       self.angle = 0.0
       self.grayImage = np.empty([0,0], dtype=np.uint8)
       self.colorImage = np.empty([0,0,0], dtype=np.uint8)

     def __repr__(self): # Para imprimir el contenido
       rep = f"Card number: {str(self.cardId)} --  Real Suit/Figure:  {self.realSuit} / {self.realFigure} -- Predicted
Suit/Fifure: {self.predictedSuit} / {self.predictedFigure}"
       bb = f"Bounding Box: {str(self.boundingBox)}  Rect angle: {str(self.angle)}"
       ims = f"Gray image:  {str(self.grayImage.shape)}  Color image:  {str(self.colorImage.shape)}"
       new_line = "\n"
       return rep + new_line + bb + new_line + ims
```

► *Ejercicio 5 – Creación de una lista de cartas*

1. Inserte la clase 'Card' en el script anterior y observe los distintos atributos que contiene. Cuidado porque al cortar y pegar se pueden modificar el indentado del código.

2. Cree una variable global que sea una lista vacía (Cards = []), que contendrá las cartas que vamos creando mediante Cards.append(), y otra variable global (icard=0) para manejar el número de cartas que vamos a ir introduciendo.

3. Tras obtener los datos del *bounding box* y las imágenes *roi* y *roi_color*, cree una variable de tipo *Card* y añada los datos que contiene. Después inserte la carta en la lista *Cards*.

4. Observe que las distintas cartas que vamos segmentando se van introduciendo en la lista Cards. Para un posible uso posterior guarde esa lista mediante la función:
   np.savez('cartas.npz', Cartas=Cards)
   El nombre del archivo será '*cartas.npz*'. Compruebe el tamaño del archivo en la carpeta donde está el srcipt.

3. Orientación de las cartas

En las aplicaciones industriales es bastante común el tener que reorientar los objetos segmentados para colocarlos en una posición determinada, con el propósito de hacer posteriores comparaciones. En este caso no es imprescindible, pero podemos averiguar cómo lo haríamos.

En esta aplicación las cartas se obtienen mediante un *Bounding box* donde aparece la carta girada. El *Bounding box* es siempre un rectángulo en posición horizontal (up-right bounding rectangle). Pero es posible obtener otro rectángulo girado (*Rotated Rectangle*), de área mínima, que encuadra el contorno del objeto deseado. La figura 2 muestra la diferencia entre ambos.

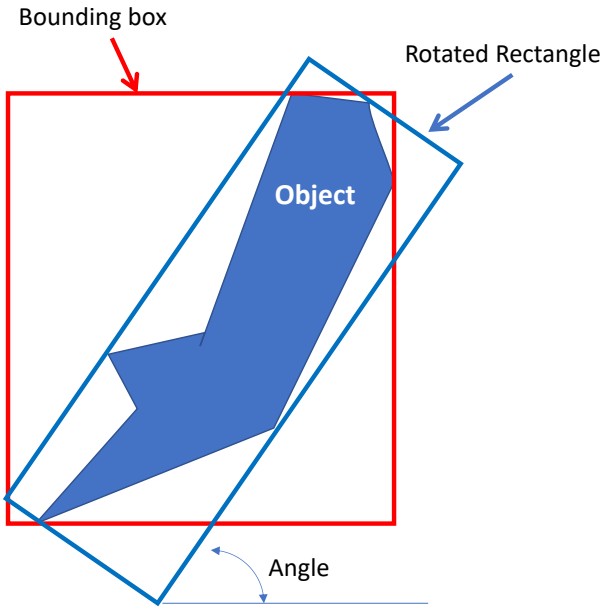

Figura 2 – Diferencias entre el *Bounding Box* y el *Rotated Rectangle* de un objeto.

► *Ejercicio 6– Reorientación de las cartas*

1. En el script anterior, inserte la función minRect = cv2.minAreaRect(contours[0]). Con este rectángulo mínimo obtenga una matriz de rotación (cv2.getRotationMatrix(…)) y aplique la rotación mediante cv2.warpAffine(..).

2. Aplique la rotación a la imagen de gris (*roi*) o la imagen en color (*roi_color*) y vea los resultados en alguna de las ventanas disponibles.

3. Obtenga el ángulo de giro de cada rectángulo mínimo e inclúyala en la estructura de datos de la carta.

4. Extracción de los objetos de cada carta: Motivos

En script de ***segmentar_carta.py*** hemos conseguido extraer cada objeto '*Card*' presente en la imagen, y hemos guardado su imagen de niveles de gris (*roi*), una versión en color de la misma (*roi_color*), y algunos atributos de la misma. La versión en color es interesante pues quizás necesitemos más adelante la información de color. De forma opcional, hemos conseguido una versión con la carta aproximadamente en vertical.

Lo que queremos hacer, a continuación, es identificar la figura y el palo de cada carta. Los palos son de motivos negros: **Clubs** (Tréboles**)** y **Spades** (Picas); y de motivos rojos: **Diamonds** (Diamantes) y **Hearts** (Corazones). Cada palo está formado por 13 figuras, de las cuales 9 son numerales y 4 literales, que son: **0, A, 2, 3, 4, 5, 6, 7, 8, 9, J, Q, K**. Las cartas literales son **A** (As: ace), **J** (Jota: Jack), **Q** (Reina: queen) y **K** (Rey: king). Esta información ya la tenemos en la estructura de la clase Card. A todos estos elementos los llamaremos '**Motivos'**.

Para hacer esto deberemos analizar los motivos que contiene cada carta y, en base a lo que encontremos, asignar un palo o una figura a cada motivo. Emplearemos las siguientes etiquetas:

```
card_labels = ('Diamonds','Spades','Hearts','Clubs','0','2','3','4','5','6','7','8','9','A','J','Q','K','Others')
```

Más adelante, analizaremos los motivos que encontremos en cada carta y, en base a ellos, asignaremos un palo y una figura a la carta completa, que serán los card.realSuit y card.realFigure.

Para obtener los motivos vamos a emplear la misma técnica explicada en los apartados anteriores, consistente en analizar la imagen de gris de la carta (*roi*) mediante umbralización global y obtendremos la imagen binaria BW. Los objetos que hay dentro de la carta son parte del fondo, pero son objetos individuales – motivos de picas, corazones, ..., números y posiblemente algún punto aislado de ruido o algunos detalles de las figuras (Others). 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. Luego analizaremos las componentes conectadas de esta imagen BW y eliminaremos la primera componente, que se corresponde con el fondo, quedándonos con el resto de objetos. Véase la figura 3 para recordar el resultado.

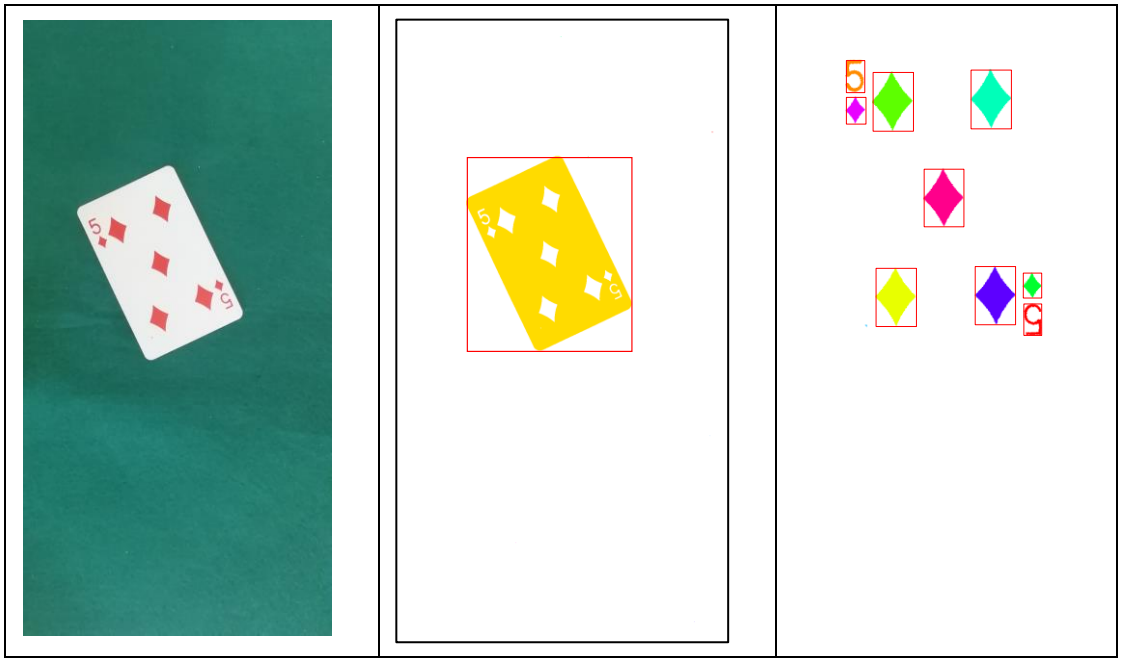

Figura 3 - Resultado de la función *segmentar_objetos_carta*(). (a) Imagen original (b) Imagen segmentada (c) Motivos dentro de la carta.

► *Ejercicio 7– Obtención de los motivos de cada carta*

1. En el script anterior, inserte la función segmentar_objetos_carta(Carta). El objeto a pasar como argumento es uno de clase Card y la función debe obtener la lista de motivos que contiene mediante las acciones de umbralización y componentes conectadas, como en los ejercicios anteriores.

2. En la clase Card inserte un nuevo atributo que será una lista de motivos. Probablemente habrá que definir una clase Motif que contenga, como atributos, todas las características que vamos a guardar de cada motivo.

3. Visualice los motivos encontrando sus contornos mediante (cv.findContours(..)) y visualícelos mediante(cv2.drawContours(..)) en cualquiera de las ventanas disponibles que muestren la carta.

5. Características de los motivos

En la función *segmentar_objetos_carta()* hemos conseguido extraer y apuntar cada objeto 'motivo' presente en la carta y les hemos añadido el conjunto de características geométricas que nos ha proporcionado las funciones anteriores, como el área, perímetro, contorno, etc. Algunas de estas características serán útiles para el proceso subsiguiente de reconocimiento, pero faltan algunas más. Especialmente características que sean invariantes o de color. Por eso se recomienda estudiar las funciones que ofrece OpenCV.

https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html

► *Ejercicio 8 –Características de los motivos*

1. En la función anterior, dentro de la clase Motif, inserte algunas variables con características útiles. Estudie las funciones: cv2.contourArea, cv2.moments, cv2.HuMoments, cv2.fitEllipse, …

2. Busque características de color de los motivos (p.e. color medio), dentro de la imagen *roi_color* y añádalos a las estructuras de datos. Intente que todas las características encontradas estén en un np.ndarray de valores reales que será nuestro vector de características.

3. Pruebe con distintas cartas.

6. Sobre el trabajo 2D

Todas las funciones obtenidas en esta práctica son la base para la realización del trabajo 2D de la asignatura, utilizando el *Dataset* de cartas de póker. Si se propone otra aplicación distinta, la operativa aquí indicada también puede ser de utilidad.
En la siguiente práctica abordaremos el problema de reconocimiento de formas para identificar los motivos de cada carta, y con ellos, dar una indicación de figura y palo de cada carta.
No hay que entregar ninguna información de esta práctica, pues lo ahora desarrollado formará parte del trabajo 2D, que se entregará al final del curso.