Universidad Nacional de San Luis

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS Y NATURALES
DEPARTAMENTO DE ELECTRÓNICA

INGENIERÍA ELECTRÓNICA CON ORIENTACIÓN EN SISTEMAS DIGITALES

IMPLEMENTACIÓN DE UNA FUNCIÓN PARA ENCONTRAR LOS BORDES EN OPENCV-PYTHON

Trabajo Práctico Nº 11 - Tema Nº 3

Materia:
DSP II

Docentes:
Mg. Ing. Ricardo Petrino
Ing. Jesús Garcia
Aux. Gonzalo Bertello

Autores:
Pablo Ezequiel Córdoba
Paul Andrés Romero Coronado

1.º cuatrimestre de 2022

Índice

1. Introducción

Se tiene como objetivo de esta investigación la implementación de una función en OpenCV-Python para encontrar los bordes de objetos en imágenes binarias y guardar la las coordenadas de los mismos en matrices.

2. Desarrollo

Para lograr el objetivo se hizo uso de dos funciones ya existentes de la libreria OpenCV [1]:

- *cv2.findContours*: Encuentra los bordes de una imagen.
- *cv2.drawContours*: Dibuja los bordes anteriormente encontrados en la imagen.

2.1. cv2.findContours

La función cv2.findContours requiere de los siguientes parámetros [2]:

- Parámetros de entrada:

  InputArray image: imagen binarizada a la que se busca encontrar los bordes.

  int mode: algoritmo de recuperación de bordes. Los modos posibles son:

  1. RETR_EXTERNAL: se recuperan únicamente los bordes externos.
  2. RETR_LIST: se recuperan todos los bordes sin una jerarquía que los relacione
  3. RETR_CCOMP: se recuperan los bordes externos y los bordes internos (huecos), organizando una jerarquía de dos niveles, siendo el primer nivel los bordes externos y el segundo nivel los internos.
  4. RETR_TREE: se recuperan todos los bordes y se reconstruye una jerarquía completa de bordes interconectados.

  int method: método de aproximación de bordes. Para este parámetro se utilizará únicamente el método cv2.CHAIN_APPROX_SIMPLE debido al poco procesamiento que este requiere en comparación con el método cv2.CHAIN_APPROX_NONE. Donde el primero descarta puntos del contorno que son redundantes, ahorrando memoria.

- Parámetros de salida:

  OutputArrayOfArrays contours: Matriz donde se encuentran las coordenadas de los bordes.

  OutputArray hierarchy: La jerarquía de los contornos, la topología de la imagen.

En la Figura 1 se muestra el borde del círculo en color negro. Al utilizar cv2.CHAIN_APPROX_NONE, se calcularon 678 puntos.

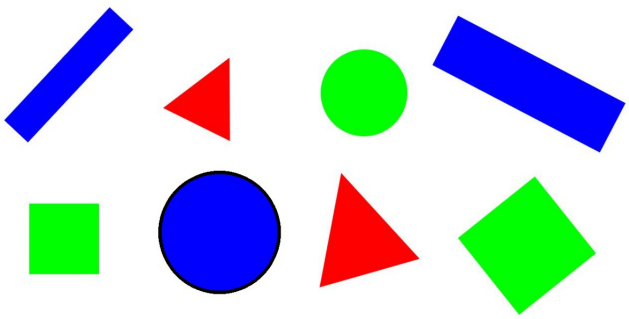


Figura 1 – Bordes utilizando cv2.CHAIN_APPROX_NONE. 678 puntos.

En la Figura 2 se muestra el borde del circulo al utilizar cv2.CHAIN_APPROX_SIMPLE, se calcularon 336 puntos.

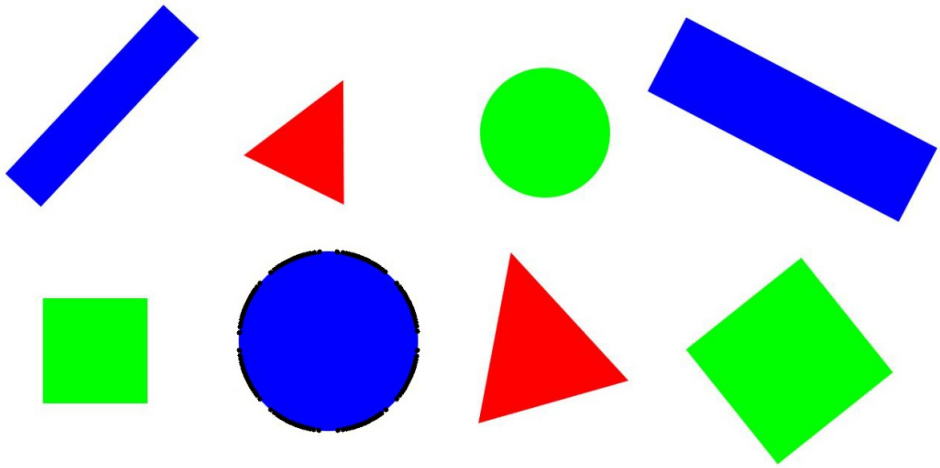


Figura 2 – Bordes utilizando cv2.CHAIN_APPROX_SIMPLE. 336 puntos.

En la Figura 3 se muestra todos los bordes detectados utilizando cv2.CHAIN_APPROX_SIMPLE.

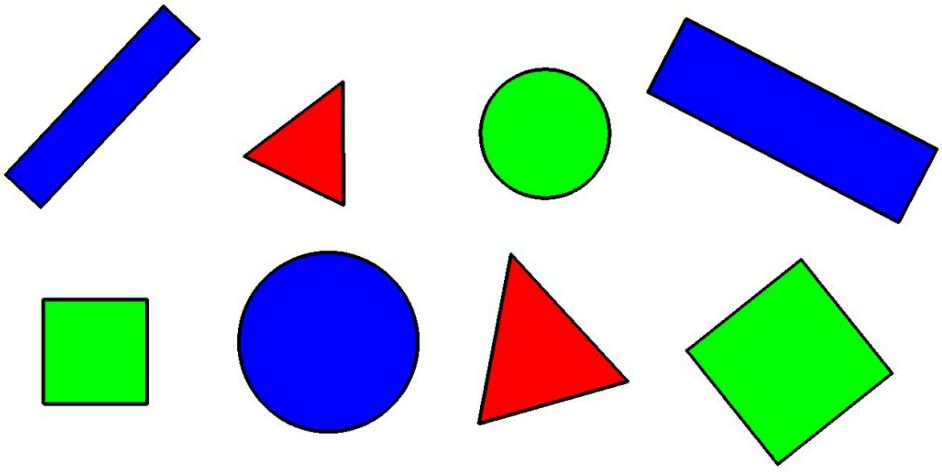


Figura 3 – Todos los bordes detectados. cv2.CHAIN_APPROX_SIMPLE

Se puede observar como reduciendo la cantidad de puntos, es suficiente para graficar todos los bordes de una imagen. De esta manera, se ahorra en procesamiento y en memoria.

Por último, se requiere que la imagen ya esté binarizada a la hora de utilizar cv2.findContours.

2.2. cv2.drawContours

La función cv2.drawContours requiere de los siguientes parámetros [3]:

- Parámetros de entrada/salida:

  InputOutputArray image: imagen a la que se le dibujará los bordes ya detectados.

- Parámetros de entrada:

  InputArrayOfArrays contours: Anteriormente explicado.

  int contourIdx: indica la cantidad de bordes a dibujar. Si se lo define como "-1" dibuja todos los bordes anteriormente detectados.

  const Scalar & color: indica el color de los bordes.

  int thickness: indica el grosor de los bordes. Si el numero es negativo dibuja los bordes interiores.

2.3. Función encontrar_bordes

Para poder encontrar los bordes tal como lo requiere la consigna, se decidió realizar una función llamada *encontrar_bordes* para ayudar al usuario. La misma posee la siguiente estructura:

```
encontrar_bordes(imagen,tipo_umbral,tipo_contorno,color_borde,ancho_borde)
->imagen_bordeada, bordes
```

Donde:

- imagen: Imagen fuente en escala de gris o en color.
- tipo_umbral (string): Como se mencionó antes, la imagen se necesita umbralizar, y la misma se realiza con la la umbralización Otsu, con este parámetro se elige el tipo de binarizado, esto es:
  - "b"− binaria.
  - "binv"− binaria inversa.
  - "t"− truncado.
  - "tz"− to zero.
  - "tzinv"− to zero inversa.
- tipo_contorno (string): Es el tipo de contorno que se desee detectar:
  - "ext"− bordes externos.
  - "list"− todos los contornos (sin relacion de jerarquia).
  - "ccomp"− contornos externos e internos (huecos)[jerarquia de 2 niveles].
  - "tree"− todos los contornos (con relación de jerarquía).
- color_borde (string): El color del borde:
  - "r"− rojo.
  - "g"− verde.
  - "b"− azul.
  - "y"− amarillo.
  - "m"− magenta.
  - "o"− naranja.
  - "yg"− amarillo_verde.
  - "cg"− cyan_verde.
  - "cb"− cyan_azul.
  - "p"− morado.
  - "rm"− rosado/fucsia (rojo_magenta).
  - "k"− negro.
- ancho_borde (int): El ancho del borde que se desee dibujar al objeto.
- imagen_bordeada: Imagen de salida con los bordes marcados.
- bordes: Matriz de salida donde contiene las coordenadas en las filas y columnas de los objetos detectados.

En el anexo de este documento se adiciona un ejemplo de uso de la función.

2.4. Libreria RC

Para utilizar la función *encontrar_bordes*, previamente se necesita instalar una librería [4] realizada por los autores de este documento, por lo que se deben seguir las siguientes instrucciones, las cuales se encuentran en el proyecto de Github [5].

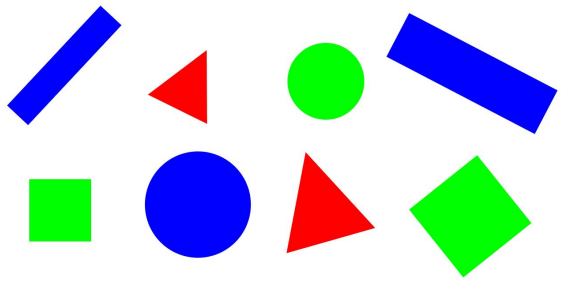
1. Ir a `https://pypi.org/project/encontrar-bordes/0.0.10/` y copiar con Copy to clipboard "pip install encontrar-bordes==0.0.10", para evitar errores.
2. Abrir "Anaconda prompt(anaconda3)", luego pegar e instalar. Nota: *En caso que aparezca error que no encuentre la librería, volver a realizar el paso 1.*
3. Listo. Ahora puede importar la libreria con el comando *from RC_lib import encontrar_bordes* en su script.

Luego de haber realizado los pasos previos, se debe ver la pantalla que se muestra en la Figura 4.

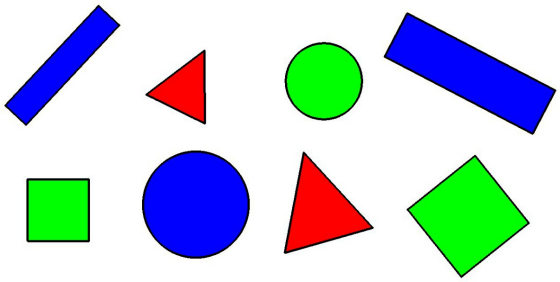
```
(base) C:\Users\EZE>pip install encontrar-bordes==0.0.10
Collecting encontrar-bordes==0.0.10
  Downloading encontrar_bordes-0.0.10-py3-none-any.whl (3.2 kB)
Requirement already satisfied: numpy in c:\users\eze\anaconda3\lib\site-packages (from encontrar-bordes==0.0.10) (1.20.3)
Requirement already satisfied: cv2 in c:\users\eze\anaconda3\lib\site-packages (from encontrar-bordes==0.0.10) (2.0.2)
Installing collected packages: encontrar-bordes
  Attempting uninstall: encontrar-bordes
    Found existing installation: encontrar-bordes 0.0.9
    Uninstalling encontrar-bordes-0.0.9:
      Successfully uninstalled encontrar-bordes-0.0.9
Successfully installed encontrar-bordes-0.0.10
```

Figura 4 – Anaconda Prompt.

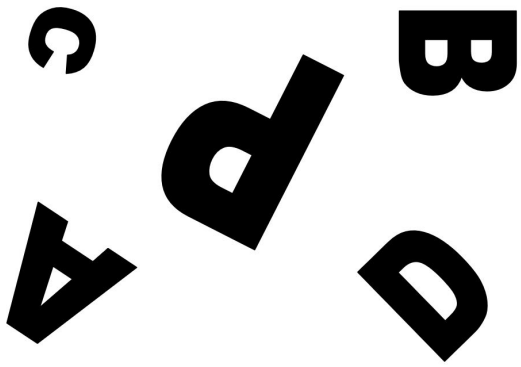
Como se mencionó, en el anexo se adiciona el código de ejemplo de uso y este también se encuentra en Github. Con el mismo, se obtienen los resultados que se muestran en la Figura 5.



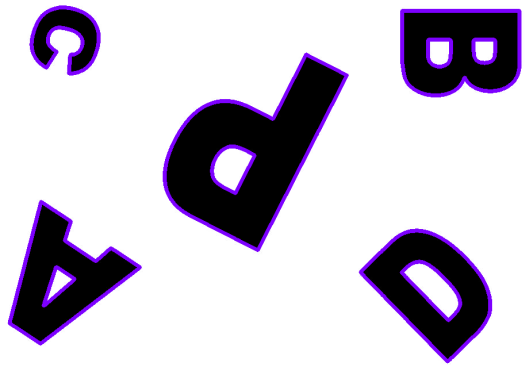
(a) Imagen original propuesta.



(b) Bordes de la imagen propuesta.



(c) Imagen alternativa.



(d) Bordes de la imagen alternativa

Figura 5 – Resultado código ejemplo.

3. Conclusión

El único problema de esta nueva función es que no puede detectar únicamente los bordes interiores (huecos), solo los externos o ambos a la vez. Esto se debe a que en la función *cv2.findContours()*, su parámetro *mode*, el que indica el algoritmo de recuperación de bordes, carece de un modo que pueda detectar solamente esos bordes. Sin embargo, esta función sigue logrando su objetivo de encontrar bordes, externos e internos.

Se puede concluir con la finalización de esta investigación de manera satisfactoria, ya que como se ha mostrado, la función implementada mediante librería posee mucha flexibilidad, sin necesidad de que la imagen introducida posea unas características exclusivas. Por lo tanto, este fue un buen ejercicio para enriquecer la programación en lenguaje Python, aprendiendo a crear funciones y librerías propias que puedan ser accedidas por cualquier persona que se interese en lo que estas proveen.

A. Anexo

A.1. Función

```
import cv2
import numpy as np


def encontrar_bordes(imagen,tipo_umbral,tipo_contorno,color_borde,ancho_borde)
    :

"""
encontrar_bordes(imagen,tipo_umbral,tipo_contorno,color_borde,ancho_borde)
->imagen_bordeada, bordes

Funcion realizada por los alumnos:
Cordoba Pablo Ezequiel y Romero Coronado Paul Andres

Esta funcion encuentra los bordes de una imagen.
Devuelve la imagen con los bordes dibujados y una matriz con las coordenadas
de los bordes

imagen: imagen fuente
tipo_umbral (string): "b" - binaria
"binv" - binaria inversa
"t" - truncado
"tz" - to zero
"tzinv" - to zero inversa

tipo_contorno (string): "ext" - bordes externos
"list" - todos los contornos (sin relacion de jerarquia)
"ccomp" - contornos externos e internos (huecos)[jerarquia de 2 niveles]
"tree" - todos los contornos (con relacion de jerarquia)

color_borde (string): "r" - rojo
"g" - verde
"b" - azul
"y" - amarillo
"c" - cyan
"m" - magenta
"o" - naranja
"yg" - amarillo_verde
"cg" - cyan_verde
"cb" - cyan_azul
"p" - morado
"rm" - rosado/fucsia (rojo_magenta)
"k" - negro
ancho_borde (int): ancho del borde a dibujar
"""
# Si la imagen es a color,
# len(imagen.shape[:]) = 3
# si es en escala de grises
# len(imagen.shape[:]) = 2
#----------
```

```
50      escala_color = len(imagen.shape[:])
51      #----------
52
53      img_aux = np.copy(imagen)
54      if (escala_color==3):
55      imagen1 = cv2.cvtColor(img_aux, cv2.COLOR_BGR2GRAY)
56      else:
57      imagen1 = np.copy(imagen)
58      #Una vez en escala de gris, se binariza la imagen mediante binarizacion OTSU.
59      #Por eso, la variable tipo_umbral indica el tipo de umbralizacion que se
            quiere aplicar
60
61      #Umbralizacion binaria
62      if(tipo_umbral=='b'):
63      ret,thresh = cv2.threshold(imagen1,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
64
65      #Umbralizacion binaria inversa
66      elif (tipo_umbral=='binv'):
67      ret,thresh = cv2.threshold(imagen1,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU
            )
68
69      #Umbralizacion Trunc
70      elif(tipo_umbral=='t'):
71      ret,thresh = cv2.threshold(imagen1,0,255,cv2.THRESH_TRUNC+cv2.THRESH_OTSU)
72
73      #Umbralizacion Tozero
74      elif(tipo_umbral=='tz'):
75      ret,thresh = cv2.threshold(imagen1,0,255,cv2.THRESH_TOZERO+cv2.THRESH_OTSU)
76
77      #Umbralizacion Tozero inversa
78      elif(tipo_umbral=='tzinv'):
79      ret,thresh = cv2.threshold(imagen1,0,255,cv2.THRESH_TOZERO_INV+cv2.THRESH_OTSU
            )
80      #En el caso de que la imagen de entrada ya se encuentre binarizada, poner
81      #en la variable tipo_umbral "none" para no aplicar binarizacion.
82
83      #Se encuentran los contornos de la imagen:
84
85      if (tipo_contorno=='ext'): #Solo los contornos externos
86      contornos, hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.
            CHAIN_APPROX_SIMPLE)
87
88      elif (tipo_contorno=='list'): #Todos los contornos (sin relacion de jerarquia)
89      contornos, hierarchy = cv2.findContours(thresh,cv2.RETR_LIST,cv2.
            CHAIN_APPROX_SIMPLE)
90
91      elif (tipo_contorno=='ccomp'): #Contornos externos e internos (huecos)[
            jerarquia de 2 niveles]
92      contornos, hierarchy = cv2.findContours(thresh,cv2.RETR_CCOMP,cv2.
            CHAIN_APPROX_SIMPLE)
93
94      elif (tipo_contorno=='tree'): #Todos los contornos (con relacion de jerarquia)
95      contornos, hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.
```

```
        CHAIN_APPROX_SIMPLE)

    if (escala_color==2):
    img_aux = cv2.cvtColor(imagen1, cv2.COLOR_GRAY2BGR)

    #Por ultimo, se dibuja los contornos de la imagen fuente,
    #se necesita que previamente este en color

    if (color_borde=='r'): #Rojo
    cv2.drawContours(img_aux, contornos, -1, (0,0,255), ancho_borde)

    elif (color_borde=='g'): #Verde
    cv2.drawContours(img_aux, contornos, -1, (0,255,0), ancho_borde)

    elif (color_borde=='b'): #Azul
    cv2.drawContours(img_aux, contornos, -1, (255,0,0), ancho_borde)

    elif (color_borde=='y'): #Amarillo
    cv2.drawContours(img_aux, contornos, -1, (0,255,255), ancho_borde)

    elif (color_borde=='c'): #Cyan
    cv2.drawContours(img_aux, contornos, -1, (255,255,0), ancho_borde)

    elif(color_borde=='m'): #Magenta
    cv2.drawContours(img_aux, contornos, -1, (255,0,255), ancho_borde)

    elif(color_borde=='o'): #Naranja
    cv2.drawContours(img_aux, contornos, -1, (0,125,255), ancho_borde)

    elif(color_borde=='yg'): #Amarillo-Verde
    cv2.drawContours(img_aux, contornos, -1, (0,255,125), ancho_borde)

    elif(color_borde=='cg'): #(Cyan-Verde)
    cv2.drawContours(img_aux, contornos, -1, (125,255,0), ancho_borde)

    elif(color_borde=='cb'): #(Cyan-Azul)
    cv2.drawContours(img_aux, contornos, -1, (255,125,0), ancho_borde)

    elif(color_borde=='p'): #Morado
    cv2.drawContours(img_aux, contornos, -1, (255,0,125), ancho_borde)

    elif(color_borde=='rm'): #Rosado/Fucsia (Rojo-Magenta)
    cv2.drawContours(img_aux, contornos, -1, (125,0,255), ancho_borde)

    elif(color_borde=='k'): #Negro
    cv2.drawContours(img_aux, contornos, -1, (0,0,0), ancho_borde)

    return img_aux,contornos
```

A.2. Código de ejemplo

```
    import cv2
    from RC_lib import encontrar_bordes
```

```
3
4       #Funciona tanto para imagenes en color como en escala de gris
5       imagen_original=cv2.imread("imagen4.jpg",1)
6
7       imagen_original2=cv2.imread("imagen5.JPG",0)
8
9
10      imagen_final,bordes=encontrar_bordes(imagen_original,"binv","ext","k",5)
11
12      imagen_final2,bordes2=encontrar_bordes(imagen_original2,"binv","tree","p",5)
13
14
15      print("Contorno:",bordes)
16      cv2.imshow("Imagen original",imagen_original)
17      cv2.imshow("Imagen original2",imagen_original2)
18      cv2.imshow("Bordes encontrados",imagen_final)
19      cv2.imshow("Bordes internos y externos",imagen_final2)
20      cv2.waitKey()
21      cv2.destroyAllWindows()
```

Referencias

[1] Contours: Getting started
https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

[2] Parámetros de la función cv2.findContours
https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gadf1ad6a0b82947fa1fe3c3d497f260e0

[3] Parámetros de la función cv2.drawContours
https://docs.opencv.org/4.x/d6/d6e/group__imgproc__draw.html#ga746c0625f1781f1ffc9056259103edbc

[4] Cómo crear una librería de Python
https://antonio-fernandez-troyano.medium.com/crear-una-libreria-python-4e841fbd154f

[5] Proyecto de Github
https://github.com/Ezequiel2003/UNSL—Proyecto-DSP2—Cordoba-Romero