Proyecto 2 - Figuras

Definición del problema

- ¿Qué queremos obtener?

Dada una imagen, reconocer las Figuras de la Imagen y su color bajo las siguientes categorías. Categorías: Cuadriláteros (C) , Triángulos(T), Círculos (O), Otros (X)

Entrada

- ¿Con qué datos contamos ?
    - Imagen. Formato: Mapa de bits (.BMP)
        - **Tamaño**: Lectura
        - **Formato**: .BMP
        - **Tipo**: matrix
        - **Fuente:** ruta de la imagen desde la terminal
- ¿ Son suficientes?
    - Por el momento si pues solo tenemos que identificar las figuras de la imagen.
- ¿Qué hace que el resultado obtenido sea el que queremos?
    - En los datos del proyecto nos especifican el Output, por lo que estamos seguros de cuál es el resultado esperado.
- ¿Qué operaciones se tienen que hacer para que el resultado sea el esperado?
    - Recibir la Imagen
    - Procesar la imagen
        - Convertir la imagen a **matriz**
        - Reconocer los **diferentes colores** que hay en la imagen
            - Podemos guardarlos en un set
        - **Aislar un color** para procesar la figura
        - Obtener el **centro** de la figura, al sacar el promedio de las coordenadas del contorno a la figura
        - Sacar una función que cuente la **distancia del centro al contorno**.
        - Función que reconozca el número de **máximos totales** = f(n)
        - Regresar la categoría de a la que pertenecen las figuras en la imagen
            - Si f(n) == 4, Figura = C
            - Si f(n) == 3, Figura = T
            - Si f(n) > 9, Figura = O
            - Else: Figura = X
    - Regresar Salida

Requisitos Funcionales

¿Qué debe dar como salida dado un cierto tipo de entrada? ¿Qué se debe llevar a cabo?

- Dada una imagen (.BMP) clasificar las figuras de la imagen bajo cuatro categorías; Círculo, Triángulo, Cuadrilátero y Otros.

Salida

- Categoría de cada una de las figuras en la imagen {Color, Categoría}
    - **Tipo:** {Hexadecimal, Carácter}
    - **Formato:** {String, Char}
    - **Tamaño:** El color tiene 6 caracteres, la categoría 1
    - **Cantidad:** número de figuras
    - **Destino:** Terminal

Requisitos no funcionales

- Eficiencia

Para medir la eficiencia lo separaremos en las funciones principales.

GetColors. Recorre todo el mapa de bits de la imagen, así que la complejidad depende del tamaño de la imagen. **O(n*m)** donde n y m con el ancho y largo de la imagen.

Get Centers. Obtiene los centros de las figuras de la imagen, primero filtrando la imagen por el colores de la imagen. Luego obtiene las coordenadas del contorno de la figura y saca el promedio para encontrar el centro . Así que la complejidad es lineal bajo el tamaño de la imagen, multiplicado por el número de figuras. **O(n*m*f)**

Distancia. Toma la distancia del centro al contorno, la complejidad depende del número de puntos en el contorno.

- Tolerancia a fallas

Para el funcionamiento correcto del programa estamos suponiendo que cada figura tiene un color único y distinto al fondo. El programa puede fallar cuando la figura está muy distorsionada o muy pequeña y puede que cuente picos extra.

- Amigabilidad

El programa es fácil de usar para personas que saben usar la terminal, pues solo hace falta poner la dirección de la imagen.

- Escalabilidad

Se podría hacer una aplicación web para hacerla más escalable a todos los usuarios.

Arsenal

Lenguaje: Python
Bibliotecas:

- OpenCV
    - La utilizamos para leer las imagenes, filtar las figuras y obtener el contorno de las imágenes
- Matplot
    - Crear las gráficas de las distancias
- Scipy
    - Reconoce lod picos de la funcion de las distancias

IDE: PyCharm

Decidimos utilizar **Python** porque nos facilita la sintaxis y cuenta con la librería de Opencv para el procesamiento de la imagen, que nos permite convertir la imagen a una matriz y cambiar pixeles. Y utilizamos PyCharm porque es una IDE especializada para Python y con un entorno muy completo para realizar el proyecto.

Análisis del problema: componentes, atributos y comportamiento

Componentes:

- Imagen
    - Figuras
        - Color figura
        - Centro
        - Distancias Centro-Contorno
        - Número de picos

Como solo nos interesa el número de picos de cada figura, y distinguir cada figura por su color decidimos procesar cada figura una por una y solo guardar la información al procesarla. Por lo que no vimos necesaria la construcción de clases.

Modelo de datos

Imagen (**Matriz**): guardamos la imagen en un mapa de bits, para poder usar las funciones de la librería de OpenCV.

Colores (**Set**): guardamos en un set los diferentes de la imagen

Centros(**Array**): Guardamos los centros de todas las figuras en un arreglo

Distancia a los centros(**Matriz)**: cada arreglo de la matriz guarda las distancias del centro al contorno para cada figura

Pruebas Unitarias

- test_hex_to_rgb
    - Checa que la conversión de hexadecimal a RGB sea correcta
- test_hex_to_bgr
    - Checa que la conversión de hexadecimal a BGR sea correcta
- test_get_colors
    - Checa que los colores que se obtienen del método sean iguales a los de la imagen
- test_hex_color
    - Checa la conversión a hexadecimal sea correcta
- test inputs
    - Checa que los inputs(las imágenes) y los outputs (las categorías de las figuras) sean correctas

Pseudocódigo

Recibir la Imagen y convertirla en mapa de bits

- Utilizamos la función de *imread* de la librería de Opencv que lee la imagen la convierte en un mapa de bits

```
function leerImagen(file):
    return cv2.imread( “relative path” + file)
```

Reconocer los diferentes colores que hay en la imagen

- Iteramos sobre la matriz de pixeles y guardamos todos los colores que sean diferentes.

```
function get_colors(image):
    color_set = {}
    for i in  image:
        color_set.append(i)
```

Filtrar una figura de la imagen y encontrar los contornos

- Filtramos la imagen por los colores del set, y creamos una máscara de la imagen que solo guarde el color que queremos. Luego decidimos invertir la máscara para que el fondo fuera blanco y la figura fuera negra.

```
function mask(image):
    for i in colors_set:
        lower = color[i].toBGR
        upper= color[i].toBRG
        mask = cv2.inRange(img, lower, upper)
        inv_mask = cv2.bitwise_not(mask)
        cv2.findContours(inv_mask,cv2.RETR_TREE,
```

cv2.CHAIN_APPROX_NONE)

Obtener el centro de la figura

- Calculamos el centro de la figura haciendo el promedio de los píxeles del contorno.

```
function centers(contours):

    for point in contours:
        sumX += point.X
        sumY += point.Y
    averageX = sumX / len(contours)
    averageY = sumY / len(contours)
```

Distancia del centro al contorno

- Calculamos la distancia entre el centro y el contorno, con la distancia entre las coordenadas de los dos puntos.

```
function distance(center,contours):
    for point in contours:
        distancia = sqrt((pointX- centerX)^2 + (pointY-centerY)^2)
        distancias[figura].append(distancia)
```

Calcula los picos de las distancias de una figura

- Suaviza las distancias tomando el promedio de los datos que estén a distancia menor o igual 2 en el arreglo (abs(i - j) <= 2) y obtiene los picos de esa gráfica que estén a una distancia mayor que 9/10 veces la media mediante la funcion get_peaks de scipy.

```
function count_peaks(signal):
    smooth_data := []
    for i : = 0 … len(signal):
        smooth_data.add(average(signal[i-2] + signal[i-1] + signal[i] +
signal[i+1] + signal[i+2]))
    mean = average(smooth_data)
    peaks = find_peaks(smooth_data)
    peaks = filter(peaks, > mean)
    return len(peaks)
```

Regresa la categoría a la que pertenece la figura

- Regresa el color en hexadecimal de las figuras y la categoría a la que pertenecen dependiendo del número de picos

```
function categories(colors):
    for i in colors:
        if picos(colors[i] == 3):
```

```
        print(colors + ": " + T)
else if picos(colors[i] == 4):
        print(colors + " :" + C)
else if picos(colors[i] >  9):
        print(colors + ": " + O)

else:
        print(colors + ": " + X)
```

Mantenimiento

- Hacer una interfaz gráfica para mejorar la detección
- Hacer un algoritmo diferente para figuras pequeñas, que detecte el contorno de las figuras de otra manera sin usar la librería para mejorar la precisión.