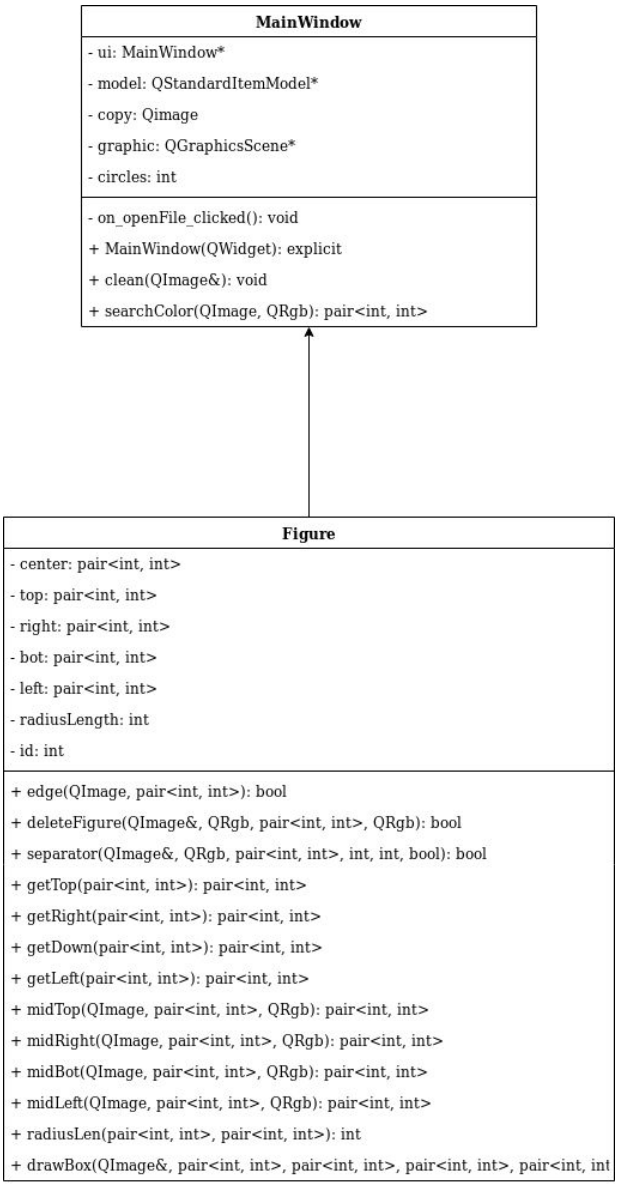
Localización de círculos

Jesús Uriel Guzmán Mendoza

I. Diagrama de clases propuesto



II. Objetivo

Diseñar un programa que tenga la capacidad de seleccionar y analizar imagenes que contienen círculos negros. El sistema computacional localizará las posiciones de las figuras que considere como círculos y los señalará gráficamente, también mostrará sus radios y las coordenadas en las cuales estos círculos fueron localizados. Todas aquellas figuras que tengan forma de dona y/o sean figuras incompletas serán ignoradas, y las figuras con forma de óvalo serán eliminadas.

III. Marco teórico

Búsqueda de profunidad (DFS) es un algoritmo para viajar a través de estructuras de datos de árboles y grafos. El algoritmo comienza en el nodo raíz (que puede ser cualquier nodo arbitrario) y expora tan lejos como sea posible a través de cada rama antes de hacer un retorno. Podemos representar una imagen como una estructura de datos parecida a un grafo en la cual los pixeles vecinos que sean del mismo color que el nodo raíz son nodos de un grafo conectado, y los vértices pueden ser interpretados como la distancia de exactamente un pixel desde el nodo siendo analizado y un nodo vecino que se encuentre en un píxel superior, derecho, inferior o izquierdo del pixel actual.

La búsqueda de profunidad en este escenario nos permite identificar islas de píxeles (o islas) y separar cada una de ellas para su separación, descarte, eliminación y coloración de rellenado.

El DFS es un algoritmo más eficiente de rellenado que el algoritmo de relleno por difusión (Apéndice A), el cual llega a causar un desbordamiento de pila debido al tamaño de las imagenes, ya que también tiene la característica de marcar a todos los nodos que ya han sido previamente visitados.

IV. Desarrollo

El programa comienza abriendo una imagen, la cual pasa a ser mandada por medio de un constructor a transformarse un objeto QImage, este objeto es transformado a un mapa de pixeles (Pixmap) a una escénca gráfica de Qt, la cual se muestra en la interfaz gráfica.

Binarizando la imagen

(FIG. 1.) El programa después llama a una función que itera sobre cada pixel del QImage y binariza la imagen; reemplaza todos los píxeles que tengan un RGB diferente a blanco a un

RGB negro. Esto con el propósito de tratar el ruido ocasionado por la exportación a formato PNG como parte del círculo.

Cada cluster tiene un color RGB diferente

(FIG. 2.) Cada cluster tiene un color RGB diferente. El programa en seguida llama a una función que actúa como separador de figuras; recibe una referencia de el objeto QImage, un color RGB inicial y un punto de partida. Esta función es una variación de un algoritmo de búsqueda de profunidad y su objetivo está en colorear a todas las figuras aisladas que encuentre de un color RGB ligeramente diferente a la isla anterior, esto para poder marcar a cada figura con un código único, y que después será utilizado para buscar figuras específicas.

La búsqueda profunidad verifica si cada pixel vecino es un pixel válido, si no es así, el separador se llama recursivamente a sí misma con un color que se sabe que jamás podrá ser identificador de uno de los círculos (en este caso, azúl o (0, 0, 255) debido a la naturaleza en la cual los códigos de RGB van incrementando, esto permite descartar a la figura como una figura no válida e ignorarla (FIG. 13.), también ocasiona que el número de círculos disminuya (ya que no es una figura válida) y que el código RGB se reinicie al código anterior (de modo que no se brinque el identifcador que iba a ser asignado en caso de ser un círculo).

Al finalizar la búsqueda de profundidad, el algoritmo manda a buscar otra función que detecta la posición inicial (el pixel inicial) de la próxima figura que es negra, debido a que colorea las figuras de una manera secuencial, cuando esta función externa ya no encuentre más pixeles negros en la imagen, significa que ya todos los clústeres han sido procesados y se les ha asignado un código RGB úncio o han sido ignorados con color que representa el descarte. Esta función también devuelve el número total de círculos como un entero.

Se hace un ciclo de 1 hasta el número de círculos (regresado por el separador) y en una variable de tipo QRgb se genera un código RGB generado exactamente para el número de círculo que fue identificado por el separador. Esto se logra inicializando el primer color de RGB como qRgb(2, 2, 2) (correspondiendo al primer círculo o también qRgb(2*i, 2*i, 2*i) tal que i = 1) y se llama a a 4 funciones que buscan el punto medio del círculo en sus puntos arriba, derecha, abajo e izquierda; cada una de estas funciones es una variación de el método de encontrar un punto medio; (a + b)/2 en donde a es el primer píxel de color qRgb(2*i, 2*i, 2*i) y b es el último pixel de color qRgb(2*i, 2*i, 2*i), visto desde 4 ángulos de referencia.

Ajustando simetrías

(FIG. 3.) Debido a que los círculos son figuras imperfectas, es posible que los 4 puntos no coincidan exactamente con sí mismos y es esperable que se necesite hacer un ajuste debido

a la naturaleza de los círculos. Para arreglar esto, si (a + b)/2 es impar, entonces los números son truncados, y el eje de las y del punto superior pasa a ser el punto máximo de el eje de las y en su punto superior y el punto de las y en su punto inferior; esto con el objetivo de siempre favorecer al punto medio de un rango impar (en caso de que exista), y alinear los rangos impares con los pares; el eje de las x es igual al eje de las x de el punto superior, el eje de las x del punto de la izquierda pasa a ser el máximo de de la coordenada del eje de las y del punto de las izquierda vs el punto máximo del eje de las y del punto de la derecha, y el eje de las y del punto derecho es igual al eje de las y del punto izquierdo.

Calculando el centroide y los radios

(FIG. 4.) Debido que ahora todos los puntos están alineados correctamente, es posible calcular un centroide en una posición exacta asignando el eje de las x del centroide igual al eje de las x del punto superior, y el eje de las y del centroide al eje de las y del punto izquierdo. Los radios se pueden sacar también matemáticamente calculando la diferencia entre el centroide y uno de los respectivos puntos, la suma de estos radios nos da como resultado también la longitud del eje de las y del círculo y la longitud del eje de las x del mismo.

Ignorando donas

(FIG. 5.) Ya que las donas no son consideradas círculos, este proceso de obtención de puntos extremos ignora completamente si existe o no existe un punto sin rellenar en el centro, por lo cual detectar una dona consiste simplemente de pregunta si el centroide es blanco. Si esto es así, se manda a llamar a una función de búsqueda de profunidad que colorea la figura como un color inaccesible por el algoritmo de generación de códigos RGB, al igual que las figuras incompletas anteiormente mencionadas. También se decrementa el contador para la sincronización de identificadores correcta.

Eliminación de óvalos

(FIG. 6.) La determinación si uno de los clústeres de pixeles sobrevivientes es un círculo o una dona (las únicas dos opciones restantes) se puede calcular obteniendo la distancia entre la longitud del eje de las y actual y el eje de las x del mismo, si esta diferencia sobrepasa los 10 pixeles de diferencia, entonces es una dona, y por lo tanto, se manda al mismo algoritmo pero esta vez en lugar de un color inválido, se colorea del RGB del fondo (en este caso, blanco) para representar una eliminación completa de la figura.

Todo lo que resta es un círculo

(FIG. 7.) En este punto, todas las figuras restantes son círculos, por lo tanto, se manda a llamar una función que recibe como parámetro todos los puntos de los extremos del círculo y se dibuja una caja verde al rededor de los círculos como un “reconocimiento” de se ha detectado un círculo. Aquí finaliza el programa; lo siguiente es etiquetar a todos los círculos con un identificador de un número; correspondiendo a el orden en el que fueron encontrados, y una serie de coordenadas que pasan a la interfaz gráfica en una tabla; señalando la posición de estos círculos en el espacio; esta tabla contiene las coordenadas de los 4 puntos extremos, el centroide y el radio.

V. Pruebas y resultados

FIG. 1. **Binarizando la imagen**

FIG. 2. **Cada cluster tiene un color RGB diferente**

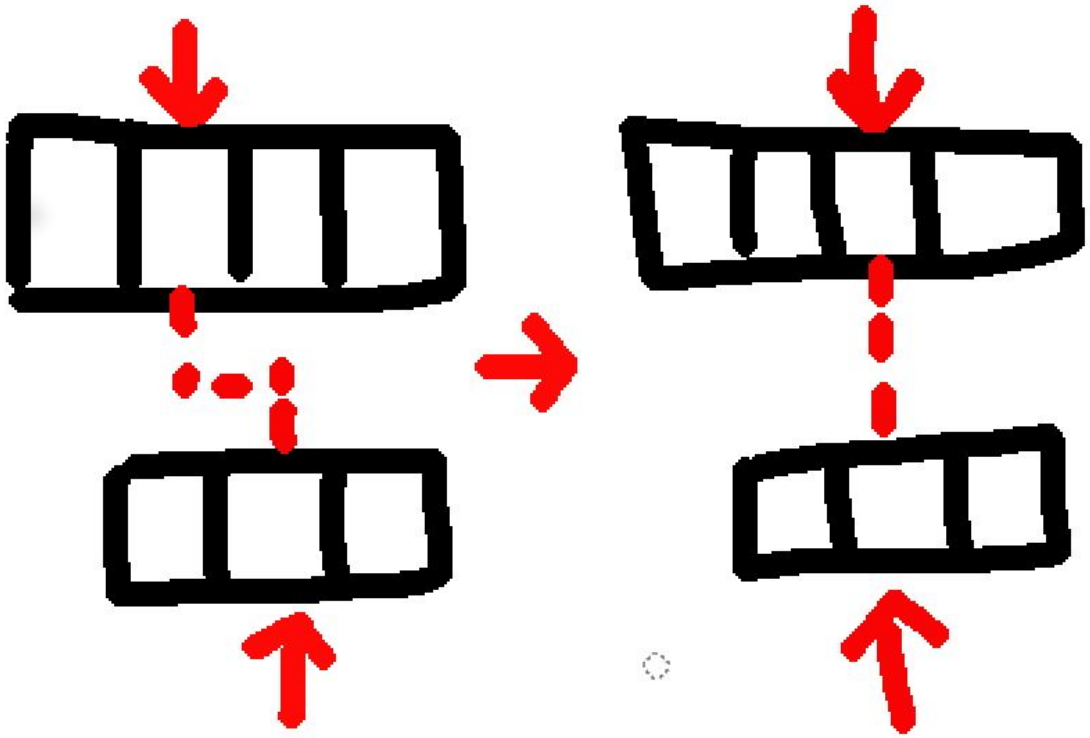


FIG. 3. **Ajustando simetrías**

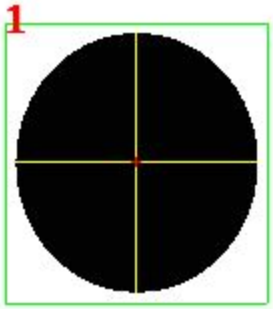


FIG. 4. **Calculando el centroide y los radios**

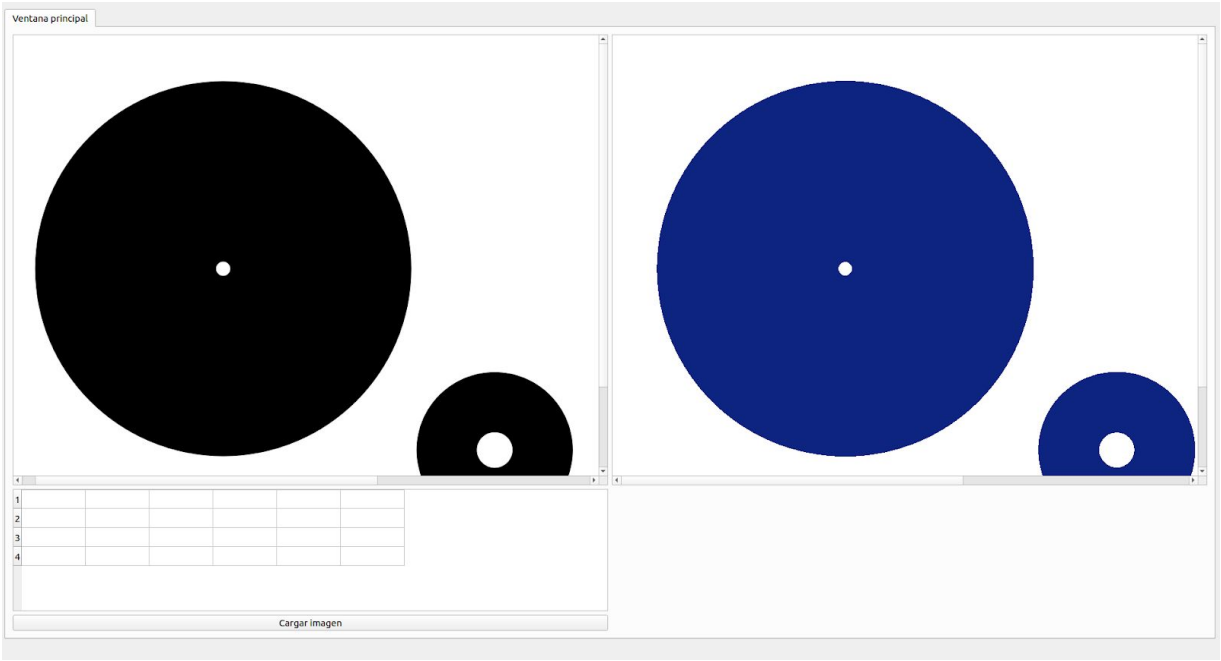


FIG. 5. **Ignorando donas**

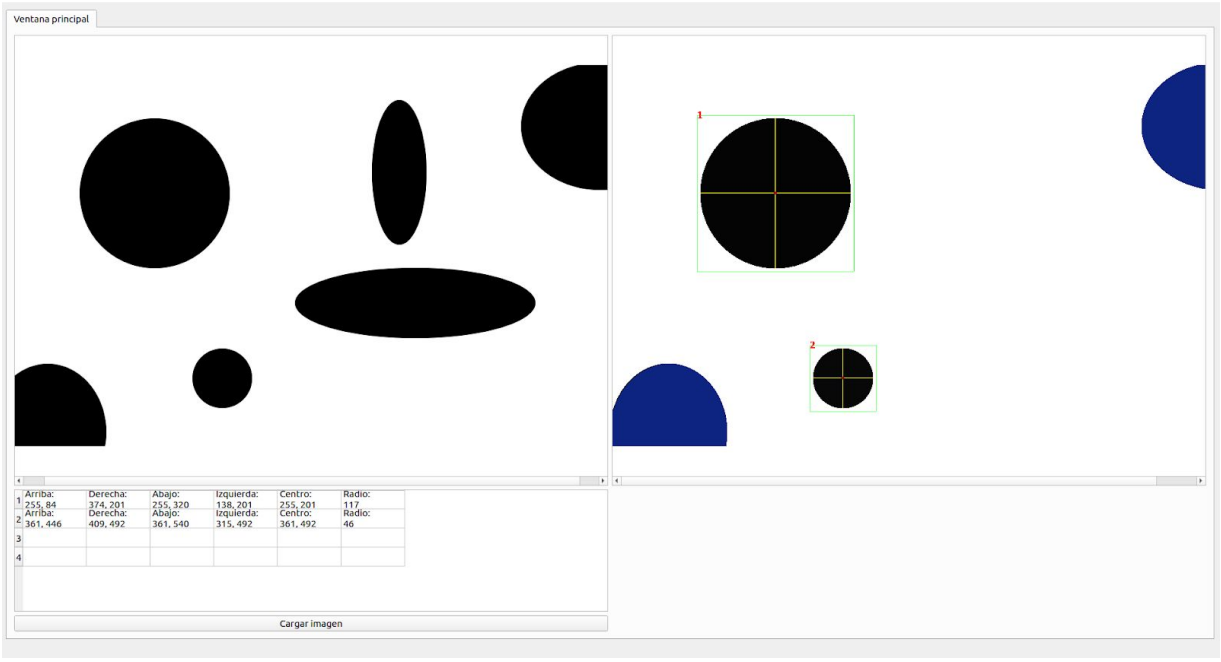


FIG. 6. **Eliminación de óvalos**

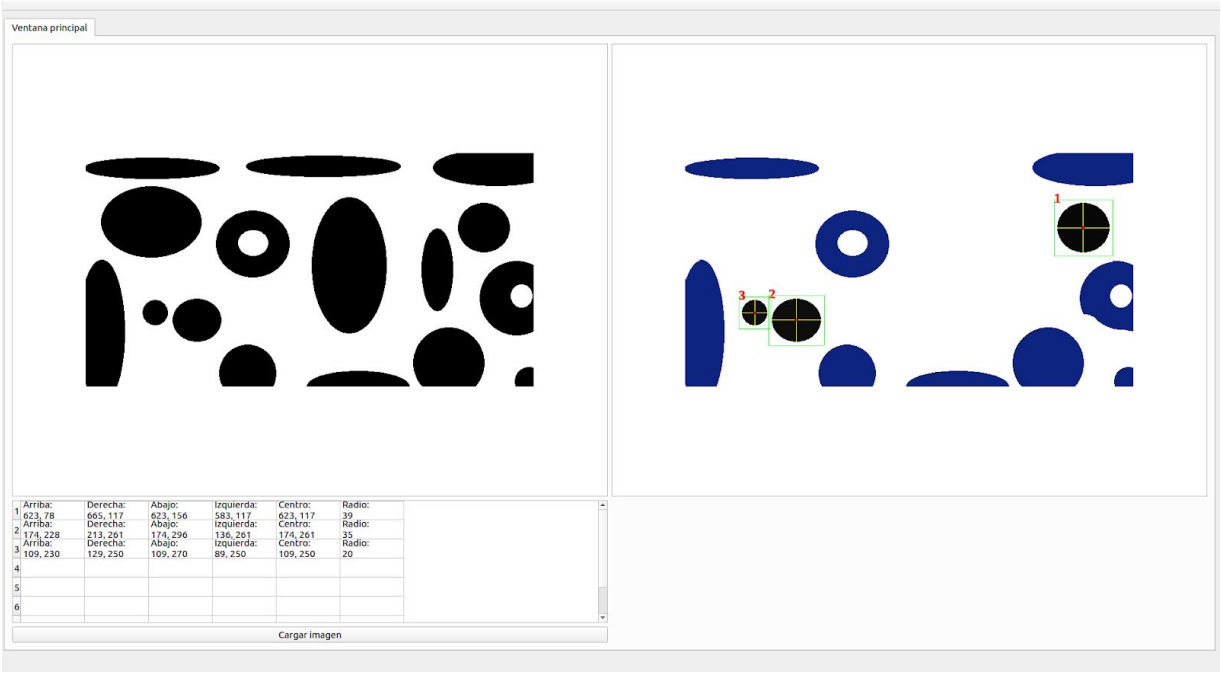


FIG. 7. **Todo lo que resta es un círculo**

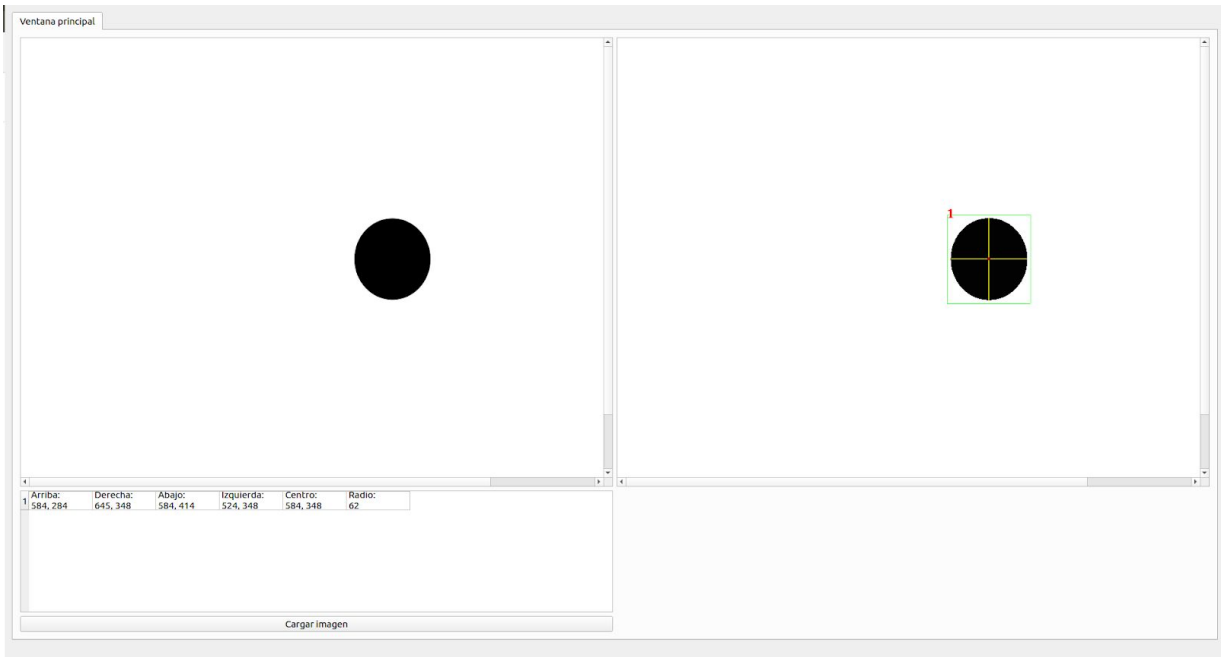


FIG. 8. **Clasificación de un solo círculo**

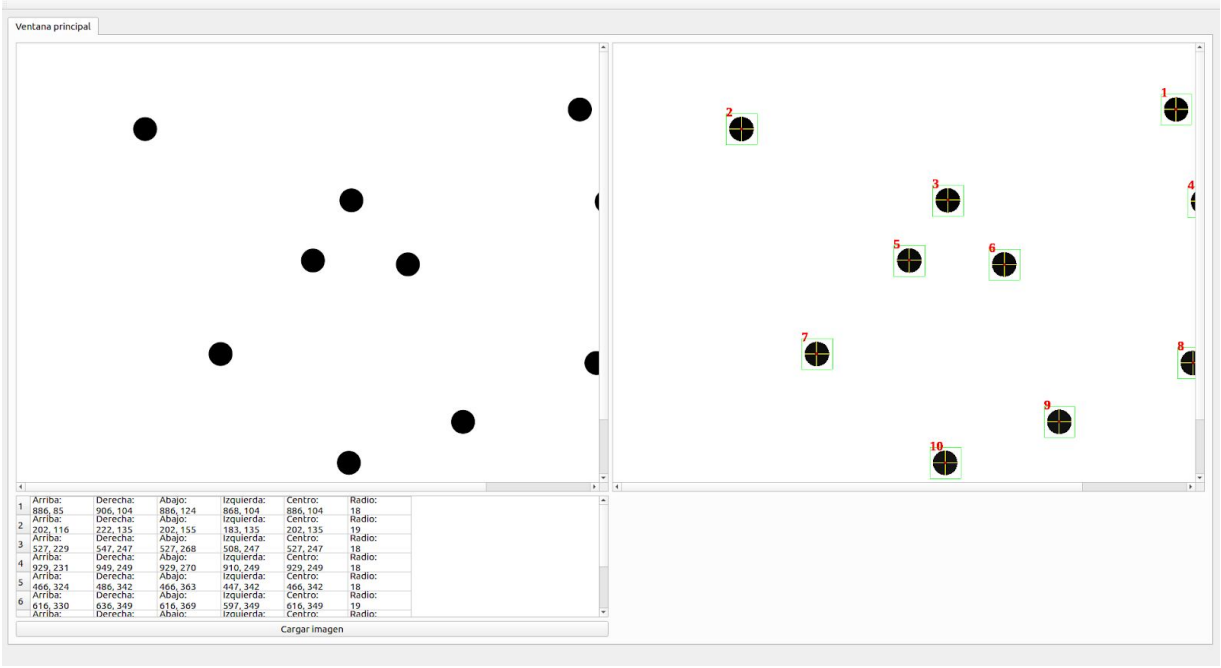


FIG. 9. **Clasificación de varios círculos**

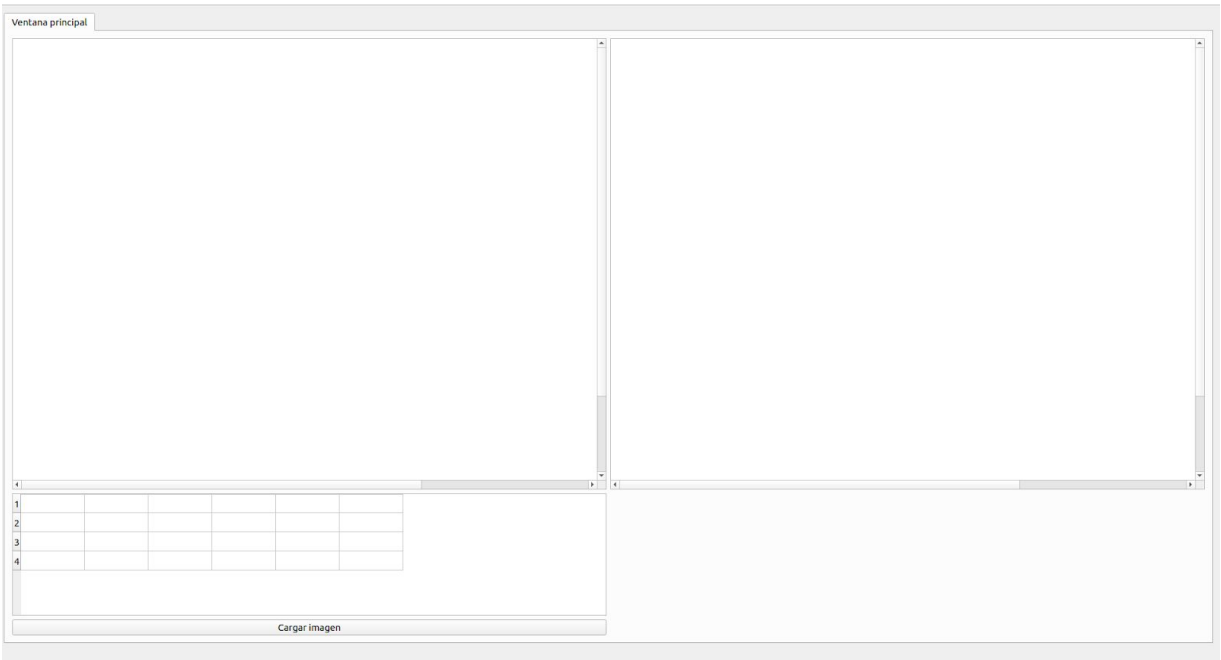


FIG. 10. **¿Qué ocurre si no hay nada?**

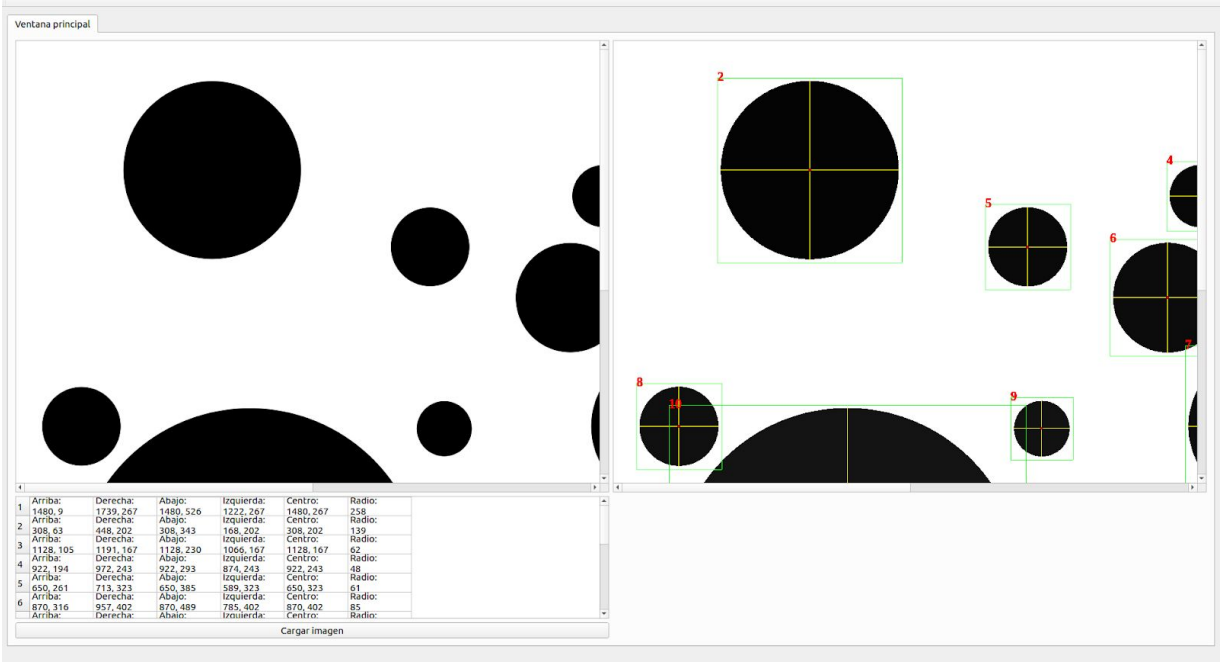


FIG. 11. **Clasificación de círculos de diversos tamaños**

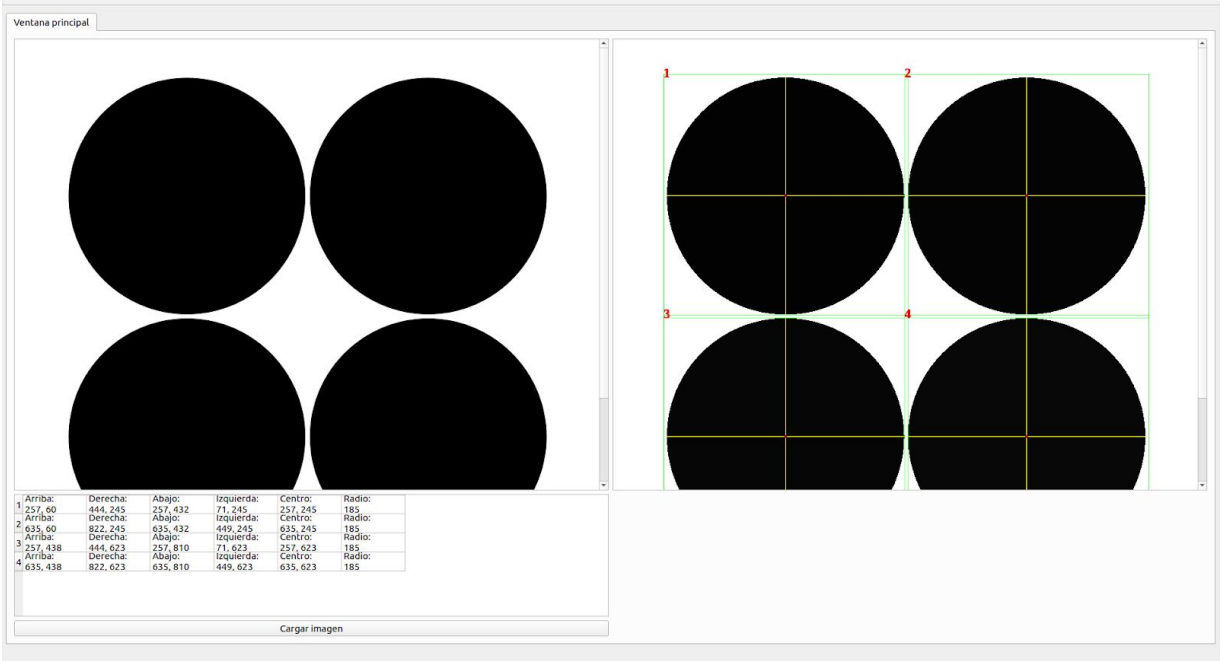


FIG. 12. **Clasificación de círculos muy cercanos a sí mismos**

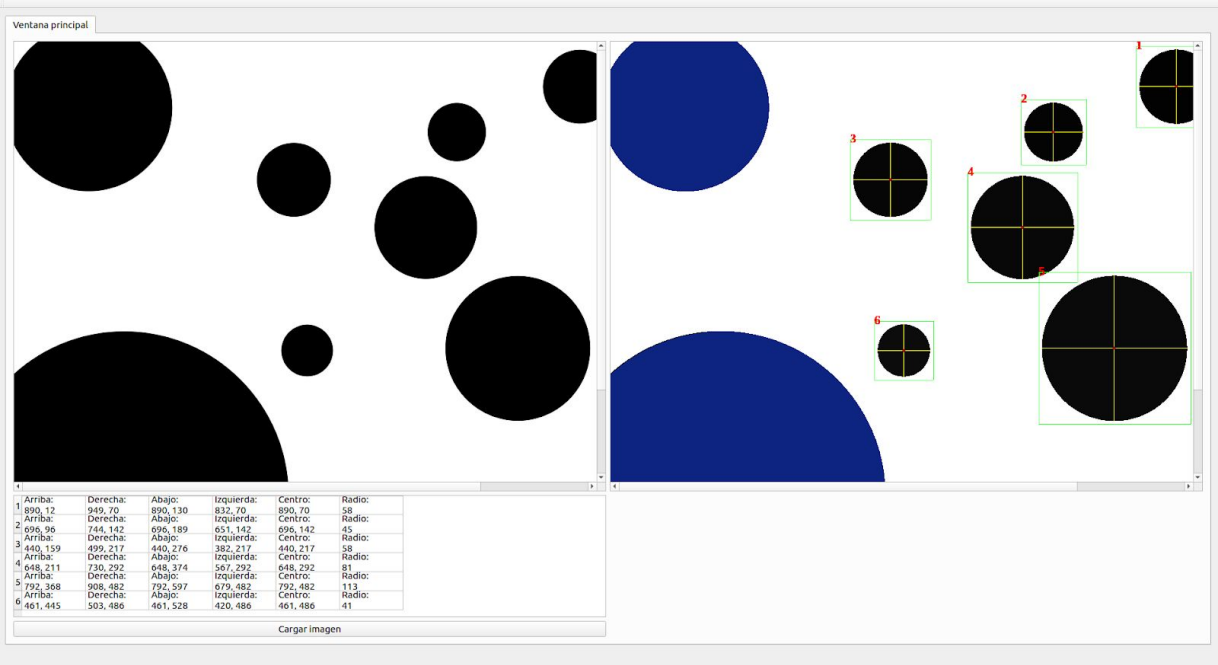


FIG. 13. **Figuras incompletas**

VI. Conclusiones

El procesamiento de imagenes es un área bastante grande en la cual podemos implementar algoritmos de grafos y/o árboles si se tiene la suficiente imaginación. Debido a que las dimensiones de una imagen suelen ser bastante grandes, es necesario utilizar algoritmos eficientes tales como búsqueda de profunidad para poder procesar la imagen en un tiempo aceptable.

Una imagen no es más que un conjunto de pixeles en una matriz, y un pixel no es más que información en un cierto tipo de formato, en este caso RGB, y por lo tanto, herramientas de las ciencias de datos son aplicables y toda la matemática que conlleva el estudio de este campo.

La detección de círculos podría resultar ser más compleja de lo que se trató en esta actividad, ya que en estas imagenes solo habían círculos y las imagenes estaban binarizadas a solamente blanco y negro. En un caso más realístico, puedo ver cómo la detección de círculos en un lugar con demasiado ruido puede fácilmente confundir a uno de estos algoritmos, y por lo tanto pienso que se necesitarían metodologías más sofisticadas para

generalizar el concepto de un círculo en un entorno con distracciones (una calle, por ejemplo) y que en una situación no controlada, se requerirían herramientas de inteligencia artificial tales como redes neuronales convolucionales.

VII. Apéndice A

Algoritmo de relleno por difusión

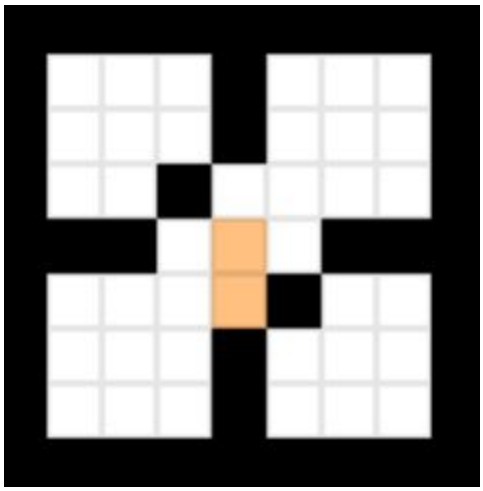


FIG. 14. El algoritmo de relleno por difusión, también llamado algoritmo de relleno, o -directamente del inglés- floodfill determina el área formada por elementos contiguos en una matriz multidimensional. Se usa en la herramienta Bote de pintura de programas de dibujo para determinar qué partes de un mapa de bits se van a rellenar de un color (o una textura), y en juegos como el Buscaminas, Puyo Puyo, Lumines y Magical Drop para determinar qué piezas pueden retirarse o seleccionarse.

El algoritmo de relleno en programas de dibujo, creado por S. Fazzini, requiere tres parámetros: un nodo inicial, un color para sustituir, y otro de relleno. El algoritmo rastrea todos los nodos que sean del color seleccionado, y a la vez contiguos entre sí y con el inicial, y los sustituye por el color de relleno.
Hay muchas maneras en las que el algoritmo de relleno por difusión puede ser estructurado, pero todas ellas hacen uso de tipos de datos tales como la cola o la pila, explícita o implícitamente. Una implementación del algoritmo de relleno por difusión basada en pilas se define de la siguiente manera (para un arreglo bidimensional):

Flood-fill (node, target-color, replacement-color):
1. Si el color de un *node* es distinto del que se pretende sustituir, se termina el algoritmo.
2. Asigna el color de *node* a *replacement-color*.

3. Se ejecuta de nuevo el algoritmo, usando el nodo situado a la izquierda del presente, y los mismos parámetros de color.
Se ejecuta de nuevo el algoritmo, usando el nodo situado a la derecha del presente, y los mismos parámetros de color.
Se ejecuta de nuevo el algoritmo, usando el nodo situado inmediatamente superior al presente, y los mismos parámetros de color.
Se ejecuta de nuevo el algoritmo, usando el nodo situado inmediatamente inferior al presente, y los mismos parámetros de color.
4. Fin del algoritmo.

Imagen e información de algoritmo de relleno por difusión:
https://es.wikipedia.org/wiki/Algoritmo_de_relleno_por_difusi%C3%B3n