Tarea de Investigación No. 2
DETECTOR DE BORDES EN IMÁGENES DIGITALES

Abiel Porras Garro - 2020209597[1], Elias Castro Montero - 2020098930 [2]
[1]Ingeniería en Computación, Instituto Tecnológico de Costa Rica, San José, Costa Rica
[2]Ingeniería en Computación, Instituto Tecnológico de Costa Rica, San José, Costa Rica

I. INTRODUCCIÓN

LA detección de bordes en imágenes digitales permiten obtener información relevante para identificar figuras y reconocer objetos. Para realizar este proceso, se pueden emplear diversas técnicas desde aquellas más simples como cálculo de derivadas para los píxeles de la imagen, hasta el uso de funciones de bibliotecas que implementan complejos algoritmos de varios pasos.

En nuestro caso, partiremos de un algoritmo detector de bordes, que implementa la derivación de la magnitud del valor de cada línea de píxeles tanto en el eje X como en el Y, para posteriormente superponiendo los valores obtenidos, con el fin de obtener el resultado final, una nueva imagen en la que se pueden apreciar los bordes de la imagen.

Utilizando la biblioteca OpenCV, se hará uso de su función `cv.Canny()` que emplea el famoso algoritmo de detección de bordes llamado **Canny Edge Detection** desarrollado por John F. Canny, este documento describirá el funcionamiento de este algoritmo, el cual consta de 5 pasos, que permiten obtener los bordes de una imagen digital de forma precisa.

Finalmente, se hará una comparación de los resultados obtenidos entre ambos algoritmos, tanto el nuestro que superpone la derivación en el eje X e Y, y la función `cv.Canny()` que proporciona OpenCV, con relación al tiempo promedio de ejecución del algoritmo, y nuestra percepción de la calidad de los bordes detectados de acuerdo a sus resultados finales.

II. DESARROLLO

A. Canny Edge Detection

La herramienta elegida para realizar la detección de bordes es la función `Canny` incluida dentro de la biblioteca OpenCV. Esta herramienta fue desarrollada por John F. Canny, de ahí su singular nombre [1].

El algoritmo que utiliza esta función cuenta con varias etapas; la primera de ellas es la de la eliminación de ruido, que es realizada mediante un filtro gaussiano de 5x5 píxeles [1].

Luego de esto se pasa a la siguiente etapa que consiste en encontrar el gradiente de intensidad de la imagen, mediante un filtro sobel en dirección X e Y para encontrar sus respectivas derivadas, ya que mediante estas podemos obtener su gradiente de arista y la dirección que sigue cada píxel [1].

En la siguiente etapa se realiza un escaneo de toda la imagen para detectar píxeles que no formen parte del borde a obtener, mediante la verificación de los píxeles continuos de cada píxel para verificar que si formen parte del borde total [1].

Finalmente, en la última etapa se decide cuáles líneas son realmente bordes y cuáles no, para esto se solicitan dos umbrales, uno máximo y otro mínimo, ambos números son parametrizables a la hora de utilizar la función [1]. Las aristas que tengan una gradiente de intensidad superior al umbral máximo son interpretadas como verdaderos bordes [1]. Las que están por debajo del umbral mínimo son descartadas [1]. Las aristas con un gradiente de intensidad que estén entre los dos valores de umbral son sometidas a una función que verifica si tienen conectividad con alguno de las aristas que ya fueron clasificadas como bordes, si se cumple la condición se suman a lista de bordes y por el lado contrario se desechan. Luego de este proceso se obtiene finalmente la nueva imagen con los bordes bien definidos [1].

B. Comparación de Resultados

Los resultados generados por el algoritmo que realizamos son bastante notorios y fáciles de interpretar, los bordes captados se muestran claramente en las imágenes generadas. Esto se logra gracias a que el algoritmo no realiza binarización de píxeles, por lo que es posible obtener distintos niveles de intensidad, lo que logra visibilizar mejor los bordes que están más marcados. No obstante, lo anterior también hace que haya un mayor nivel de ruido en el resultado por lo que puede complicarle la tarea a programas que realicen reconocimiento en estas imágenes, como reconocimiento de figuras a partir de los bordes identificados. La figura 1 muestra la imagen original de un paisaje, y la figura 2 muestra los bordes detectados en la imagen por nuestro algoritmo.

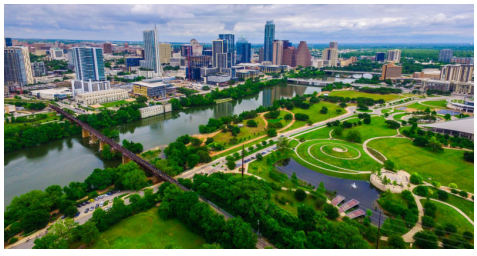

Fig. 1: Imagen original de un paisaje

Fig. 2: Bordes detectados por nuestro algoritmo en la imagen del paisaje

Por otro lado, los resultados generados por el algoritmo Canny Edge Detection tienen la característica de que están muy bien marcados en la imagen resultante. Esto debido a que este algoritmo si realiza binarización, por lo que no hay distintos grados de intensidad en la imagen y el proceso para obtener los bordes es más detallado. Sin embargo, no es perfecto, debido a que deja pasar muchos bordes que se ven similares a otros bordes más marcados, lo que hace que los más destacados se terminen perdiendo entre tantas líneas y en consecuencia se ensucia el resultado final, ya que se ve muy cargado visualmente. La figura 3 muestra los bordes detectados en la imagen utilizando Canny Edge Detection.

Fig. 3: Bordes detectados por la implementación de Canny Edge Detection de OpenCV

Finalmente, en términos de rendimiento, utilizando 16 imágenes de paisajes que se pueden encontrar en la carpeta *discord-imagenes*, es posible notar como el promedio de tiempo de ejecución utilizando Canny Edge Detection es de 0.0133 segundos, mientras que el de nuestro algoritmo que utiliza derivación es de 0.4474 segundos, por lo que es posible notar como la implementación de OpenCV del algoritmo Canny Edge Detection es en promedio más de 30 veces más rápido en concretar su resultado, que el algoritmo realizado por nosotros. Lo más probable es que esto suceda debido a que el algoritmo de OpenCV esta mucho más optimizado y evita muchos pasos que nosotros realizamos para llegar al resultado, además de que podría implementar técnicas de paralelización, que acelere su tiempo de ejecución. Finalmente, sus etapas podrían sean mucho más ligeras a nivel de cómputo que las nuestras.

III. Conclusión

- A pesar de que hay distintas maneras de obtener los bordes de una imagen, ninguna es perfecta, al menos por el momento. Por lo que a la hora de escoger un algoritmo debemos investigar a profundidad sus ventajas y desventajas para conocer la que mejor se adapte a los resultados que necesitamos.
- La calidad de la imagen es de gran importancia debido a que entre mejor definidos estén sus bordes va a ser mucho más sencillo separarlos y resaltarlos. Gracias a lo anterior se puede evitar suciedad de líneas de bordes principales y obtener un mejor acabado final. De lo contrario, se deberán utilizar adicionalmente técnicas para limpieza de ruido como el filtro gaussiano empleado en el algoritmo Canny Edge Detection.
- El uso de funciones de bibliotecas de procesamiento de imágenes como en este caso la función `Cany`, nos permiten a los desarrolladores experimentar con diversos algoritmos de forma sencilla, en este caso, se hizo uso de un algoritmo complejo de detección de bordes, dividido en 5 etapas, sin embargo, desde nuestro punto de vista, la detección de bordes se realizo únicamente llamando a una función. Como consecuencia, esto genera una nueva capa de abstracción, que podría hacer que desconozcamos la implementación que existe por detrás de la función, lo que a su vez nos da menos control sobre el trabajo que se realice, por ende, se recomienda hacer uso de la documentación, con el objetivo de entender, cuál es la implementación realizada por la biblioteca, para conocer a mayor profundidad sus ventajas, desventajas, y analizar casos de uso en donde podría ser más o menos óptima la solución.

References

[1] “OpenCV: Canny Edge Detection.” [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html