Tarea de Investigación 3

1st Ericka Céspedes Moya
Tecnológico de Costa Rica
San José, Costa Rica
ericka.cespedes@gmail.com

2nd Esteban Alonso González Matamoros
Tecnológico de Costa Rica
San José, Costa Rica
esteb.gonza29@gmail.com

***Resumen*—El presente documento comprende la primera y segunda parte de la tarea de investigación 3 para el curso Introducción al Reconocimiento de Patrones. En esta tarea se usan las bibliotecas de OpenCV y NumPy para el procesamiento de imágenes en Python. En la primera parte se usa el concepto de derivadas discretas recorriendo cada píxel de la imagen y escalamiento a la hora de superponer cada eje de la imagen para obtener los bordes de la imagen ingresada. Luego, en la segunda parte, se investiga acerca de distintos métodos para encontrar los bordes de una imagen y hacer la comparación entre la implementación de la primera y la segunda parte.**

***Index Terms*—procesamiento, imágenes, mapeo, bilineal, Python, OpenCV**

I. Investigación, resultados y comparaciones

Dentro de esta sección se estarán observando las diferencias o similitudes entre el programa para identificar bordes dentro de una imagen realizado por nuestro grupo y el programa que viene incluido dentro de la biblioteca de OpenCV.

I-A. Funcionamiento

I-A1. Sobel: Según Fengshou Tian, el operador Sobel es el más simple entre varios operadores de detección de bordes. Consta de dos grupos de 3x3 composición de la matriz (aquí representada por la matriz horizontal $Sobel - x$ y la matriz vertical $Sobel - y$). Los dos conjuntos de operadores y la imagen (representada por A aquí) se pueden convolucionar en un plano para obtener el valor aproximado de la diferencia de brillo vertical y horizontal. [1] Entonces $Gx = sobelx * A$, $Gy = sobely * A$, el resultado final es: $G_x = \sqrt[2]{G_x^2 + G_y^2}$ [2]

Los filtros $Sobel - x$ y $Sobel - y$ 3 × 3 se pueden obtener de la siguiente manera.

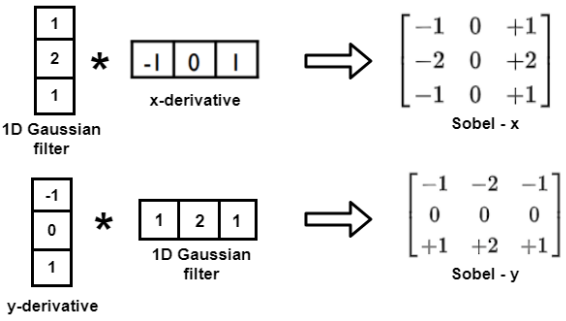

Figura 1. Operador de Sobel.

Atul Krishna Singh y Pankaj Kang (2019) establecen que mientras el filtro gaussiano se usa para difuminar, el operador de Sobel calcula el gradiente con suavizado. Por tanto, es menos sensible al ruido. Debido a la propiedad de separabilidad del kernel, el operador de Sobel es computacionalmente eficiente.

Este método es muy similar al primer algoritmo utilizado con derivadas discretas solo que se utiliza una máscara y se convolucionan. [3]

I-A2. Canny: La detección de bordes Canny es un algoritmo de detección de bordes popular desarrollado por John F. Canny en 1986. Este es un algoritmo de etapas múltiples: reducción de ruido, determinación del gradiente, supresión no máxima y umbral de histéresis. Dado que la detección de bordes es sensible al ruido en la imagen, el primer paso es eliminar el ruido de la imagen con un filtro gaussiano de 5x5. Luego, el núcleo de Sobel filtra la imagen suavizada en las direcciones horizontal y vertical para obtener la primera derivada en la dirección horizontal (G_x) y vertical (G_y). A partir de estas dos imágenes, podemos encontrar el gradiente de borde y la dirección de cada píxel. La dirección del gradiente siempre es perpendicular a los bordes.

Una vez se reduce el ruido y se determina el gradiente, el siguiente paso es la supresión no máxima. Después de obtener la magnitud y la dirección del gradiente, la imagen se escanea completamente para eliminar los píxeles no deseados que pueden no constituir un borde: se verifica si el píxel es un máximo local a lo largo de su dirección de gradiente dentro de su vecindad.

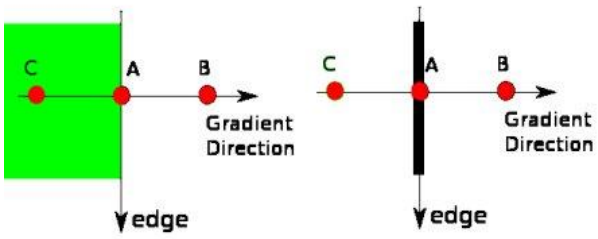

Figura 2. Supresión no máxima.

El punto A está en el borde (dirección vertical). La dirección del gradiente es el borde normal. Los puntos B y C están en una dirección de gradiente. Así que se verifica el punto A con los puntos B y C para ver si forma un máximo local. Si es

así, se sigue a la siguiente etapa, de lo contrario se suprimirá (será cero).

La última etapa es el umbral de histéresis donde se determinan los bordes. Se tienen dos umbrales, MinVal y MaxVal. Cualquier gradiente de intensidad mayor que MaxVal es definitivamente un borde y menor que MinVal definitivamente no es borde, por lo que se descarta.

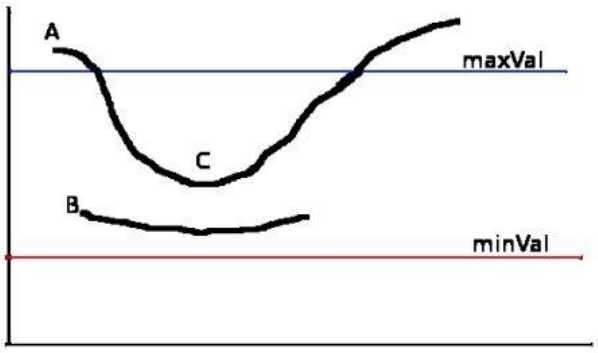

Figura 3. Umbral de histéresis.

El borde A está en MaxVal entonces se considera un ”borde definido”. Aunque el borde C está por debajo de MaxVal, está conectado al borde A y, por lo tanto, se considera un borde válido. Obteniendo una curva completa. Sin embargo, el borde B, aunque está por encima de MinVal y es la misma área que el borde C, no está conectado a ningún ”borde determinado” y se descarta. Por lo tanto, se tiene que elegir MinVal y MaxVal y obtener el resultado correcto. Esta etapa también elimina la suposición de ruido de píxeles pequeños entonces terminamos con bordes fuertes en la imagen.

El algoritmo de Canny usa 4 máscaras para encontrar los bordes en las direcciones horizontal, vertical y diagonal (se pueden comparar con las dos de Sobel). Sin embargo, el algoritmo de Canny tiene varias etapas como ya se explicaron y los bordes identificados son muy particulares y bastante diferentes a los identificados con los demás algoritmos. [4]

I-A3. Laplacian: Según Fengshou Tian, el uso de la primera derivada para la detección de bordes a menudo resulta en la pérdida de detalles de la imagen. Por lo tanto, se usa la segunda derivada para la detección de bordes, que es el operador de Laplace. La derivación de una función bidimensional es la siguiente:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Figura 4. Función bidimensional de Laplace.

La función anterior se puede aproximar de la siguiente manera:

$$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

Figura 5. Función bidimensional de Laplace.

La primera derivada a menudo produce bordes más anchos, mientras que la segunda derivada produce bordes más delgados para bordes suaves y puede producir bordes dobles. Además, la segunda derivada tiene una fuerte respuesta a los detalles, como líneas finas y puntos de ruido aislados.

Para la detección de bordes de Laplace, el mayor problema es la amplificación del ruido. Por lo tanto, las personas a menudo usan el filtrado Gaussiano antes de usar la detección de bordes de Laplace. [5]

El operador laplaciano es un operador diferencial de segundo orden en el espacio euclidiano n-dimensional, que se puede utilizar para mejorar la imagen o extraer los bordes. El efecto no es muy diferente al anterior. Pertenece al algoritmo de segundo orden como Canny. Sin embargo, debido al cálculo del gradiente, el operador laplaciano de OpeCV también utiliza Sobel. [1]

I-B. Imágenes resultantes

A continuación, mostramos la imagen original que se ingresó a ambos programas para comparar las imágenes resultantes, y por último, comparamos los resultados obtenidos con cada método. Cabe recalcar que esto es solo un ejemplo de los muchos realizados durante el análisis de introducir imágenes y revisar sus resultados correspondientes.

Figura 6. Imagen original ingresada a ambos programas.

Con el método utilizado en la primera parte de derivadas discretas logramos un mejor resultado o uno más detallado. La aplicación de cada método dependerá del resultado deseado.

Figura 7. Imagen resultante con el método 1 aplicado solo en el eje x.

Figura 8. Imagen resultante con el método 1 aplicado solo en el eje y.

Figura 9. Imagen resultante con el método 1 aplicado superponiendo ambos ejes.

También se realizó para el laboratorio 3 un método utilizando Sobel, donde como resultado obtuvimos el siguiente.

Figura 10. Imagen resultante con el método Sobel programado por nuestro grupo.

Las siguientes imágenes fueron las obtenidas a partir de las funciones incluidas en la biblioteca de OpenCV.

Figura 11. Imagen resultante con el método de Sobel aplicado solo en el eje x.

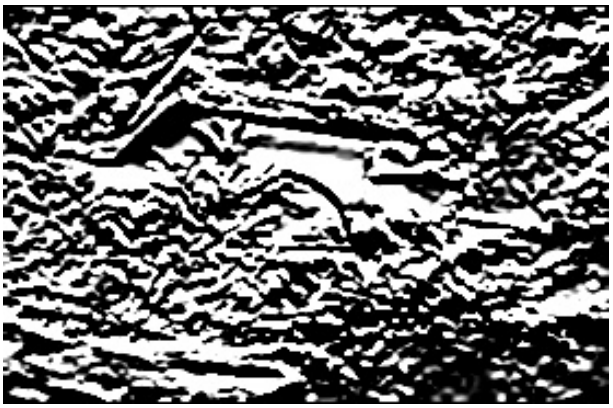

Figura 12. Imagen resultante con el método de Sobel aplicado solo en el eje y.

Figura 13. Imagen resultante con el método de Sobel aplicado en ambos ejes.

Figura 14. Imagen resultante con el método Laplacian.

Figura 15. Imagen resultante con el método de Canny.

I-C. Comparaciones

En cuanto a comparaciones, luego de varias ejecuciones con distintas imágenes podemos observar que las imágenes resultante con el método 1 en ambos ejes, el método de Canny y el método Laplacian brindan los mejores y más exactos resultados en cuanto a la detección de bordes de figuras dentro de una imagen, los bordes en estos tres métodos se ven de una manera más ”limpia” y ordenada, permitiendo que sean más fáciles de entender los bordes identificados, sin embargo, opinamos que la mejor solución para este problema fue la implementación realizada por nuestro grupo, la cual se puede apreciar en la figura 9, esto debido a que las lineas para definir bordes se notan más finas, y las formas se resaltan de mayor proporción comparado a los demás métodos y resultados. Se realiza esta comparación general de primero debido a que opinamos que realizar distintas comparaciones de uno a uno entre todos los resultados tendría muy poca relevancia, pocas diferencias notables y resultaría bastante repetitivo, esto debido a que muchos resultados son muy semejantes a otros o no tienen algo significativo que comparar con los demás. Por lo tanto, a partir de este párrafo solo se explicarán las comparaciones más interesantes o relevantes dentro de las figuras anteriores.

Luego, encontramos muy interesante que utilizar el método 1 programado por nuestro grupo obtuviera mejores resultados que aquellos métodos incluidos en la biblioteca de OpenCV. Como resultados de que esta afirmación es real y se puede confirmar, se pueden observar las figuras 9 (Programado por nuestro grupo y con resultado en ambos ejes), 13 (Método Sobel de OpenCV con ambos ejes), 14 y 15 (Método Canny obtenido de OpenCV), y compararlas con los que deberían de ser los bordes en la original (Figura 6). Sin duda, se puede observar que no solo los bordes son más finos y mejor demarcados, sino, que la forma que terminarían realizando se presenta de mejor manera y es más fácil de identificar para el ojo humano.

También, habiendo realizado un método que utilice Sobel (Figura 10) y utilizar el método de Sobel de la biblioteca Opencv (Figura 13) podemos realizar una comparación entre estos dos, al observar ambos resultados, notamos que el método programado obtiene resultados más entendibles y más definidos que el de OpenCV, pero con el costo de incluir más bordes de los necesarios en el resultado, también se puede observar que el método de OpenCV intenta hacer las formas de una manera más simple o minimalista, intentando que por ejemplo, una planta se vea como un conjunto de rayas bien definidas en lugar de la forma compleja que dicha planta tenga.

Otro aspecto importante a comparar es las diferencias entre usar un método en el eje x o en el eje y, debido a que aunque en los párrafos anteriores ya se comentó el hecho de que ambos ejes son mejores que cualquiera de los dos separados, para identificar bordes, también es necesario identificar las variantes entre un eje y el otro. Primeramente, los objetos que estén más enfocado a crecer en cierto eje o que sean más extensos hacia cierto eje van a ser identificados de mayor manera al aplicar el método en ese eje, por ejemplo, las plantas y los arboles suelen crecer o verse en el eje y o, dicho de otra forma, ser verticales, por lo que al ejecutar el reconocimiento de bordes en dicho eje, estas formas van a ser reconocidas de mayor manera, al igual que con pilares o edificios, sin embargo, en cuanto a techos o formas más horizontales como aceras o pisos, estos se van a reconocer mejor en el método con el eje x, sin embargo, es importante explicar que las diferencias no suelen ser muy extensas, solo que muchos detalles de los bordes de un objeto

enfocado a cierto eje, se verán mejor al ejecutar el método en dicho eje, pero principalmente serían detalles, a menos que el objeto sea muy delgado o pequeño, no debería ser eliminado de la figura resultante al utilizar un eje o el otro.

II. Conclusiones

La primera diferencia que se puede observar es que el primer algoritmo utiliza diferencias de píxeles simples para calcular el cambio de intensidad en el gradiente de la imagen. Por otro lado, los demás emplean las diferencias entre el píxel central y los elementos vecinos. Además, usualmente se utiliza una máscara para su implementación. Sin embargo, son muy similares los algoritmos por cómo recorren la imagen y calculan la derivada.

En segundo lugar, es importante recalcar la importancia de utilizar ambos ejes al realizar la detección de bordes, esto sin importar el método utilizado, debido a que la unión de ambos permite una imagen con mejores resultados y evita que se pierdan bordes que solo se identificarían en uno de los dos ejes, esto se puede observar en las figuras 9 y 13.

En el primer caso se puede decir que el algoritmo es muy sensible al ruido, no proporciona información sobre la orientación de los bordes y funciona mejor con imágenes binarias. En el caso de Sobel, se detectan bordes donde la magnitud del gradiente es alta. Esto hace que el detector de bordes Sobel sea más sensible al borde diagonal que a los bordes horizontales y verticales. Con respecto al algoritmo Canny: Este algoritmo utiliza un umbral de histéresis y supresión no máxima para detectar bordes débiles y más fuertes y finalmente producir un borde de 1 píxel de grosor.

Los métodos derivados de segundo orden como el Laplaciano son métodos más sofisticados para la detección de bordes automatizada, sin embargo, siguen siendo muy sensibles al ruido. Como la diferenciación amplifica el ruido, se sugiere suavizar antes de aplicar los laplacianos. El Laplaciano es una derivada de segundo orden y la detección de bordes de ”orden superior” no necesariamente da resultados más nítidos (asumiendo que la ”detección de orden superior” significa utilizar una derivada de orden superior).

En cuanto a métodos que varían según el eje seleccionado, concluimos que el usar un eje o el otro no suelen generar cambios significativos, sino que simplemente eliminan detalles pequeños de los bordes, sin embargo, igual es preferible analizar la imagen e identificar cual eje posee más objetos relacionados para evitar perder más información de la necesaria al utilizar uno de estos métodos. En caso de ser posible, se recomienda utilizar ambos ejes en lugar de uno solo, pero en caso contrario, utilizar según lo mencionado anteriormente en este párrafo.

Se puede concluir que con cada método se obtienen resultados diferentes y dependiendo de lo que se requiera, se puede usar el método de preferencia.

Referencias

[1] F. Tian, “Detección de bordes de imagen opencv (4),” *Programador Clic*. [Online]. Available: https://programmerclick.com/article/14231755892/

[2] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*. United States: McGraw-Hill, Inc., 1995, ch. 5: Edge Detection.

[3] A. K. Singh and P. Kang, “First-order derivative kernels for edge detection,” *TheAILearner*, 2019. [Online]. Available: https://theailearner.com/2019/05/24/first-order-derivative-kernels-for-edge-detection/

[4] F. Tian, “Detección de bordes opencv canny,” *Programador Clic*. [Online]. Available: https://programmerclick.com/article/650572950/

[5] ——, “Introducción a la detección de borde de opencv (2): operador de laplace,” *Programador Clic*. [Online]. Available: https://programmerclick.com/article/7067315508/