INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

“Detección de bordes Canny”

Alumno:

Ayala Arroyo Raúl Eduardo

Unidad de aprendizaje:

Visión Artificial

Profesor:

Sánchez García Octavio

Fecha: 04/11/22

Grupo: 5BM1

Tabla de contenido

Detección de bordes con Canny

Es un reconocido algoritmo de visión por computador para detección de bordes, lleva su nombre por su desarrollador John F. Canny.

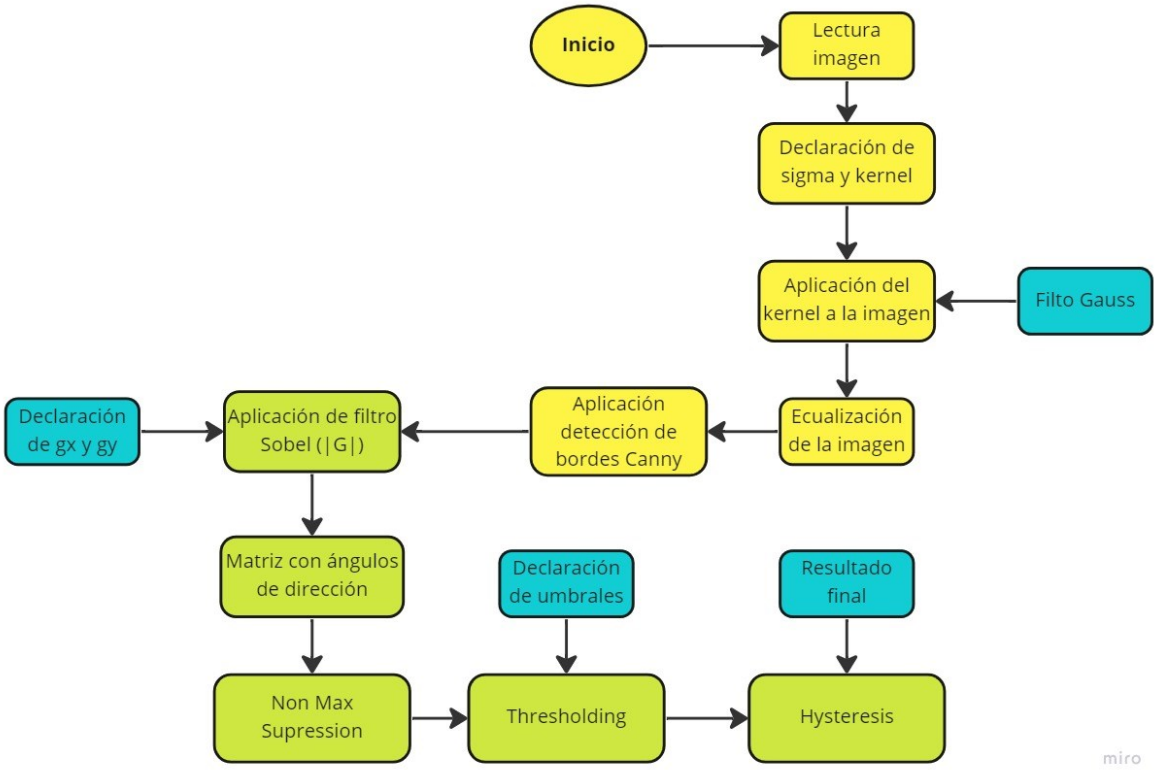
Este Algoritmo se puede usar en openCV mediante la función cv.Canny, internamente esta función realiza las siguientes etapas:

- Noise Reduction Filtra el ruido en la imagen mediante un filtro gaussiano
- Finding Intensity Gradient of the Image Encuentra el gradiente. El gradiente define dos valores: la dirección en donde el cambio de intensidad es máximo y la magnitud de esa dirección.

- Non-maximum Suppression Después de obtener la magnitud y la dirección del degradado, se realiza un escaneo completo de la imagen para eliminar los píxeles no deseados que pueden no constituir el borde. Esto elimina los píxeles que no se consideran parte de un borde. Por lo tanto, solo quedarán líneas finas (bordes candidatos).
- Hysteresis Thresholding Esta etapa decide cuáles son los bordes que son realmente bordes y cuáles no. Para esto, necesitamos dos valores de umbral, minVal y maxVal.

Los bordes con un gradiente de intensidad superior a maxVal seguramente serán bordes y aquellos por debajo de minVal seguramente no serán bordes, por lo que se descartan. Aquellos que se encuentran entre estos dos umbrales se clasifican como bordes o no bordes en función de su conectividad. Si están conectados a píxeles de “borde seguro”, se consideran parte de los bordes. De lo contrario, también se descartan.

Diagrama y descripción

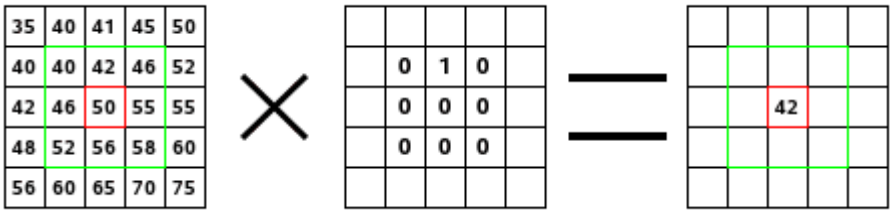


- **Lectura imagen**
  Por medio “imread” leemos la información de una determinada imagen para poder procesarla.

- **Declaración de sigma y kernel**
  Se le pide al usuario que determine el tamaño del kernel además del valor de sigma, para que puedan ser declarados de manera dinámica y a su vez aplicarlos posteriormente.
  El valor de cada posición en el kernel será determinado por la fórmula de Gauss:

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- **Filtro Gauss**
  Recorremos la imagen en grises aplicando la suma de los productos de los valores del kernel por los valores de los pixeles vecinos del píxel que se está procesando.



- **Ecualización de la imagen**
  Por medio de la función de openCv ecualizamos la imagen, enviando como parámetros la imagen que queremos ecualizar, y la matriz de imagen donde se guardará el resultado.

```
//ecualizado
cv::equalizeHist(imgFiltrada, imgEcualizada);
```

- **Aplicación de bordes Canny**
  - **Aplicación de filtro sobel (Gradientes)**
    Parecido al proceso del kernel, recorremos cada píxel en la imagen ecualizada y aplicamos la convolución para cada kernel gx y gy, y los valores los guardamos en una matriz Gx y Gy.

```
[-1.,  0.,  1.] [ 1.,  2.,  1.]
[-2.,  0.,  2.] [ 0.,  0.,  0.]
[-1.,  0.,  1.] [-1., -2., -1.]
```

h_x h_y

Una vez obtenido Gx y Gy obtenemos la magnitud de G (|G|) por medio de :

$$G = \sqrt{(F_x^2 + F_y^2)}$$

Y finalmente generamos la matriz de magnitudes.

  - **Matriz con ángulos de direcciones**
    Una vez obtenido los valores de fx y fy podemos obtener la matriz de direcciones de cada píxel, la cual nos informará en qué dirección el píxel a evaluar se parecerá más. Para esto ocupamos:

$$\theta = tan^{-1}\left(\frac{F_y}{F_x}\right)$$

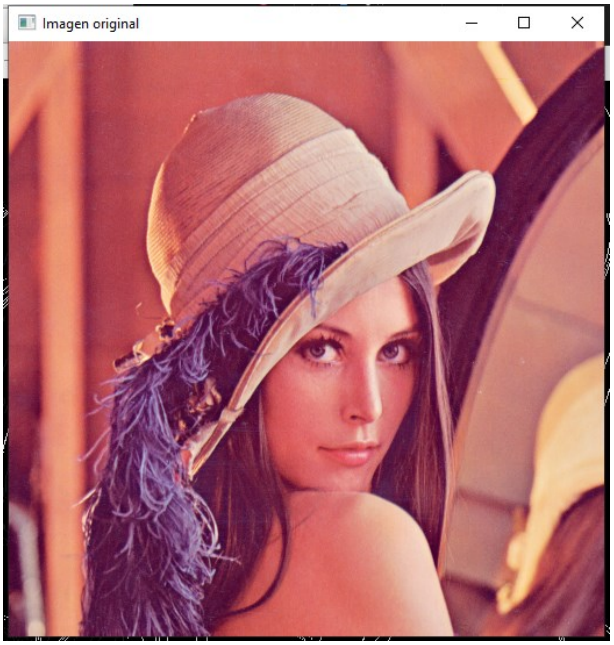
  - **Non Max Supression**
    Una vez obtenida la dirección, evaluamos los 2 pixeles adyacentes al píxel que se está evaluando, donde si el píxel a evaluar es mayor que ambos, entonces permanecerá, de lo contrario se irá a cero.

- **Thresholding**
  En este apartado se definen 2 tipos de umbrales, uno que será alto y otro será bajo. Se recorrerán los valores de cada píxel y si el valor del pixel es igual o mayor al umbral alto, será considerado como fuerte y se le pasará un valor de 255, si se encuentra dentro de los dos umbrales será considerado como débil con un valor de 25, y en el caso de que el valor quede por debajo del umbral bajo, será considerado como irrelevante y se quedará en cero.

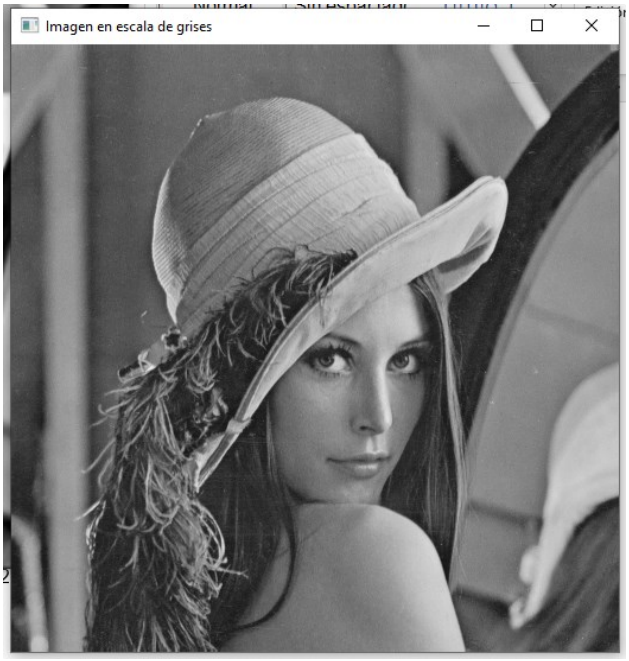
- **Hysterisis**
  Como paso final, se recorrerá la imagen en busca de los pixeles débiles, donde si cada pixel débil tiene un vecino fuerte, este mismo se hará fuerte, de lo contrario se hará irrelevante pasando su valor a cero.

Resultados

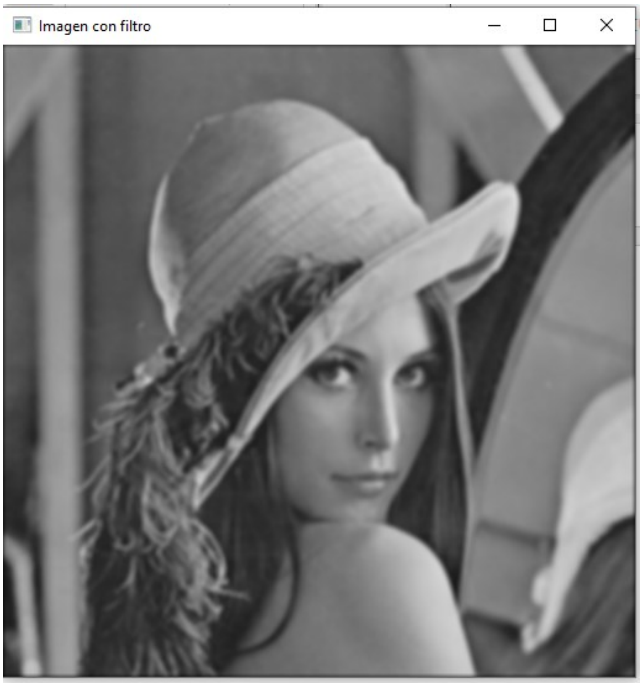
1. Imagen original



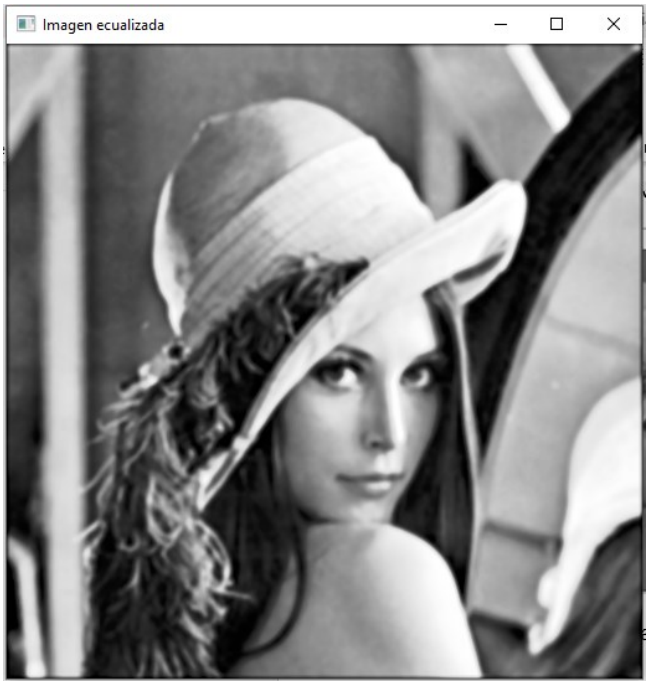
2. Imagen escala de grises



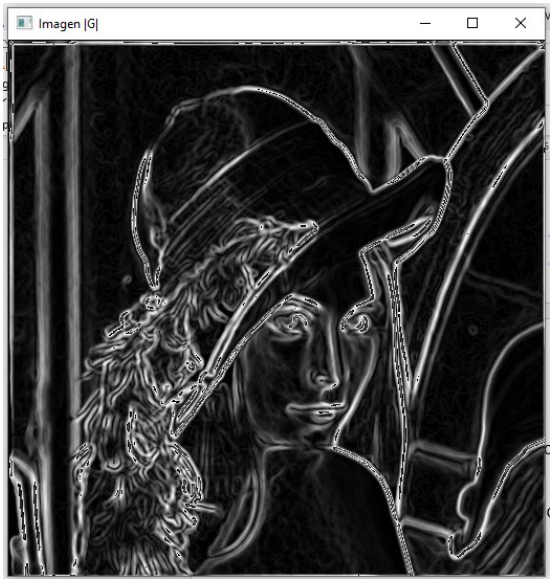
3. Imagen suavizada (kernel = 7, sigma = 2)



4. Imagen ecualizada

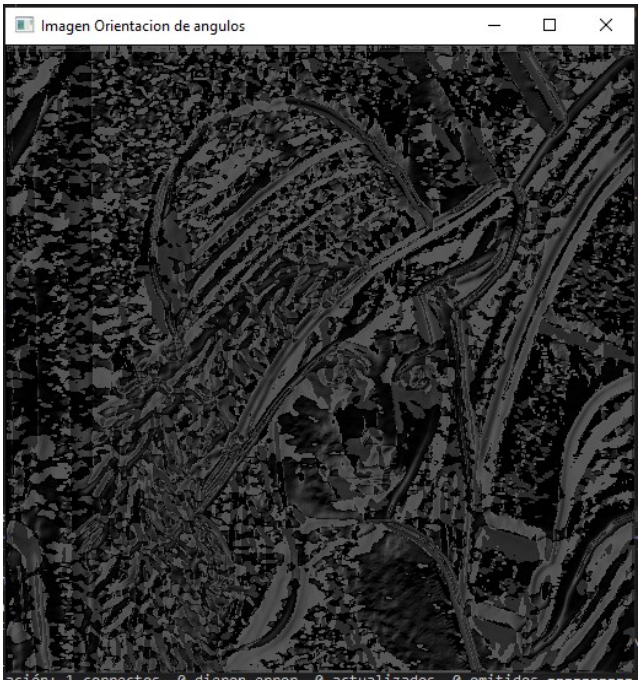


5. Imagen resultada de aplicar |G|

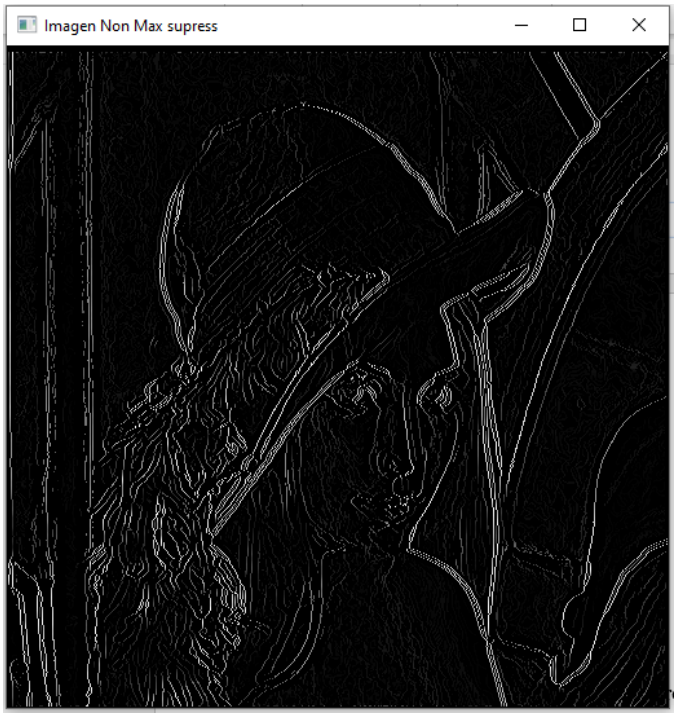


6. Imagen detección de borde Canny

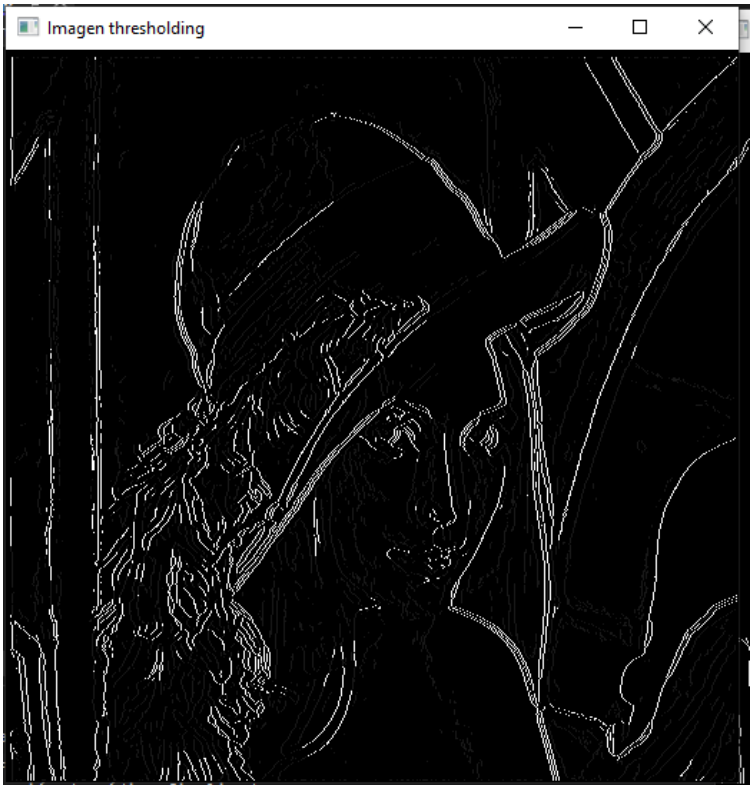
6.1 Imagen orientación de ángulos



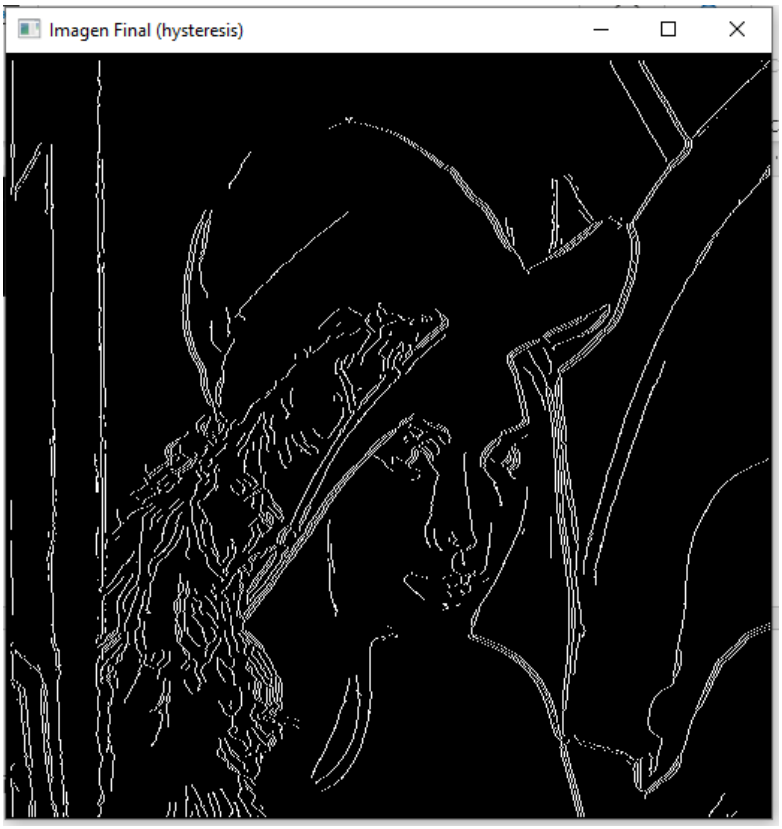
6.2 Imagen Non Max Supression



6.3 Imagen thresholding



6.4 Imagen hysteresis (final)



Referencias

Mimi. (2021, agosto 4). Detección de bordes Canny - openCV python. *KipunaEc*. https://noemioocc.github.io/posts/Detecci%C3%B3n-de-bordes-Canny-openCV-python/