Fourth report - Canny-Deriche filter implementation

Kimberly Grace Sevillano C.

***Subject*— Implementing the Canny-Deriche filter for edge detection in an image**

I. Objective

The objective of this report is to present the results of the implementation of the Canny-Deriche filter for edge detection in an image. The report describes the problem, methods used, methodology, results obtained, and conclusions drawn from the implementation

A. Problem Description

Edge detection is an important task in image processing and computer vision. It involves identifying boundaries between objects or regions in an image. Edge detection is used in various applications such as object recognition, image segmentation, and feature extraction.

The traditional approach to edge detection involves convolving the image with a low-pass filter to remove noise, and then applying a high-pass filter to enhance the edges. However, this approach can result in blurred edges and false detections in images with noise and variations in intensity.

The Canny-Deriche filter is a variant of the Canny filter that uses a Gaussian smoothing filter and a Gaussian derivative filter to improve edge detection. The filter has adjustable parameters that can be used to control the smoothness and accuracy of the detected edges.

II. Methods

There are several methods for edge detection in images. These include:

A. Sobel operator

A gradient-based method that uses a kernel to calculate the intensity gradient in the horizontal and vertical directions.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Fig. 1. Sobel operator

B. Prewitt operator

Similar to the Sobel operator but uses a different kernel to calculate the intensity gradient.

C. Laplacian of Gaussian (LoG)

A method that convolves the image with a Gaussian kernel and then applies the Laplacian operator to enhance edges.

D. Canny filter

A multi-stage method that uses a Gaussian filter, a gradient calculation, non-maximum suppression, and hysteresis thresholding to detect edges.

The Canny-Deriche filter is a variant of the Canny filter that uses a Gaussian smoothing filter and a Gaussian derivative filter to improve edge detection.

III. Methodology

The methodology for implementing the Canny-Deriche filter for edge detection is as follows:

- Define the Canny-Deriche smoothing filter in terms of the bandwidth parameter and the smoothing parameter alpha.

```
# Define the Canny-Deriche smoothing filter
def canny_deriche_filter(s, alpha):
    n = 1
    si = np.linspace(-s, s, 2*s+1)
    h = np.exp(-alpha*np.abs(si))
        *((alpha*np.abs(si))**n)
        * ((1-np.exp(-alpha*np.abs(si)))/((si
    **2)
                                        + 1e
    -10))
    h /= np.sum(h)
    return h, si
```

Listing 1. Function for Canny-Deriche smoothing filter in python

- Load the image and convert it to grayscale.
- Apply the Canny-Deriche filter to the smoothed image in both horizontal and vertical directions to obtain the magnitude and direction of the gradient.
- Apply non-maximum suppression to the gradient magnitude to eliminate pixels that are not part of edges.
- Apply a threshold to the gradient magnitude to obtain a binary image of edges.
- Refine the edge segmentation using a hysteresis thresholding algorithm to eliminate false edges and retain true edges.

IV. Results

The Canny-Deriche filter was successfully implemented for edge detection in the provided image. The filter parameters s and alpha were defined, and the filter was applied to the smoothed image. The magnitude and direction of the gradient were obtained, and non-maximum suppression was applied to the gradient magnitude to obtain an image of edges. A threshold was applied to the gradient magnitude to obtain a binary image of edges, and the edge segmentation was refined using a hysteresis thresholding algorithm.

The results of the edge detection were good for the provided image. The Canny-Deriche filter was effective in detecting edges in images with noise and variations in intensity. The smoothing parameter alpha was found to have a significant impact on the smoothness and accuracy of the detected edges.

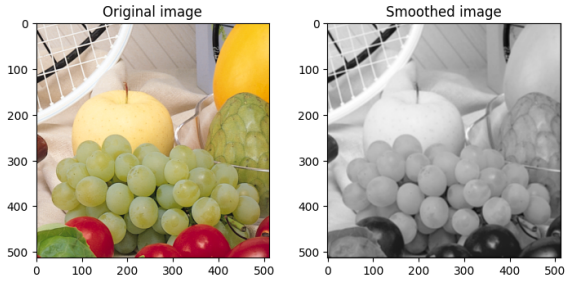

Fig. 2. Comparison of the original image and the smothed image.

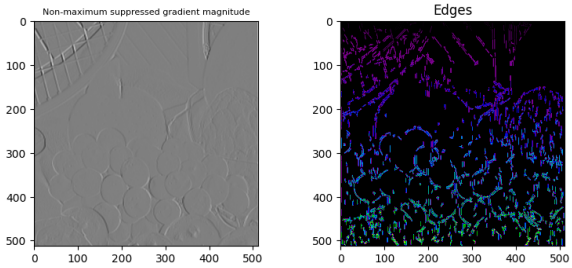

Fig. 3. The figure shows the result of applying NMS on the right and plotting edges with sh = 0.05 on the left. To show more edges, a small objects removal was performed with a threshold of 10.

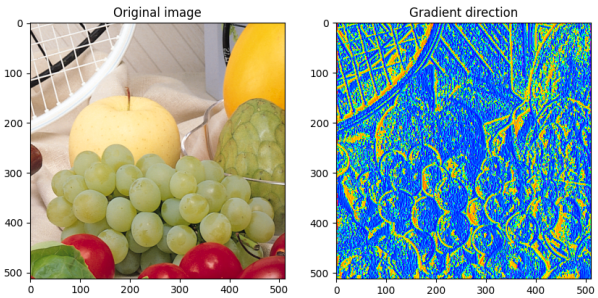

Fig. 4. Graph comparing original image and gradient direction.

When we use a larger sh value of 0.2 and set the minimum size of edges to be removed to 50 pixels, we observe that the resulting image contains fewer detected edges. This can be beneficial in situations where we are only interested in detecting strong and prominent edges in the image. However, it may also result in missing some weaker edges or details that are still important for some applications. The choice of sh and the minimum edge size to remove should be carefully considered based on the specific requirements of the image analysis task at hand.

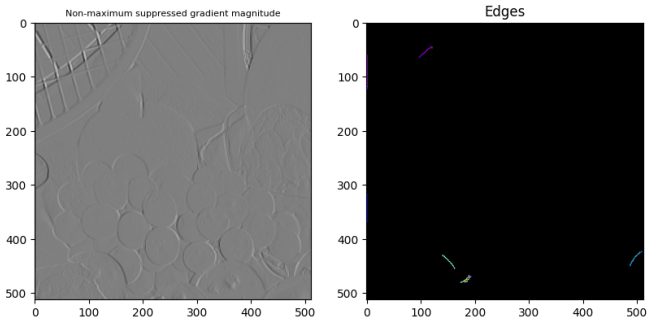

Fig. 5. Graph comparing original image and gradient direction.

V. CONCLUSIONS

The implementation of the Canny-Deriche filter was successful in detecting edges in the provided image. It is an effective variant of the Canny filter for edge detection in noisy images with intensity variations. The selection of the alpha parameter is critical for balancing edge localization and noise suppression, and thresholding reduces false positives. Future work can explore other variants of the Canny filter and apply object segmentation techniques. The Canny-Deriche filter can be used in computer vision applications such as object recognition and tracking. Overall, it is a robust and versatile method for edge detection in images.

References

[1] Canny, J. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[2] Deriche, R. Using Canny's criteria to derive a recursively implemented optimal edge detector. International Journal of Computer Vision, vol. 1, no. 2, pp. 167-187, 1987.

[3] Sarkar, S. and Boyer, K. A review of edge detection techniques. Computer Vision, Graphics, and Image Processing, vol. 38, no. 1, pp. 1-21, Oct. 1987.

[4] Zhang, Y., Liu, L., and Zhang, D. A Comparative Study of Canny and Deriche Edge Detection Algorithms. Proceedings of the 6th International Conference on Computer Graphics, Imaging and Visualization, pp. 108-113, Aug. 2009.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from skimage import io, filters, feature, morphology, color
from skimage.measure import label

# Define the Canny-Deriche smoothing filter
def canny_deriche_filter(s, alpha):
    n = 1
    si = np.linspace(-s, s, 2*s+1)
    h = np.exp(-alpha*np.abs(si)) * ((alpha*np.abs(si))**n) * ((1-np.exp(-alpha*np.abs(si))) /
((si**2) + 1e-10))
    h /= np.sum(h)
    return h, si

# Load the image
img = io.imread('C:/Users/ksevi/OneDrive/Desktop/MASTER/SCENE_LALIGANT/fourth report/fruits.png')
gray_img = color.rgb2gray(img)

# Define the filter parameters
alpha = 2
s = 10
h, si = canny_deriche_filter(s, alpha)

plt.plot(si[si>=0], h[si>=0])
plt.xlabel('s')
plt.ylabel('h(s)')
plt.title('Canny-Deriche filter (positive domain)')
plt.show()

# Truncate the support of the filter according to s
support = np.abs(si) <= s
h = h[support]
s = si[support]

# Apply the filter to the image
smoothed_img = signal.convolve2d(gray_img, np.transpose([h]), mode='same')
smoothed_img = signal.convolve2d(smoothed_img, [h], mode='same')

# Compute the gradient components
kernel = np.array([[-1, 1], [-1, 1]])
G = signal.convolve2d(smoothed_img, kernel, mode='same')
theta = np.arctan2(G[:, 1:], G[:, :-1])

# Compute the non-maximum suppressed gradient magnitude
nms = filters.unsharp_mask(G)

# Threshold the gradient magnitude
sh = 0.05
edges = (nms > sh*np.max(nms))

# Refine the segmentation with hysteresis thresholding
edges = feature.canny(nms, 0, sh*np.max(nms))

# Remove small objects
edges = morphology.remove_small_objects(edges, 10)
```

```
# Label connected components
labels = label(edges)

# Plot the results
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))
ax = axes.ravel()
ax[0].imshow(img)
ax[0].set_title('Original image')
ax[1].imshow(smoothed_img, cmap=plt.cm.gray)
ax[1].set_title('Smoothed image')
ax[2].imshow(nms, cmap=plt.cm.gray)
ax[2].set_title('Non-maximum suppressed gradient magnitude')
ax[3].imshow(labels, cmap=plt.cm.nipy_spectral)
ax[3].set_title('Edges')
plt.show()


# Plot theta
fig1, axes1 = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
ax1 = axes1.ravel()
ax1[0].imshow(img, cmap=plt.cm.gray)
ax1[0].set_title('Original image')
ax1[1].imshow(theta, cmap=plt.cm.hsv)
ax1[1].set_title('Gradient direction')
plt.show()
```