First report - Scene Segmentation and Interpretation

Kimberly Grace Sevillano C.[1]

***Subject*— Performance of different edge detectors against noise**

I. Objective

The objective of this report is to measure the performance of different edge detectors against noise using the PFOM technique - "Pratt's Figure of Merit".

A. Problem Description

Edge detection is a fundamental and widely used process in image analysis and computer vision applications. Therefore, an edge extractor must operate efficiently despite the various perturbations in the data that may be created during image acquisition. Consequently, a large number of edge detection algorithms have been developed and many methods are still being proposed. In this report we will compare 5 different types: Canny, Sobel, Roberts, Prewitt and Laplacian.

II. Methods

A. Canny Edge detector

The Canny edge detector is a multi-stage edge detection algorithm that is widely used in image processing and computer vision. It works by first smoothing the image to reduce noise, then finding the gradient magnitude and direction of the image intensity. Edges are identified as the pixels with local maxima in the gradient magnitude, and non-maximum suppression is performed to thin the edge map to single pixel width. Finally, hysteresis thresholding is applied to preserve only the edges that are strong and connected.

The Canny edge detector is considered to be an "optimal" edge detector because it balances the trade-off between detecting too many or too few edges, and it has good localization and detection accuracy.

B. Sobel Edge detector

The Sobel edge detector is a image processing technique used to detect edges in an image. It works by convolving the image with two 3x3 kernels, one for horizontal edges and one for vertical edges. The kernels are designed to highlight the differences in intensity between neighboring pixels, and the results are combined to produce a gradient magnitude representation of the image. The gradient magnitude is then thresholded to create a binary edge map.

The Sobel edge detector is a simple and fast edge detection algorithm that is widely used in computer vision and image processing. It is considered to be more robust than other simple edge detectors, such as the Roberts cross operator, but it may not be as effective as more advanced edge detection algorithms, such as the Canny edge detector.

C. Roberts Edge detector

The Roberts Cross edge detector is a basic image processing technique that uses two simple filters to detect the edges in an image. It works by subtracting the values of the pixels in a diagonal direction to highlight the changes in intensity. The filters are designed to highlight the difference in intensity between neighboring pixels, and the results are thresholded to create a binary edge map. The Roberts Cross edge detector is fast and simple, but it may not produce as clear of results as other edge detection methods, such as the Sobel or Canny edge detector.

D. Prewitt Edge detector

The Prewitt edge detector is a simple and fast edge detection algorithm that is similar to the Sobel edge detector. However, it uses different convolution kernels that are specifically designed to highlight the edges in an image. Like the Sobel edge detector, the Prewitt edge detector is considered to be a robust edge detection algorithm, but it may not be as effective as more advanced edge detection algorithms, such as the Canny edge detector.

E. Laplacian Edge detector

The Laplacian edge detector is an image processing technique used to detect edges in an image. It works by convolving the image with a Laplacian kernel, which is a second-order differentiation filter. The Laplacian kernel is designed to highlight the changes in intensity between neighboring pixels, and the result of the convolution is a representation of the gradient magnitude of the image. The gradient magnitude is then thresholded to create a binary edge map.

The Laplacian edge detector is considered to be a powerful edge detection algorithm, as it is able to detect both the gradient magnitude and direction of the edges in an image. It is often used in combination with other edge detection algorithms, such as the Canny edge detector, to improve the accuracy and robustness of the edge map. However, the Laplacian edge detector is sensitive to noise in the image, so it may need to be smoothed before being used to detect edges.

III. Method

As a performance measurement technique, we will use Pratt's of Figure of Merit. The Pratt Figure of Merit (FOM) is a performance evaluation technique used in image processing

[1]G. Sevillano, University of Burgundy, France

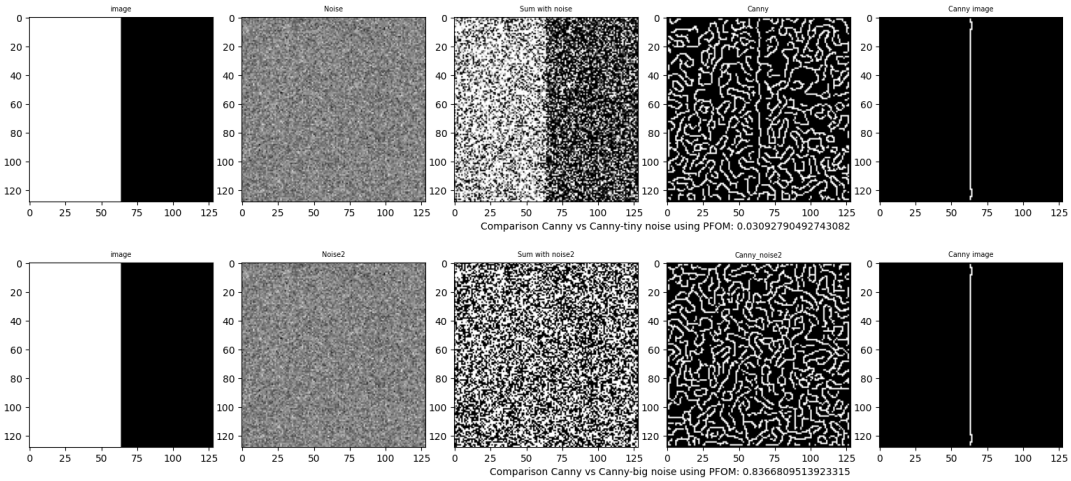


Fig. 1. Comparison of Canny edge detector with different noises

to assess the quality of edge detectors. The Pratt FOM is based on the concept of the true positive rate (TPR) and the false positive rate (FPR). TPR measures the fraction of true edges in the image that are correctly detected by the edge detector, while FPR measures the fraction of false edges in the image that are incorrectly detected by the edge detector.

The Pratt FOM is calculated as the geometric mean of TPR and (1-FPR), and it is used to measure the overall performance of an edge detector in terms of both accuracy and completeness. A high Pratt FOM indicates that the edge detector is able to accurately detect the edges in the image while minimizing the number of false edges detected. The Pratt FOM can be used to compare the performance of different edge detectors and to select the best edge detector for a given application. In Listing 1 the FOM function was created for its calculation.

```
DEFAULT_ALPHA = 1.0 / 9

def fom(img, img2, alpha = DEFAULT_ALPHA):
    dist = distance_transform_edt(img)
    fom = 1.0 / np.maximum(
        np.count_nonzero(img),
        np.count_nonzero(img2))
    N, M = img.shape
    for i in range(0, N):
        for j in range(0, M):
            if img[i, j]:
                fom += 1.0 / ( 1.0 + dist[i, j] *
    dist[i, j] * alpha)
    fom /= np.maximum(
        np.count_nonzero(img),
        np.count_nonzero(img2))
    return fom
```

Listing 1. Function for PFOM calculation in python

IV. Methodology

- First we create our delimited.
- We add Gaussian noise.
- Then we apply the edge detector filter we will use, add a threshold if necessary.
- Finally we compare the performance between filters and filters with noise.

V. Results

As the first PFOM result between the comparison of the Canny filter and a Canny filter with small noise we have 0.03092 and the comparison between the Canny filter with small noise and a Canny filter with large noise we have 0.83668. (As shown in 1).

This indicates that the result of the Canny algorithm with large noise has more detected and localized edges than the one with small noise.

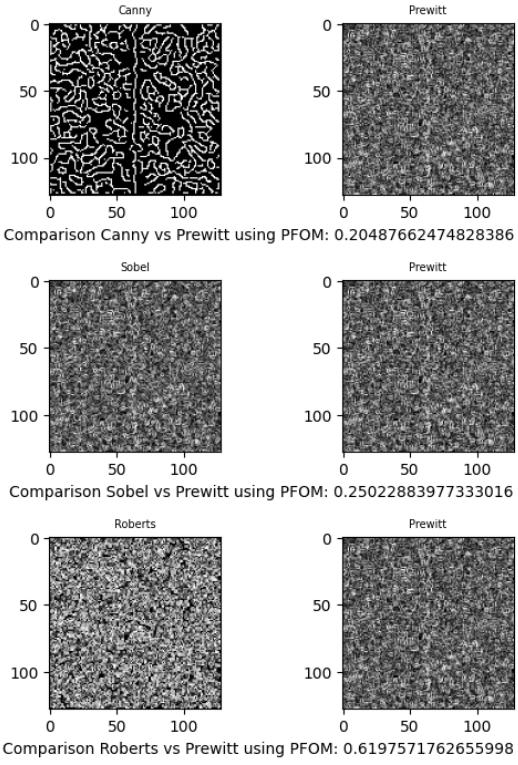


Fig. 2. Comparison of Canny, Sobel and Roberts filters versus the Prewitt filter.

Other results obtained between the comparison of the Canny, Sobel and Roberts filters versus the first Prewitt filter and then versus the Laplacian filter will be shown in 2 and 3.

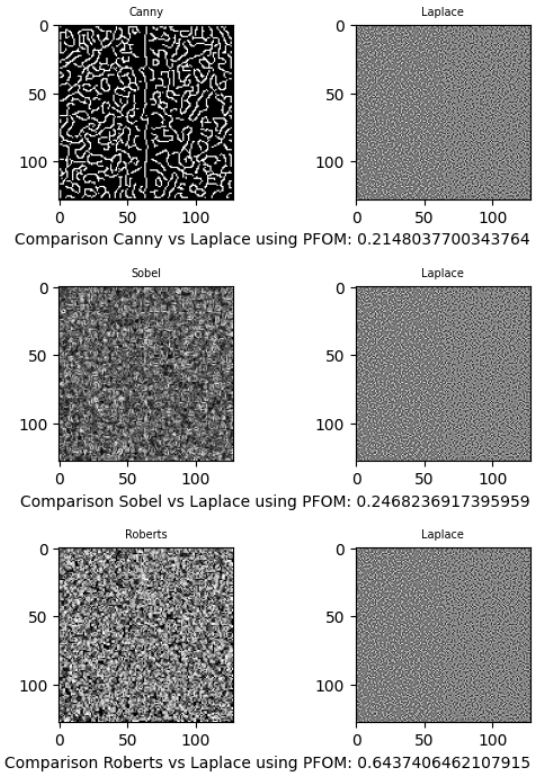
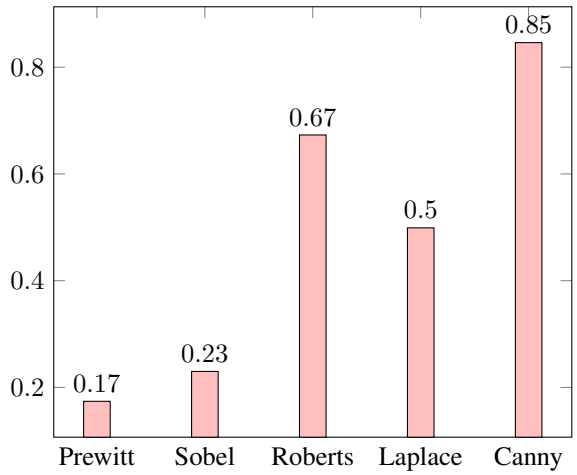


Fig. 3. Comparison of Canny, Sobel and Roberts filters versus the Laplacian filter.

VI. CONCLUSIONS

One of the conclusions that we can obtain after comparing each of the filters with small noise and its corresponding with big noise, is that the best performance was obtained by the canny filter and on the contrary the one that had the worst performance was the Prewitt.



REFERENCES

[1] Sadiq, B.O., Sani, S.M., & Garba, S. (2015). Edge Detection: A Collection of Pixel based Approach for Colored Images. ArXiv, abs/1503.05689.

[2] Javier Gimenez, Jorge Martinez, Ana Georgina Flesia, Unsupervised edge map scoring: A statistical complexity approach, Computer Vision and Image Understanding, Volume 122, 2014, Pages 131-142,

[3] Baptiste Magnier. Edge Detection Evaluation: A New Normalized Figure of Merit. ICASSP 20192019 IEEE International Conference on Acoustics, Speech and Signal Processing, May 2019, Brighton, United Kingdom. pp.2407-2411,

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage import feature,filters
from scipy import ndimage
from scipy.ndimage import distance_transform_edt

imag = np.zeros((128, 128), dtype=float)
imag[:, :-64] = 1
noise =  np.random.normal(loc=0, scale=1, size=imag.shape)
noise2 =noise*20

# noise overlaid over image
noisy = np.clip((imag + noise),0,1)
noisy2 = np.clip((imag + noise2),0,1)

Canny_1 = feature.canny(noisy,sigma=2)
Canny_img = feature.canny(imag,sigma=2) # CANNY REAL IMAGE
Canny_2 = feature.canny(noisy2,sigma=2)

Sobel_1 = filters.sobel(noisy)
Sobel_2 = filters.sobel(imag)
Sobel_3 = filters.sobel(noisy2)

Roberts_1 = filters.roberts(noisy)
Roberts_2 = filters.roberts(imag)
Roberts_3 = filters.roberts(noisy2)
#--------- propuestos------------
Prewitt_1 = filters.prewitt(noisy)
Prewitt_2 = filters.prewitt(imag)
Prewitt_3 = filters.prewitt(noisy2)

Laplace_1 = ndimage.laplace(noisy)
Laplace_2 = ndimage.laplace(imag)
Laplace_3 = ndimage.laplace(noisy2)

DEFAULT_ALPHA = 1.0 / 9

def fom(img, img2, alpha = DEFAULT_ALPHA):
    dist = distance_transform_edt(img)
    fom = 1.0 / np.maximum(
        np.count_nonzero(img),
        np.count_nonzero(img2))
    N, M = img.shape
    for i in range(0, N):
        for j in range(0, M):
            if img[i, j]:
                fom += 1.0 / ( 1.0 + dist[i, j] * dist[i, j] * alpha)
    fom /= np.maximum(
        np.count_nonzero(img),
        np.count_nonzero(img2))
    return fom

f1 = fom(Canny_img,Canny_1)

f9 = fom(Prewitt_1,Prewitt_3)
```

```
f10 = fom(Sobel_1, Sobel_3)
f11 = fom(Roberts_1,Roberts_3)
f12 = fom(Laplace_1, Laplace_3)
f2 = fom(Canny_1, Canny_2)

fig4, ax4 = plt.subplots( figsize=(10, 7))
values=[f9,f10,f11,f12,f2 ]
index=['Prewitt', 'Sobel', 'Roberts', 'Laplace','Canny']
#total = data.sum(axis=1)
ax4.bar(index, values)

plt.show()

print("Comparison Prewitt tiny vs Prewitt-big noise: ",f9)
print("Comparison Sobel tiny vs Sobel-big noise: ",f10)
print("Comparison Roberts tiny vs Roberts-big noise: ",f11)
print("Comparison Laplace tiny vs Laplace-big noise: ",f12)
print("Comparison Canny tiny vs Canny-big noise: ",f2)
print(" ")
fig1, ax1 = plt.subplots(nrows=2, ncols=5, figsize=(10, 5))
####----CANNY-----------------
ax1[0][0].imshow(imag,cmap='gray')
ax1[0][0].set_title('image', fontsize=7)

ax1[0][1].imshow(noise,cmap='gray')
ax1[0][1].set_title('Noise', fontsize=7 )

ax1[0][2].imshow(noisy,cmap='gray')
ax1[0][2].set_title('Sum with noise', fontsize=7 )

ax1[0][3].imshow(Canny_1,cmap='gray')
ax1[0][3].set_title('Canny', fontsize=7 )
ax1[0][3].set_xlabel(f'Comparison Canny vs Canny-tiny noise using PFOM: {f1}',loc='right')

ax1[0][4].imshow(Canny_img,cmap='gray')
ax1[0][4].set_title('Canny image', fontsize=7 )

#--------CANNY with big noise--------------
ax1[1][0].imshow(imag, cmap='gray')
ax1[1][0].set_title('image', fontsize=7 )

ax1[1][1].imshow(noise2, cmap='gray')
ax1[1][1].set_title('Noise2', fontsize=7 )

ax1[1][2].imshow(noisy2, cmap='gray')
ax1[1][2].set_title('Sum with noise2', fontsize=7 )

ax1[1][3].imshow(Canny_2, cmap='gray')
ax1[1][3].set_title('Canny_noise2', fontsize=7 )
ax1[1][3].set_xlabel(f'Comparison Canny vs Canny-big noise using PFOM: {f2}',loc='right')

ax1[1][4].imshow(Canny_img, cmap='gray')
ax1[1][4].set_title('Canny image', fontsize=7 )

f3 = fom(Canny_1,Prewitt_1)
f4 = fom(Sobel_1, Prewitt_1)
```

```
f5 = fom(Roberts_1,Prewitt_1)
f6 = fom(Canny_1,Laplace_1)
f7 = fom(Sobel_1,Laplace_1)
f8 = fom(Roberts_1,Laplace_1)

##COMPARACIONES
fig2, ax2 = plt.subplots(nrows=3, ncols=2, figsize=(10, 5))
#-----------------canny vs prewitt
ax2[0][0].imshow(Canny_2, cmap='gray')
ax2[0][0].set_title('Canny', fontsize=7 )

ax2[0][1].imshow(Prewitt_3, cmap='gray')
ax2[0][1].set_title('Prewitt', fontsize=7 )
ax2[0][1].set_xlabel(f'Comparison Canny vs Prewitt using PFOM: {f3}',loc='right')
#-------sobel vs prewitt
ax2[1][0].imshow(Sobel_3, cmap='gray')
ax2[1][0].set_title('Sobel', fontsize=7 )

ax2[1][1].imshow(Prewitt_3, cmap='gray')
ax2[1][1].set_title('Prewitt', fontsize=7 )
ax2[1][1].set_xlabel(f'Comparison Sobel vs Prewitt using PFOM: {f4}',loc='right')
#---------------Robert vs prewitt
ax2[2][0].imshow(Roberts_3, cmap='gray')
ax2[2][0].set_title('Roberts', fontsize=7 )

ax2[2][1].imshow(Prewitt_3, cmap='gray')
ax2[2][1].set_title('Prewitt', fontsize=7 )
ax2[2][1].set_xlabel(f'Comparison Roberts vs Prewitt using PFOM: {f5}',loc='right')

fig3, ax3 = plt.subplots(nrows=3, ncols=2, figsize=(10, 5))
#-----------------CANNY VS LAPLACE -----------
ax3[0][0].imshow(Canny_2, cmap='gray')
ax3[0][0].set_title('Canny', fontsize=7 )
ax3[0][1].imshow(Laplace_3, cmap='gray')
ax3[0][1].set_title('Laplace', fontsize=7 )
ax3[0][1].set_xlabel(f'Comparison Canny vs Laplace using PFOM: {f6}',loc='right')
#-----------------SOBEL VS LAPLACE -----------
ax3[1][0].imshow(Sobel_3, cmap='gray')
ax3[1][0].set_title('Sobel', fontsize=7 )
ax3[1][1].imshow(Laplace_3, cmap='gray')
ax3[1][1].set_title('Laplace', fontsize=7 )
ax3[1][1].set_xlabel(f'Comparison Sobel vs Laplace using PFOM: {f7}',loc='right')
#-----------------ROBERTS  VS LAPLACE -----------
ax3[2][0].imshow(Roberts_3, cmap='gray')
ax3[2][0].set_title('Roberts', fontsize=7 )
ax3[2][1].imshow(Laplace_3, cmap='gray')
ax3[2][1].set_title('Laplace', fontsize=7 )
ax3[2][1].set_xlabel(f'Comparison Roberts vs Laplace using PFOM: {f8}',loc='right')

print("Comparison Canny vs Canny-tiny noise: ",f1)
print("Comparison Canny vs Canny-big noise: ",f2)
print(" ")
print("Comparison Canny vs Prewitt: ",f3)
print("Comparison Sobel vs Prewitt: ",f4)
print("Comparison Roberts vs Prewitt: ",f5)
print(" ")
```

```
print("Comparison Canny vs Laplace: ",f6)
print("Comparison Sobel vs Laplace: ",f7)
print("Comparison Roberts vs Laplace: ",f8)

fig1.tight_layout(pad=1, w_pad=1, h_pad=2)
fig2.tight_layout()
fig3.tight_layout()
plt.show()
```