

MODUL DETEKSI TEPI (Edge Detection)

1. TUJUAN

- Mahasiswa mampu memahami proses deteksi tepi dengan turunan pertama dan kedua
- Mahasiswa mampu melakukan operasi filter dengan turunan pertama dan kedua
- Mahasiswa mampu menganalisis perbedaan antara setiap filter yang diterapkan

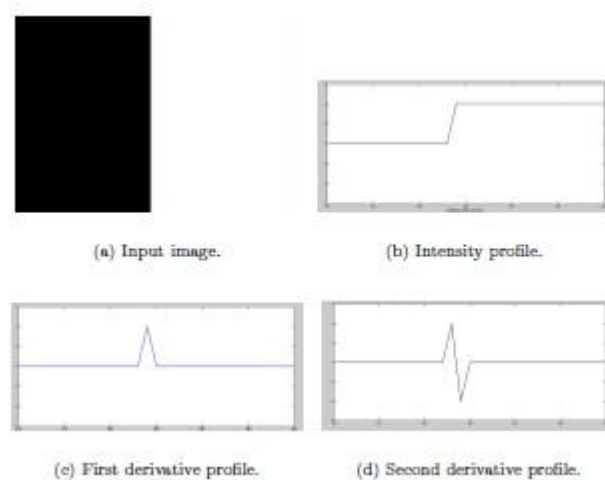
2. PENDAHULUAN

Tepi adalah himpunan titik-titik pada suatu citra yang mengalami perubahan intensitas antara satu sisi titik tersebut dengan sisi lainnya. Dari kalkulus, kita tahu bahwa perubahan intensitas dapat diukur dengan menggunakan turunan pertama atau kedua. Pertama, kita pelajari bagaimana perubahan intensitas mempengaruhi turunan pertama dan kedua dengan mempertimbangkan gambar sederhana dan profil yang sesuai. Metode ini akan menjadi dasar untuk menggunakan filter turunan pertama dan kedua untuk deteksi tepi.

Gambar 1(a) adalah gambar input dalam skala abu-abu. Sisi kiri gambar gelap sedangkan sisi kanan terang. Saat melintasi dari kiri ke kanan, di persimpangan antara dua wilayah, intensitas piksel berubah dari gelap ke terang. Gambar 1(b) adalah profil intensitas yang melintas horizontal dari gambar input. Perhatikan bahwa pada titik transisi dari daerah gelap ke daerah terang, terjadi perubahan intensitas pada profil. Jika tidak, intensitasnya konstan di daerah gelap dan terang. Untuk lebih jelasnya, hanya daerah di sekitar titik transisi yang ditunjukkan pada profil intensitas (Gambar 1(b)), turunan pertama (Gambar 1(c)), dan turunan kedua (Gambar 1(d)). Di daerah transisi, karena profil intensitas meningkat, turunan pertama adalah positif, sedangkan di daerah gelap dan terang menjadi nol. Turunan pertama memiliki maksimum atau puncak di tepi. Karena turunan pertama naik sebelum tepi, turunan kedua positif sebelum tepi.

Demikian juga, karena turunan pertama menurun setelah tepi, turunan kedua negatif setelah tepi. Juga, turunan kedua adalah nol di daerah gelap dan terang karena turunan pertama yang sesuai adalah nol. Di tepi, turunan kedua adalah nol. Fenomena turunan kedua ini mengubah tanda dari positif sebelum tepi menjadi negatif setelah tepi atau sebaliknya dikenal sebagai *zero-crossing*, karena mengambil nilai nol di tepi. Gambar input disimulasikan di komputer dan tidak memiliki noise. Namun, gambar yang diperoleh akan memiliki noise yang dapat memengaruhi deteksi *zerocrossing*. Juga, jika intensitas berubah dengan cepat dalam profil, tepi palsu akan dideteksi oleh *zero-crossing*. Untuk mencegah masalah karena *noise* atau intensitas yang berubah dengan

cepat, gambar diproses terlebih dahulu sebelum penerapan filter turunan kedua. Penjelasan ini menjelaskan alasan penggunaan turunan pertama dan kedua dalam deteksi tepi.



Gambar 1. contoh *zero-crossing*

3. LANGKAH PRAKTIKUM

A. Filter Sobel

Salah satu filter turunan pertama yang paling populer adalah filter Sobel. Filter atau topeng Sobel itu digunakan untuk mencari tepi horizontal dan vertikal seperti yang diberikan pada Gambar 2.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Gambar 2 . Filter Sobel Fitur

penting dari filter Sobel adalah:

- Jumlah koefisien pada gambar topeng adalah 0. Ini berarti bahwa piksel dengan skala abu-abu konstan tidak terpengaruh oleh filter turunan.
- Efek samping dari filter turunan adalah terciptanya noise tambahan. Oleh karena itu, koefisien +2 dan 2 digunakan dalam gambar topeng untuk menghasilkan *smoothing*.

Latihan 1: Berikut ini adalah kode Python untuk filter Sobel:

```
Import Library

import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage import data

[1] Python

Read Input Image

img = data.camera()

[2] Python

Proses Sobel

#sobel
img_sobelx = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)
img_sobely = cv2.Sobel(img,cv2.CV_8U,0,1,ksize=5)
img_sobel = img_sobelx + img_sobely

[3] Python

Plot Output Image

fig, axes = plt.subplots(4, 2, figsize=(20, 20))
ax = axes.ravel()

ax[0].imshow(img, cmap = 'gray')
ax[0].set_title("Citra Input")
ax[1].hist(img.ravel(), bins = 256)
ax[1].set_title("Histogram Citra Input")

ax[2].imshow(img_sobelx, cmap = 'gray')
ax[2].set_title("Citra Output")
ax[3].hist(img_sobelx.ravel(), bins = 256)
ax[3].set_title("Histogram Citra Output")

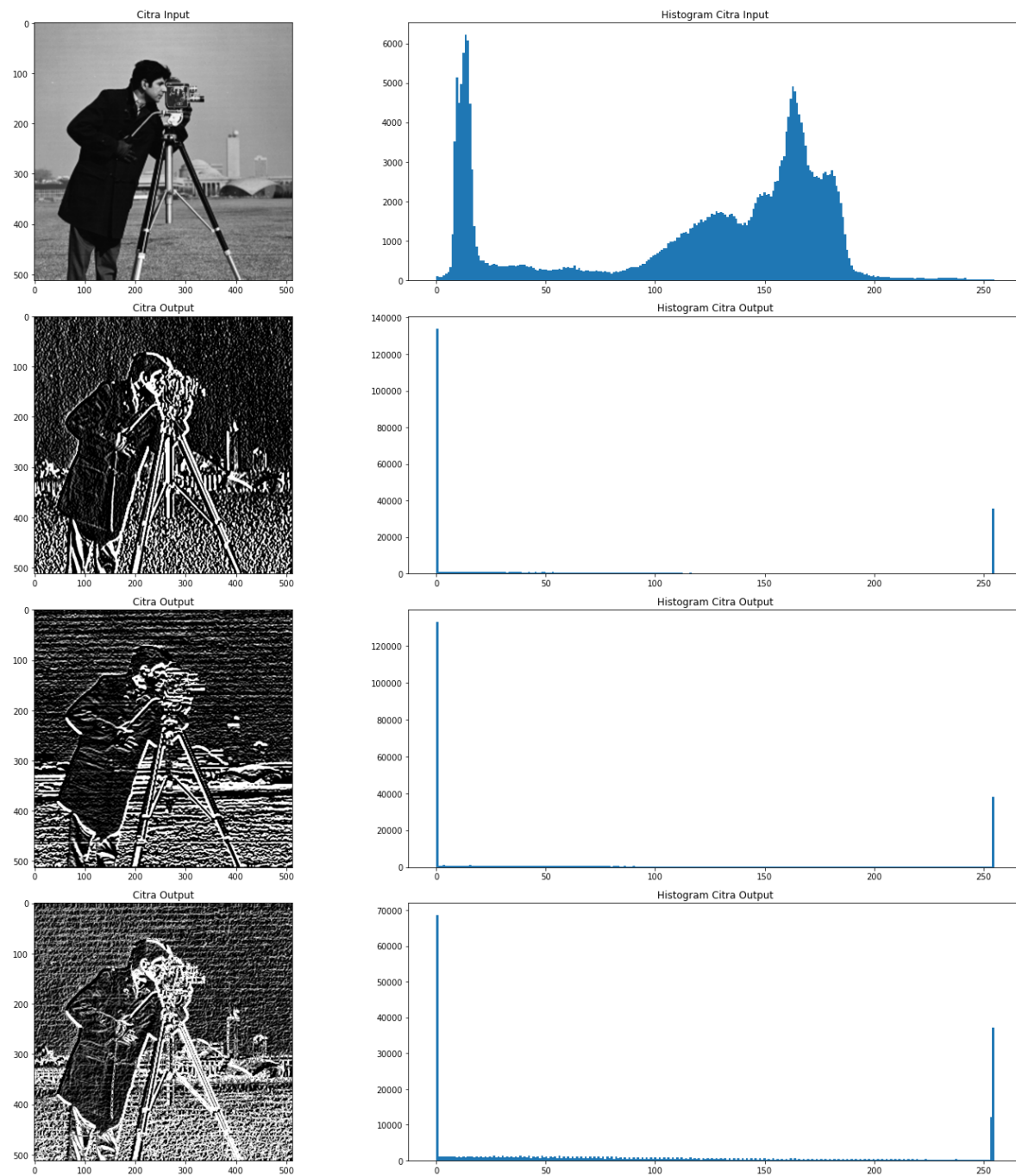
ax[4].imshow(img_sobely, cmap = 'gray')
ax[4].set_title("Citra Output")
ax[5].hist(img_sobely.ravel(), bins = 256)
ax[5].set_title("Histogram Citra Output")

ax[6].imshow(img_sobel, cmap = 'gray')
ax[6].set_title("Citra Output")
ax[7].hist(img_sobel.ravel(), bins = 256)
ax[7].set_title("Histogram Citra Output")

fig.tight_layout()

[4] Python
```

Output yang ditampilkan:



B. Filter Prewitt

Filter turunan pertama yang populer lainnya adalah Prewitt . Filter untuk filter Prewitt diberikan pada Gambar 3.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Gambar 3. Filter Prewitt

Seperti dalam kasus filter Sobel, jumlah koefisien di Prewitt juga 0. Oleh karena itu filter ini tidak mempengaruhi piksel dengan skala abu-abu konstan. Namun, filter tidak mengurangi noise seperti filter Sobel. Untuk Prewitt, argumen fungsi Python mirip dengan argumen fungsi Sobel.

Latihan 2. Berikut ini adalah kode Python untuk filter Prewitt:

```

Import Library

import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage import data

[1]

Read Input Image

img = data.camera()

[2]

Proses Prewitt

#prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])

img_prewittx = cv2.filter2D(img, -1, kernelx)
img_prewitty = cv2.filter2D(img, -1, kernely)
img_prewitt = img_prewittx + img_prewitty

[3]

```

Plot Citra Output

```
fig, axes = plt.subplots(4, 2, figsize=(20, 20))
ax = axes.ravel()

ax[0].imshow(img, cmap = 'gray')
ax[0].set_title("Citra Input")
ax[1].hist(img.ravel(), bins = 256)
ax[1].set_title("Histogram Citra Input")

ax[2].imshow(img_prewittx, cmap = 'gray')
ax[2].set_title("Citra Output Prewitt X")
ax[3].hist(img_prewittx.ravel(), bins = 256)
ax[3].set_title("Histogram Citra Output Prewitt X")

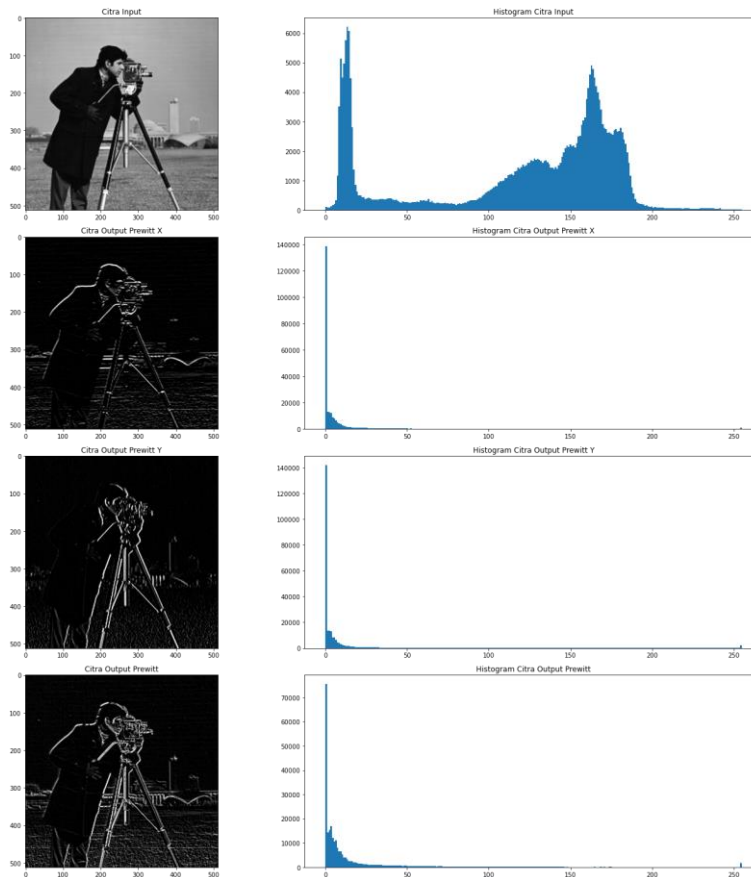
ax[4].imshow(img_prewitty, cmap = 'gray')
ax[4].set_title("Citra Output Prewitt Y")
ax[5].hist(img_prewitty.ravel(), bins = 256)
ax[5].set_title("Histogram Citra Output Prewitt Y")

ax[6].imshow(img_prewitt, cmap = 'gray')
ax[6].set_title("Citra Output Prewitt")
ax[7].hist(img_prewitt.ravel(), bins = 256)
ax[7].set_title("Histogram Citra Output Prewitt")

fig.tight_layout()
```

[6]

Output dari kode program diatas seperti di bawah ini:

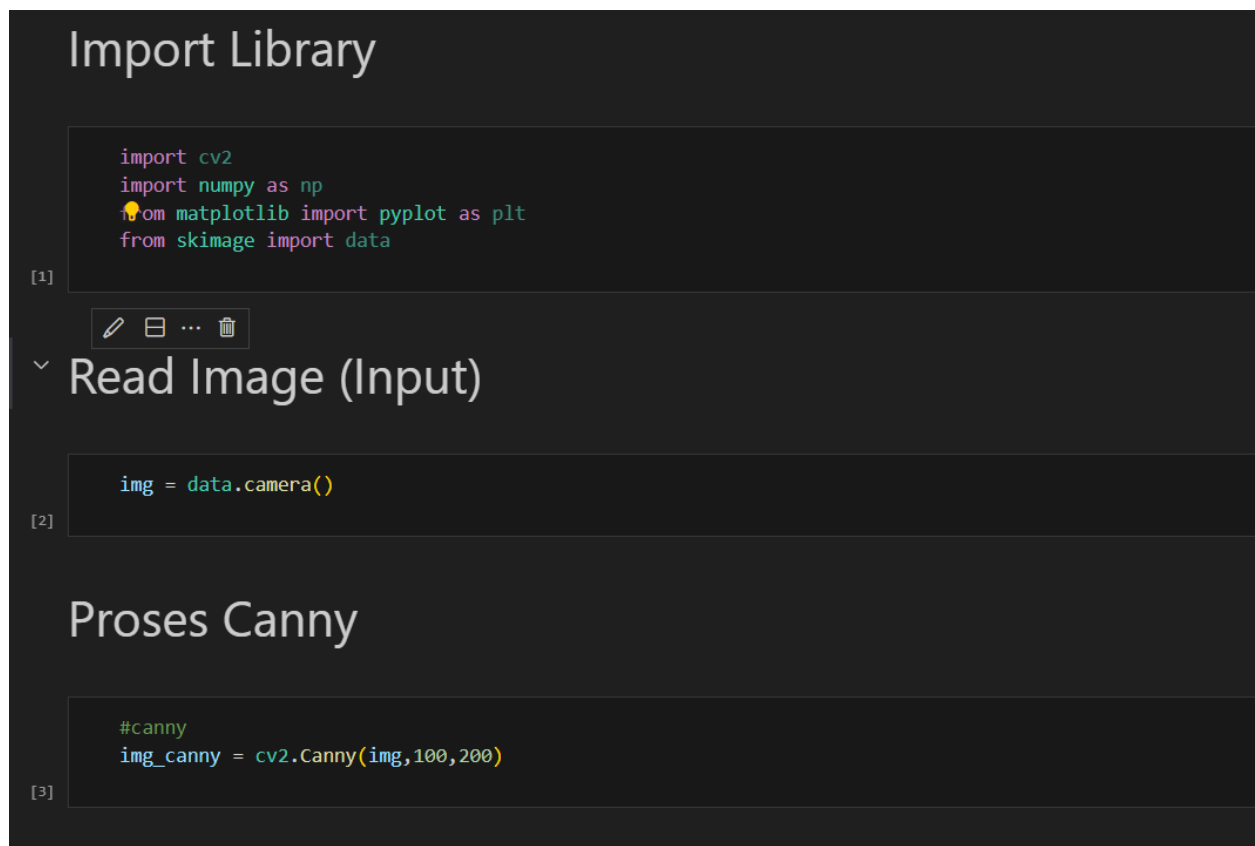


C. Filter Canny

Salah satu operator deteksi tepi modern adalah deteksi tepi dengan operator Canny. Deteksi tepi *canny* ditemukan oleh Marr dan Hildreth yang meneliti pemodelan persepsi visual manusia. Ada beberapa kriteria pendeteksi tepian paling optimum yang dapat dipenuhi oleh operator *canny*:

- Mendeteksi dengan baik, kemampuan untuk meletakkan dan menandai semua tepi yang ada sesuai dengan pemilihan parameter-parameter konvolusi yang dilakukan. Sekaligus juga memberikan fleksibilitas yang sangat tinggi dalam hal menentukan tingkat deteksi ketebalan tepi sesuai yang diinginkan.
- Melokalisasi dengan baik, Dengan Canny dimungkinkan dihasilkan jarak yang minimum antara tepi yang dideteksi dengan tepi yang asli.
- Respon yang jelas ,Hanya ada satu respon untuk tiap tepi, sehingga mudah dideteksi dan tidak menimbulkan kerancuan pada pengolahan citra selanjutnya. Pemilihan parameter deteksi tepi Canny sangat mempengaruhi hasil dari tepian yang dihasilkan. Beberapa parameter tersebut adalah nilai standart deviasi Gaussian dan nilai ambang (*threshold*).

Latihan 3. Berikut ini adalah kode Python untuk filter Canny:



```
Import Library

import cv2
import numpy as np
from matplotlib import pyplot as plt
from skimage import data

[1]

Read Image (Input)

img = data.camera()

[2]

Proses Canny

#canny
img_canny = cv2.Canny(img,100,200)

[3]
```

Plot Citra Output

```
fig, axes = plt.subplots(2, 2, figsize=(20, 20))
ax = axes.ravel()

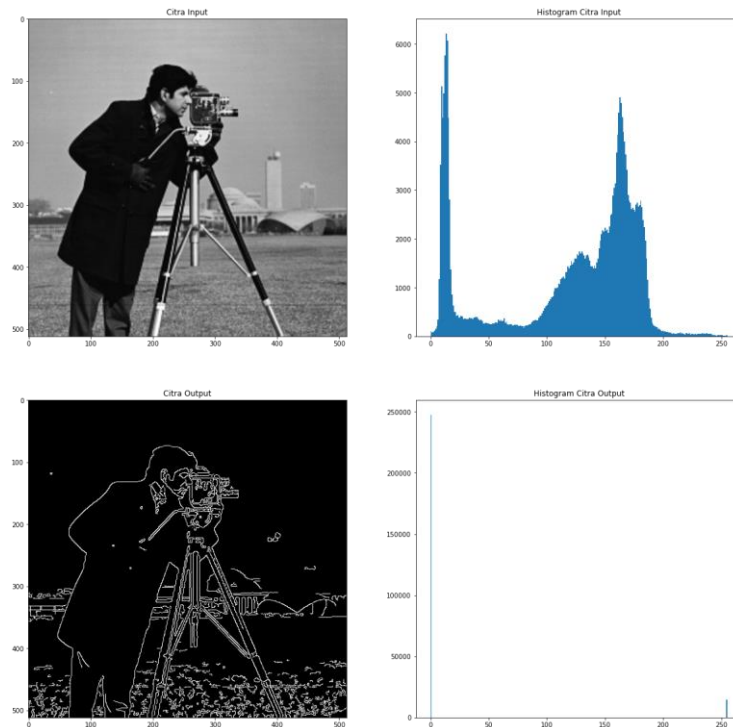
ax[0].imshow(img, cmap = 'gray')
ax[0].set_title("Citra Input")
ax[1].hist(img.ravel(), bins = 256)
ax[1].set_title("Histogram Citra Input")

ax[2].imshow(img_canny, cmap = 'gray')
ax[2].set_title("Citra Output")
ax[3].hist(img_canny.ravel(), bins = 256)
ax[3].set_title("Histogram Citra Output")
```

[5]

```
... Text(0.5, 1.0, 'Histogram Citra Output')
```

Output dari kode program diatas adalah sebagai berikut:



D. FILTER TURUNAN KEDUA

Seperti namanya, dalam filter turunan kedua, filter turunan kedua dihitung untuk menentukan tepi. Karena memerlukan komputasi turunan dari citra turunan, maka secara komputasi lebih kompleks dibandingkan dengan filter turunan pertama. Berbeda dengan detektor

tepi Sobel, detektor tepi Laplacian hanya menggunakan satu kernel. Laplacian menghitung turunan orde kedua dalam satu lintasan.

Kernel yang digunakan dalam pendeteksian Laplacian ini terlihat seperti ini:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Jika kita ingi memperhitungkan diagonal, maka filternya adalah :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Latihan 4. Berikut ini adalah kode Python untuk filter Laplacian:

```
1  import cv2
2  from matplotlib import pyplot as plt
3  #baca gambar
4  img0 = cv2.imread('grayscale.jpg',)
5  # konversi ke gray
6  gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)
7  # hilangkan noise
8  img = cv2.GaussianBlur(gray, (3, 3), 0)
9  # konvolusi dengan kernel
10 laplacian = cv2.Laplacian(img, cv2.CV_64F)
11 #tampilkan dengan matplotlib
12 plt.subplot(1, 2, 1), plt.imshow(img, cmap = 'gray')
13 plt.title('Original'), plt.xticks([], plt.yticks([]))
14 plt.subplot(1, 2, 2), plt.imshow(laplacian, cmap = 'gray')
15 plt.title('Laplacian'), plt.xticks([], plt.yticks([]))
16 plt.show()
```

Atau

```
1  import cv2
2
3  # load citra
4  img = cv2.imread('grayscale.jpg', 0)
5
6  # aplikasikan gaussian
7  blur = cv2.GaussianBlur(img, (3, 3), 0)
8
9  # aplikasikan laplacian
10 laplacian = cv2.Laplacian(blur, cv2.CV_64F)
11 laplacian1 = laplacian/laplacian.max()
12
13 cv2.imshow('laplacian-gaussian', laplacian1)
14 cv2.waitKey(0)
```

bisa juga dengan menggunakan fungsi imwrite untuk menyimpan gambar hasil.

Catatan :

Pada bagian ini silahkan load gambar dari komputer masing-masing. Maka pada latihan 4 tidak menampilkan output , karena citra hasil memiliki karakteristik yang berbeda beda untuk setiap gambar input.

E. TUGAS

- Lakukan proses seperti kode program di atas, display hasil boleh menggunakan cv2 dengan operator imshow atau imwrite atau menggunakan library matplotlib □
Bandingkan output masing masing filter.
- Lakukan analisis perbedaan.

File yang dikumpulkan:

- Kode program ipynb/py
- File word berisi screenshoot, link github dan analisis perbedaan hasil dari pemahaman yg kalian tangkap dari latihan yang dilakukan.
- Format nama file: nim_nama
- Agar drive LMS tidak penuh disarankan diupload di drive lalu attach link saat pengumpulan tugas. Pastikan link dapat diakses oleh semua orang.

Referensi :

- Ravishankar Chityala, Sridevi Pudipeddi, ***Image Processing and Acquisition using Python***, Second Edition, Chapman & Hall/CRC, 2020
- Himanshu Singh, ***Practical Machine Learning and Image Processing***, Allahabad, Uttar Pradesh, India, Apress, 2019
- Jan Erik Solem, ***Programming Computer Vision with Python***, 2012