Laporan Tugas II

Spatial Filter

Name:	**Ikhwanul Abiyu Dhiyya'ul Haq**
NRP:	5024211048
Lecturer:	Dr. Eko Mulyanto Yuniarno, S.T.,M.T.
Subject:	Image & Video Processing

Table of Contents

1 Pendahuluan

Filter spasial merupakan salah satu teknik fundamental dalam pengolahan citra digital yang digunakan untuk mengubah karakteristik citra dengan memanipulasi nilai piksel pada citra. Filter spasial adalah suatu matriks yang digunakan untuk melakukan operasi konvolusi pada citra, dengan tujuan utama untuk meningkatkan atau mengurangi fitur-fitur tertentu dalam citra, seperti menghasilkan efek blur, mengidentifikasi tepi, atau mempertajam citra.

Dalam laporan makalah ini, kami akan membahas implementasi filter spasial pada citra menggunakan teknik konvolusi 2 dimensi. Terdapat tiga jenis filter spasial yang akan dibahas, yaitu:

1. **Low Pass Filter untuk Pengaburan (Blur):** Filter ini digunakan untuk mengurangi detail dan menghasilkan efek pengaburan pada citra. Salah satu jenis yang akan kita bahas adalah filter Gaussian, yang secara efektif menghaluskan citra dengan menghilangkan noise dan mengaburkan detail yang tidak diinginkan.

2. **High Pass Filter untuk Identifikasi Tepi (Canny Edge):** High pass filter digunakan untuk menonjolkan tepi dan detail dalam citra. Salah satu teknik yang akan kita eksplorasi adalah metode Canny Edge Detection, yang memungkinkan kita untuk menemukan tepi yang tajam dalam citra dengan lebih akurat.

Dalam eksperimen kami, kita akan mencoba mengimplementasikan filter-filter ini menggunakan OpenCV, sebuah library populer untuk pemrosesan citra, dan juga dengan metode manual yang melibatkan perhitungan piksel per piksel. Dengan membandingkan kedua metode ini, kita akan mendapatkan pemahaman yang lebih baik tentang cara kerja filter spasial dan perbedaan antara implementasi otomatis dan manual.

Laporan ini akan menguraikan langkah-langkah yang diambil dalam proses implementasi, serta hasil dari eksperimen yang dilakukan, seperti perbandingan visual antara citra sebelum dan sesudah diterapkan filter spasial. Kami juga akan membahas kelebihan dan kekurangan dari masing-masing pendekatan yang digunakan, serta bagaimana filter spasial dapat diterapkan dalam berbagai konteks dalam pengolahan citra.

Dengan memahami konsep filter spasial dan kemampuan untuk mengimplementasikannya, diharapkan laporan ini akan memberikan wawasan yang lebih dalam tentang pengolahan citra digital dan aplikasinya dalam berbagai bidang seperti pengolahan gambar medis, pengolahan citra satelit, dan banyak lagi.

2 Metode

Dalam bagian ini, kami akan menjelaskan secara rinci metode yang digunakan untuk mengimplementasikan filter spasial pada citra. Kami akan memaparkan rumus-rumus yang digunakan dalam proses konvolusi dan juga memberikan gambaran umum tentang algoritma yang digunakan dalam filter spasial.

2.1 Konvolusi 2D

Konvolusi adalah operasi dasar dalam filter spasial. Operasi ini melibatkan perhitungan hasil dari persamaan konvolusi 2D yang telah disebutkan sebelumnya:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) \cdot f(x+s, y+t)$$

Di mana:
- $g(x,y)$ adalah nilai piksel pada citra hasil konvolusi di titik (x,y).
- a dan b adalah parameter yang mengatur ukuran jendela konvolusi.
- $w(s,t)$ adalah filter spasial yang digunakan.
- $f(x+s, y+t)$ adalah nilai piksel asli di citra input pada posisi $(x+s, y+t)$.

Proses ini diulangi untuk setiap piksel pada citra input, dengan jendela konvolusi berpindah ke setiap lokasi piksel.

2.2 Filter Spasial Low Pass Gaussian (Gaussian Blur)

Salah satu jenis filter spasial yang digunakan adalah filter Gaussian untuk menghasilkan efek pengaburan pada citra. Filter Gaussian didefinisikan sebagai berikut:

$$w(s,t) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{s^2+t^2}{2\sigma^2}}$$

Di mana:
- $w(s,t)$ adalah bobot pada posisi (s,t) dalam filter Gaussian.
- σ adalah parameter yang mengontrol seberapa tajam atau lebar distribusi Gaussian.

Filter ini digunakan untuk mengaburkan citra dengan mengurangi detail dan menghilangkan noise.

2.3 Filter Spasial High Pass (Operator Laplacian)

Operator Laplacian adalah salah satu operator yang digunakan dalam filter spasial high pass untuk deteksi tepi. Ini mengukur perubahan kedua intensitas piksel dalam citra. Operator Laplacian didefinisikan sebagai berikut:

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

Di sini:
- $\nabla^2 f(x,y)$ adalah nilai Laplacian pada titik (x,y) dalam citra.
- $\frac{\partial^2 f(x,y)}{\partial x^2}$ dan $\frac{\partial^2 f(x,y)}{\partial y^2}$ adalah turunan parsial orde kedua terhadap koordinat x dan y.

Operator Laplacian digunakan untuk menonjolkan perubahan kedua intensitas tingkat dalam citra, yang sering terkait dengan tepi atau detail yang tajam. Setelah menerapkan operator Laplacian, Anda dapat memproses hasilnya dengan ambang batas (thresholding) atau teknik lainnya untuk mendeteksi tepi dalam citra.

2.4 Implementasi dengan OpenCV dan Manual

Filter-filter ini diimplementasikan menggunakan perpustakaan OpenCV untuk pemrosesan citra. Selain itu, kami juga melakukan implementasi manual dengan melakukan perhitungan piksel per piksel sesuai dengan rumus konvolusi yang telah dijelaskan di atas.

Dalam eksperimen kami, kami akan membandingkan hasil implementasi dengan menggunakan metode OpenCV dan metode manual untuk memahami perbedaan antara kedua pendekatan ini.

Dengan demikian, bagian metode ini menjelaskan langkah-langkah dan rumus-rumus yang digunakan dalam mengimplementasikan filter spasial, baik itu filter Gaussian untuk pengaburan maupun metode Operator Laplacian untuk identifikasi tepi dalam citra. Dengan pemahaman yang mendalam tentang metode ini, kita dapat memahami bagaimana filter spasial dapat digunakan dalam pemrosesan citra untuk berbagai tujuan.

3 Sumber Kode

Dalam bagian ini, akan disajikan kode sumber untuk melakukan Konvolusi 2D, Filter Spasial Low Pass dengan Gaussian, dan Filter Spasial High Pass dengan Operator Laplacian yang dapat diakses melalui https://github.com/wannn-one/02-pcv-spatial-filter

3.1 Konvolusi 2D

3.1.1 Menggunakan OpenCV

```python
import cv2
import numpy as np
import os
import time

# convolution 2d with opencv
if not os.path.exists('conv-2d'):
    os.makedirs('conv-2d')

kernel_number = 3 # ukuran kernel 3x3, bisa diubah sesuai kebutuhan

# read image
img = cv2.imread('images/a.jpg', cv2.IMREAD_GRAYSCALE)

start_time = time.time()

# create kernel
kernel = np.ones((kernel_number,kernel_number), np.float32)/kernel_number**2

# apply kernel
dst = cv2.filter2D(img, -1, kernel) # -1 means the depth of the output image is
    the same as the input image

time_taken = time.time() - start_time
print("%.4f seconds" % time_taken)

# show image
cv2.imshow('Original', img)
cv2.imshow('Convolution 2D', dst)

cv2.imwrite('conv-2d/conv2d_with_opencv.jpg', dst)
with open('conv-2d/time_taken_conv2d_with_opencv.txt', 'a') as f:
    f.write(str(time_taken) + '\n')

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.1.2 Operasi manual

```python
import cv2
import numpy as np
import time
import os

if not os.path.exists('conv-2d'):
    os.makedirs('conv-2d')

kernel_number = 3 # ukuran kernel 3x3, bisa diubah sesuai kebutuhan

def Convolusi(f,w):
    baris = f.shape[0]
```

```
    kolom = f.shape[1]
    g = np.zeros((baris,kolom))
    for y in range(baris):
        for x in range(kolom):
            g[y,x] = 0
            for i in range(kernel_number):
                yy = y+i-1
                if (yy < 0) or (yy>=baris-1):
                    continue
                for j in range(kernel_number):
                    xx = x+j - 1
                    if (xx<0) or (xx>=kolom-1):
                        continue
                    g[y,x]=g[y,x]+f[yy,xx]*w[i,j]
                #end for
            #end for
        #end for
    #end for
    g = np.uint8(np.floor(g))
    return g
# Membaca File Citra
f = cv2.imread("images/a.jpg",cv2.IMREAD_GRAYSCALE)

start_time = time.time()

w =np.ones((kernel_number,kernel_number))/kernel_number**2

#   |1/9 1/9 1/9 |
#w =|1/9 1/9 1/9 |
#   |1/9 1/9 1/9 |

g = Convolusi( f,w)

time_taken = time.time() - start_time
print("%.4f seconds" % time_taken)

cv2.imshow('Original', f)
cv2.imshow('Convolution 2D', g)

cv2.imwrite('conv-2d/conv2d_without_opencv.jpg', g)
with open('conv-2d/time_taken_conv2d_without_opencv.txt', 'a') as f:
    f.write(str(time_taken) + '\n')

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.2 Low Pass Gaussian

3.2.1 Menggunakan OpenCV

```
# gaussian blur with opencv

import cv2
import time
import numpy as np
import os

if not os.path.exists('lpf-filter'):
    os.makedirs('lpf-filter')

# Membaca File Citra
start_time = time.time()
```

```
13
14 f = cv2.imread("images/a.jpg",cv2.IMREAD_GRAYSCALE)
15 g3 = cv2.GaussianBlur(f,(3,3),0)
16
17 time_taken = time.time() - start_time
18 print("%.4f seconds" % time_taken)
19
20 cv2.imshow('Citra Asli', f)
21 cv2.imshow('Hasil Gaussian Kernel 3x3', g3)
22
23 cv2.imwrite('lpf-gaussian/lpf-gaussian_gaussian_with_opencv.jpg', g3)
24 with open('lpf-gaussian/time_taken_lpf-gaussian_gaussian_with_opencv.txt', 'a')
      as f:
25     f.write(str(time_taken) + '\n')
26
27 cv2.waitKey(0)
28 cv2.destroyAllWindows()
```

3.2.2 Operasi manual

```
1  import cv2
2  import time
3  import numpy as np
4  import os
5
6  if not os.path.exists('lpf-filter'):
7      os.makedirs('lpf-filter')
8
9  def Convolosi(f,w):
10     baris = f.shape[0]
11     kolom = f.shape[1]
12     dkernel = w.shape[0]
13     dkernel2= np.int32(np.floor(dkernel/2))
14
15     g =np.zeros((baris,kolom))
16     for y in range(baris):
17         for x in range(kolom):
18             g[y,x] = 0
19             for i in range(dkernel):
20                 yy =y+i-dkernel2
21                 if (yy<0)|(yy>=baris-1):
22                     continue
23                 for j in range(dkernel):
24                     xx =x+j - dkernel2
25                     if (xx<0)|(xx>=kolom-1):
26                         continue
27                     g[y,x]=g[y,x]+f[yy,xx]*w[i,j]
28
29     g= np.uint8(np.floor(g))
30     return g
31
32 # Membaca File Citra
33 start_time = time.time()
34
35 f = cv2.imread("images/a.jpg",cv2.IMREAD_GRAYSCALE)
36
37 w = np.array
      ([[0.3679,0.6065,0.3679],[0.6065,1.0000,0.6065],[0.3679,0.6065,0.3679]])
      /4.8976
38 g3 = Convolosi(f,w)
39
40 time_taken = time.time() - start_time
```

```
print("%.4f seconds" % time_taken)

cv2.imshow('Citra Asli', f)
cv2.imshow('Hasil Gaussian Kernel 3x3', g3)

cv2.imwrite('lpf-gaussian/lpf-gaussian_gaussian_without_opencv.jpg', g3)
with open('lpf-gaussian/time_taken_lpf-gaussian_gaussian_without_opencv.txt', 'a'
    ) as f:
    f.write(str(time_taken) + '\n')

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.3 High Pass Laplacian

3.3.1 Menggunakan OpenCV

```
# canny edge with opencv

import cv2
import time
import os

def use_laplace(img):
    laplacian = cv2.Laplacian(img, cv2.CV_64F)
    return cv2.convertScaleAbs(laplacian)

def use_sobel(img):
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)

    sobel = sobelx + sobely
    return cv2.convertScaleAbs(sobel)

def use_canny(img):
    return cv2.Canny(img, 50, 100)

if not os.path.exists('canny-edge'):
    os.makedirs('canny-edge')

img = cv2.imread('images/a.jpg', 0)

start_time_canny = time.time()

canny = use_canny(img)

time_taken_canny = time.time() - start_time_canny
print("%.4f seconds for Canny" % time_taken_canny)

start_time_laplacian = time.time()

laplacian = use_laplace(img)

time_taken_laplacian = time.time() - start_time_laplacian
print("%.4f seconds for Laplacian" % time_taken_laplacian)

start_time_sobel = time.time()

sobel = use_sobel(img)

time_taken_sobel = time.time() - start_time_sobel
print("%.4f seconds for Sobel" % time_taken_sobel)
```

```
46
47 cv2.imshow('Original', img)
48 cv2.imshow('Canny Edge Detection', canny)
49 cv2.imshow('Laplacian Edge Detection', laplacian)
50 cv2.imshow('Sobel Edge Detection', sobel)
51
52 cv2.imwrite('canny-edge/canny_with_opencv.jpg', canny)
53 cv2.imwrite('canny-edge/laplacian_with_opencv.jpg', laplacian)
54 cv2.imwrite('canny-edge/sobel_with_opencv.jpg', sobel)
55
56 with open('canny-edge/time_taken_canny_with_opencv.txt', 'a') as f:
57     f.write(str(time_taken_canny) + '\n')
58
59 with open('canny-edge/time_taken_laplacian_with_opencv.txt', 'a') as f:
60     f.write(str(time_taken_laplacian) + '\n')
61
62 with open('canny-edge/time_taken_sobel_with_opencv.txt', 'a') as f:
63     f.write(str(time_taken_sobel) + '\n')
64
65 cv2.waitKey(0)
66 cv2.destroyAllWindows()
```

3.3.2 Operasi manual

```
1  import cv2
2  import numpy as np
3  import time
4  import os
5
6  if not os.path.exists('canny-edge'):
7      os.makedirs('canny-edge')
8
9  def Convolosi(f,w):
10     f  =np.float64(f)/255
11     baris = f.shape[0]
12     kolom = f.shape[1]
13     dkernel = w.shape[0]
14     dkernel2= np.int32(np.floor(dkernel/2))
15     g =np.zeros((baris,kolom))
16     for y in range(baris):
17         for x in range(kolom):
18             g[y,x] = 0
19             for i in range(dkernel):
20                 yy =y+i-dkernel2
21                 if (yy<0)|(yy>=baris-1):
22                     continue
23                 for j in range(dkernel):
24                     xx =x+j - dkernel2
25                     if (xx<0)|(xx>=kolom-1):
26                         continue
27                     g[y,x]=g[y,x]+f[yy,xx]*w[i,j]
28
29     return g
30
31 # Membaca File Citra
32 f = cv2.imread("images/a.jpg",cv2.IMREAD_GRAYSCALE)
33
34 start_time = time.time()
35
36 w = np.array([[0,1,0],[1,-4.,1],[0,1,0]])
37 g = Convolosi(f,w)
38
```

```
time_taken = time.time() - start_time
print("%.4f seconds" % time_taken)

cv2.imshow('Citra Asli', f)
cv2.imshow('Highpass Filter', g)

cv2.imwrite('canny-edge/canny_edge_without_opencv.jpg', g)
with open('canny-edge/time_taken_canny_edge_without_opencv.txt', 'a') as f:
    f.write(str(time_taken) + '\n')

cv2.waitKey(0)
cv2.destroyAllWindows()
```

4 Perbandingan Hasil

Dalam bagian ini, kami akan membandingkan hasil dari penggunaan OpenCV dan implementasi manual (tanpa OpenCV) untuk Konvolusi 2D, LPF Gaussian, dan HPF Laplacian. Perbandingan ini akan memberikan pemahaman yang lebih baik tentang keunggulan dan kelemahan masing-masing metode dalam pemrosesan gambar.

4.1 Konvolusi 2D

Berikut merupakan perbandingan Konvolusi 2D menggunakan fungsi *built-in* OpenCV dan operasi manual

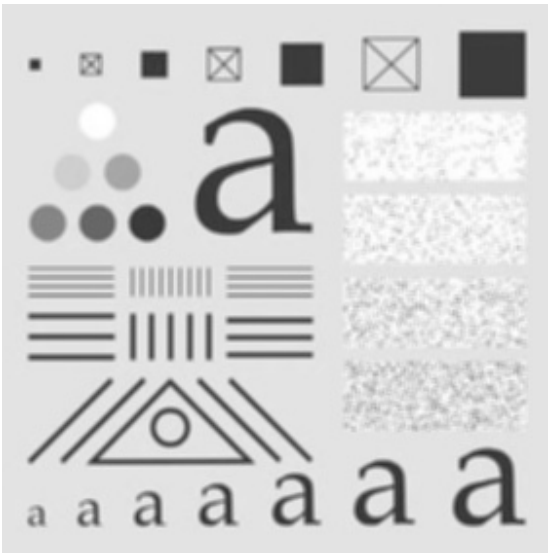

Figure 1: Konvolusi 2D dengan fungsi *built-in* OpenCV 3x3 Kernel

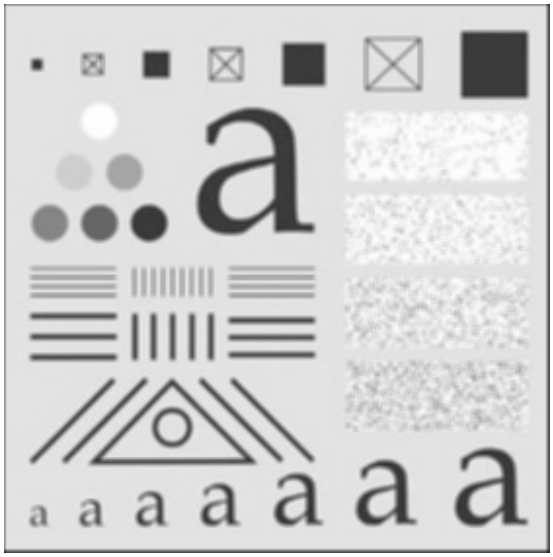

Figure 2: Konvolusi 2D dengan operasi manual OpenCV 3x3 Kernel

Hasil yang didapatkan dari kedua gambar memiliki kemiripan, namun dalam prosesnya memakan waktu yang berbeda, jika menggunakan fungsi *built-in* OpenCV, waktu yang dihabiskan oleh program akan relatif lebih lama, berikut data dari 10 kali run program Konvolusi 2D kernel 3 x 3 dengan fungsi *built-in* OpenCV dan operasi manual.

Table 1: Waktu fungsi *built-in* OpenCV

Run program ke-	Waktu (s)
1	0.0002
2	0.0002
3	0.0002
4	0.0001
5	0.0001
6	0.0001
7	0.0001
8	0.0001
9	0.0001
10	0.0001

Table 2: Waktu operasi manual

Run program ke-	Waktu (s)
1	0.3306
2	0.2964
3	0.2940
4	0.2975
5	0.2968
6	0.2976
7	0.2981
8	0.2985
9	0.2979
10	0.2989

4.2 Low Pass Gaussian

Berikut merupakan perbandingan Low Pass Gaussian menggunakan fungsi *built-in* OpenCV dan operasi manual

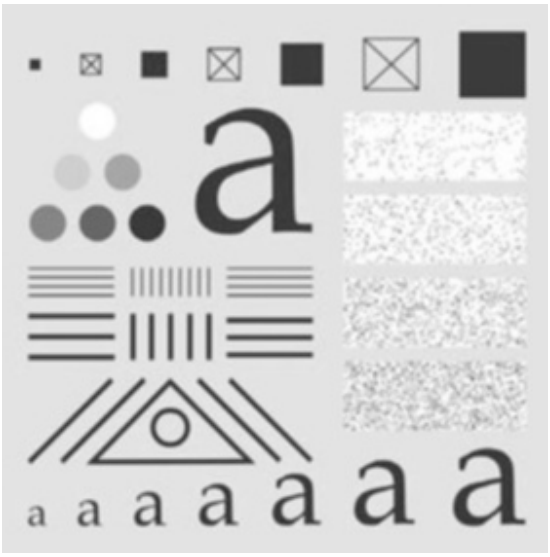

Figure 3: Low Pass Gaussian Kernel 3 x 3 dengan fungsi *built-in* OpenCV

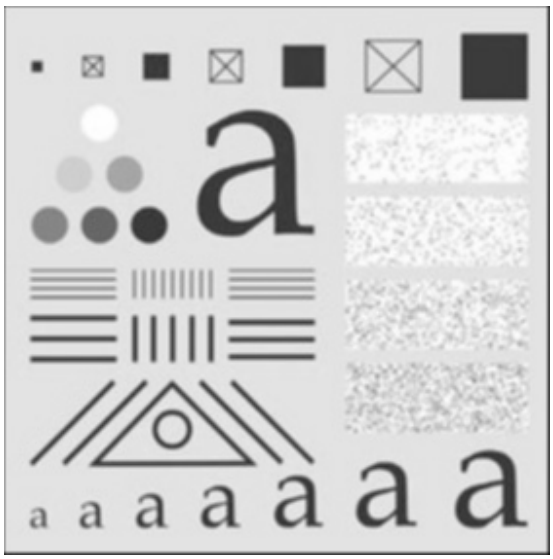

Figure 4: Low Pass Gaussian Kernel 3 x 3 dengan operasi manual

Hasil yang didapatkan dari kedua gambar memiliki kemiripan, namun dalam prosesnya memakan waktu yang berbeda, jika menggunakan fungsi *built-in* OpenCV, waktu yang dihabiskan oleh program akan relatif lebih lama juga, berikut data dari 10 kali run program Low Pass Gaussian Kernel dengan fungsi *built-in* OpenCV dan operasi manual.

Table 3: Waktu fungsi *built-in* OpenCV

Run program ke-	Waktu (s)
1	0.0012
2	0.0012
3	0.0012
4	0.0013
5	0.0012
6	0.0012
7	0.0012
8	0.0012
9	0.0012
10	0.0012

Table 4: Waktu operasi manual

Run program ke-	Waktu (s)
1	1.0518
2	1.0446
3	1.0325
4	1.0635
5	1.0132
6	1.0304
7	1.0184
8	0.9960
9	1.0195
10	1.0174

4.3 High Pass Laplacian

Berikut merupakan perbandingan High Pass Laplacian menggunakan fungsi *built-in* OpenCV dan operasi manual:

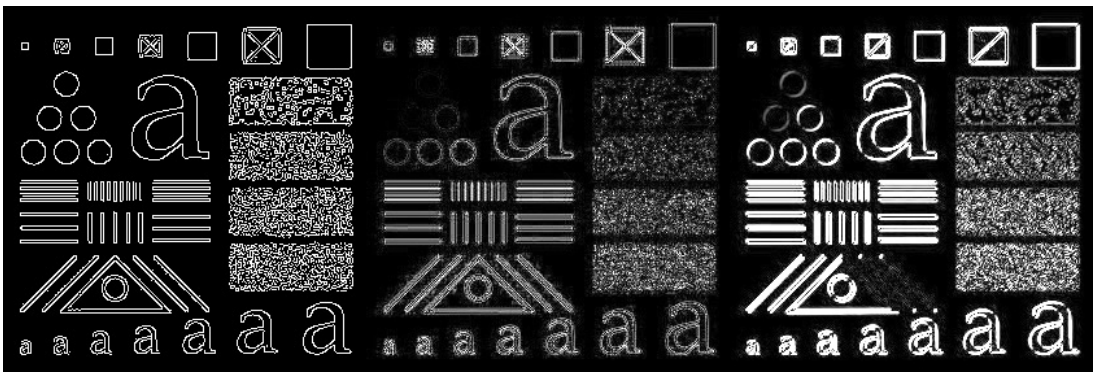

Figure 5: High Pass Laplacian dengan fungsi *built-in* OpenCV dengan Kernel 3 x 3, metode yang digunakan terdapat Canny Edge (Kiri), Laplacian (Tengah), dan Sobel (Kanan)

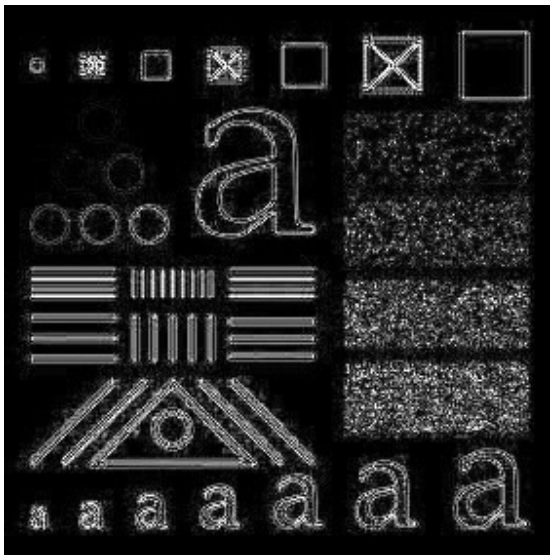

Figure 6: High Pass Laplacian dengan operasi manual dengan Kernel 3 x 3

Hasil yang didapatkan dari kedua gambar memiliki kemiripan, namun dalam prosesnya memakan waktu yang berbeda, jika menggunakan operasi manual, waktu yang dihabiskan oleh program akan relatif lebih lama. Hal ini bisa terjadi karena penggunaan perulangan for yang banyak untuk iterasi rumus, berikut data dari 10 kali run program High Pass Laplacian dengan Kernel 3 x 3 dengan fungsi *built-in* OpenCV dan operasi manual.

Table 5: Waktu operasi manual

Run program ke-	Waktu (s)
1	0.9216
2	0.9426
3	0.9320
4	0.9495
5	0.9770
6	0.9307
7	0.9225
8	0.9454
9	0.9233
10	0.9330

Table 6: Waktu operasi Canny

Run program ke-	Waktu (s)
1	0.0005
2	0.0005
3	0.0005
4	0.0005
5	0.0005
6	0.0005
7	0.0005
8	0.0005
9	0.0005
10	0.0005

Table 7: Waktu operasi Laplacian

Run program ke-	Waktu (s)
1	0.0010
2	0.0010
3	0.0010
4	0.0010
5	0.0010
6	0.0010
7	0.0010
8	0.0010
9	0.0010
10	0.0011

Table 8: Waktu Operasi Sobel

Run program ke-	Waktu (s)
1	0.0010
2	0.0010
3	0.0010
4	0.0010
5	0.0010
6	0.0010
7	0.0010
8	0.0010
9	0.0010
10	0.0011

Selanjutnya, akan dipaparkan kesimpulan dari hasil eksperimen ini pada bab kesimpulan.

5 Kesimpulan

Dari eksperimen yang telah kami lakukan, kami dapat menyimpulkan beberapa hal penting tentang penggunaan OpenCV dan operasi manual dalam melakukan Konvolusi 2D, Low Pass Gaussian, dan High Pass Laplacian.

- **Konvolusi 2D:** Kami menemukan bahwa penggunaan built-in OpenCV untuk mengatur Konvolusi 2D pada gambar lebih cepat daripada operasi manual yang ditunjukkan pada tabel 2. Fungsi cv2.filter2D() memungkinkan kami dengan mudah mengendalikan Konvolusi 2D dengan faktor tertentu, dan hasilnya diperoleh dengan waktu yang lebih singkat.
- **Low Pass Gaussian:** Kami juga menemukan bahwa penggunaan built-in OpenCV untuk mendapatkan Low Pass Gaussian atau *Gaussian Blur* lebih cepat dari operasi manual.
- **High Pass Laplacian:** Kami menemukan bahwa waktu operasi Canny adalah yang paling cepat di antara yang lain. Angka ini didapatkan dari pembulatan 4 angka di belakang koma, kemudian disusul oleh Sobel dan Laplacian yang sama sama kuat, kemudian baru operasi manual.
- **Ukuran File:** Meskipun ada perbedaan dalam waktu eksekusi antara metode di atas, ukuran file gambar tidak berubah. Penggunaan OpenCV atau operasi manual tidak memengaruhi ukuran file gambar.

Dengan demikian, pemilihan metode tergantung pada kebutuhan dan preferensi. Jika kita mengutamakan kecepatan, penggunaan built-in OpenCV mungkin menjadi pilihan yang baik untuk pengaturan kontras. Namun, jika kita memerlukan fleksibilitas dalam pengaturan kecerahan, operasi manual dapat memberikan kendali yang lebih besar. Keduanya memiliki manfaat dan kekurangan masing-masing, dan pemahaman yang baik tentang keduanya dapat membantu kita memilih metode yang sesuai.

6 Lampiran

Program dapat diakses di https://github.com/wannn-one/02-pcv-spatial-filter