

# **LAPORAN TEORI**

## **PENGOLAHAN CITRA**



**INTELLIGENT COMPUTING**

NAMA : Sabrina Herman

NIM : 202331241

KELAS : A

DOSEN : Dr. Dra. Dwina Kuswardani, M.Kom.

NO.PC : 32

ASISTEN : 1. Clarenca Sweetdiva Pereira

2. Viana Salsabila Fairuz Syahla

3. Kashrina Masyid Azka

4. Sasikirana Ramadhan Setiawan Putri

**INSTITUT TEKNOLOGI PLN**

**TEKNIK INFORMATIKA**

**2025**

## Laporan Teori

### 1. Jelaskan konsep operasi konvolusi !

Operasi konvolusi itu bisa dibilang sebagai cara menyiapkan suatu filter (yang disebut kernel) ke seluruh bagian citra untuk menghasilkan citra baru dengan efek tertentu, seperti memperjelas tepi, menghaluskan, atau mendeteksi bentuk tertentu.

Cara kerjanya kurang lebih begini:

- Kernel (sebuah matriks kecil) digeser ke setiap titik dalam citra.
- Di setiap titik, nilai pixel citra asli dikalikan dengan angka di kernel, lalu semua hasilnya dijumlahkan.
- Hasil penjumlahan ini menjadi nilai baru untuk pixel tersebut.

Contoh paling sederhana adalah pakai kernel ratarata, maka pixel baru akan jadi ratarata dari tetangga sekitarnya. Ini berguna buat menghaluskan gambar atau menghilangkan noise.

Jadi, konvolusi itu seperti alat bantu yang bisa kita atur sendiri polanya, tergantung tujuan kita ingin tajam, halus, atau deteksi tepi.

### 2. Jelaskan perbedaan mendasar antara filter ratarata, filter median, dan filter batas dalam operasi konvolusi!

Dalam pengolahan citra digital, filter seperti Mean Filter, Median Filter, dan Edge Detection Filter banyak digunakan untuk memproses citra sesuai dengan tujuan tertentu. Salah satu filter dasar yang sering digunakan adalah Mean Filter atau Filter RataRata. Sesuai dengan namanya, filter ini bekerja dengan cara merata-ratakan nilai pixel beserta tetangganya dalam area kernel tertentu. Fungsi utamanya adalah untuk menghaluskan citra serta mengurangi gangguan berupa noise jenis Gaussian. Meskipun efektif dalam hal tersebut, sayangnya penggunaan Mean Filter juga bisa menyebabkan hilangnya detail penting pada citra karena sifatnya yang cenderung “meratakan” semua perbedaan intensitas.

Berbeda dengan Mean Filter, Median Filter menggunakan pendekatan yang sedikit berbeda. Ia tidak menghitung ratarata melainkan mencari nilai tengah (median) dari sekumpulan nilai pixel di sekitar titik yang diproses. Teknik ini terbukti sangat efektif dalam menghilangkan jenis noise yang lebih kasar, seperti saltandpepper noise, tanpa mengaburkan detail objek secara signifikan. Oleh karena itu, Median Filter sering menjadi pilihan utama ketika citra mengandung noise impulsif. Namun, filter ini kurang optimal untuk mengatasi noise yang lebih halus seperti Gaussian noise.

Sementara itu, Edge Detection Filter atau Filter Deteksi Tepi memiliki tujuan yang berbeda dari kedua filter sebelumnya. Jika Mean Filter dan Median Filter lebih fokus pada penghalusan dan penghilangan noise, maka Edge Detection Filter justru mencari perubahan intensitas yang tiba-tiba untuk mendeteksi tepi objek. Filter ini menggunakan kernel khusus seperti pada operator Sobel, Prewitt, atau Canny. Dengan teknik ini, kita bisa menemukan kontur atau bentuk objek dalam citra yang berguna untuk analisis lebih lanjut seperti segmentasi atau pengenalan bentuk. Secara umum, dapat disimpulkan bahwa Mean Filter berfungsi untuk menghaluskan citra, Median Filter membantu menghilangkan noise kasar,

sedangkan Edge Detection Filter digunakan untuk mendeteksi batas atau tepi objek dalam citra.

3. Jelaskan cara kerja dan fungsi kernel!

Kernel itu semacam "template" atau "filter" kecil berupa matriks yang digunakan saat melakukan operasi konvolusi. Cara kerjanya yaitu kernel digeser ke seluruh citra, pixel per pixel. Kemudian di setiap posisi, isi kernel dikalikan satusatu dengan nilai pixel citra yang bersangkutan dan semua hasil perkalian ditambahkan, dan hasilnya jadi nilai baru untuk pixel di posisi itu.

Fungsi kernel:

- Bisa untuk menghaluskan citra (misalnya pakai kernel ratarata).
- Bisa juga untuk mendeteksi tepi (pakai kernel Sobel/Prewitt/Canny).
- Bahkan bisa buat mempertajam atau memperjelas detail tertentu.

Intinya, kernel itu alat yang bisa kita ubahubah sesuai kebutuhan untuk mengolah citra sesuai tujuan kita.

4. Apa perbedaan antara operator deteksi tepi Sobel, Pretwitt, dan Canny? Jelaskan keunggulan dan kelemahan masingmasing.

Dalam pengolahan citra digital, filter seperti Mean Filter, Median Filter, dan Edge Detection Filter banyak digunakan untuk memproses citra sesuai dengan tujuan tertentu. Salah satu filter dasar yang sering digunakan adalah Mean Filter atau Filter RataRata. Sesuai dengan namanya, filter ini bekerja dengan cara merata-ratakan nilai pixel beserta tetangganya dalam area kernel tertentu. Fungsi utamanya adalah untuk menghaluskan citra serta mengurangi gangguan berupa noise jenis Gaussian. Meskipun efektif dalam hal tersebut, sayangnya penggunaan Mean Filter juga bisa menyebabkan hilangnya detail penting pada citra karena sifatnya yang cenderung "meratakan" semua perbedaan intensitas.

Berbeda dengan Mean Filter, Median Filter menggunakan pendekatan yang sedikit berbeda. Ia tidak menghitung ratarata melainkan mencari nilai tengah (median) dari sekumpulan nilai pixel di sekitar titik yang diproses. Teknik ini terbukti sangat efektif dalam menghilangkan jenis noise yang lebih kasar, seperti saltandpepper noise, tanpa mengaburkan detail objek secara signifikan. Oleh karena itu, Median Filter sering menjadi pilihan utama ketika citra mengandung noise impulsif. Namun, filter ini kurang optimal untuk mengatasi noise yang lebih halus seperti Gaussian noise.

Sementara itu, Edge Detection Filter atau Filter Deteksi Tepi memiliki tujuan yang berbeda dari kedua filter sebelumnya. Jika Mean Filter dan Median Filter lebih fokus pada penghalusan dan penghilangan noise, maka Edge Detection Filter justru mencari perubahan intensitas yang tiba-tiba untuk mendeteksi tepi objek. Filter ini menggunakan kernel khusus seperti pada operator Sobel, Prewitt, atau Canny. Dengan teknik ini, kita bisa menemukan kontur atau bentuk objek dalam citra yang berguna untuk analisis lebih lanjut seperti

segmentasi atau pengenalan bentuk. Secara umum, dapat disimpulkan bahwa Mean Filter berfungsi untuk menghaluskan citra, Median Filter membantu menghilangkan noise kasar, sedangkan Edge Detection Filter digunakan untuk mendeteksi batas atau tepi objek dalam citra.

5. Apa perbedaan antara transformasi translasi, rotasi dan skala dalam transformasi geometric citra?

Dalam pengolahan citra, transformasi geometrik digunakan untuk memanipulasi struktur spasial dari suatu citra. Salah satu jenis transformasi tersebut adalah translasi , yaitu proses perpindahan posisi seluruh titik pixel pada citra sejauh nilai tertentu pada arah horizontal (sumbu x) dan/atau vertikal (sumbu y), tanpa mengubah orientasi atau ukuran citra. Contohnya, citra dapat digeser 20 pixel ke kanan dan 10 pixel ke atas.

Selanjutnya, rotasi adalah transformasi yang memutar citra terhadap sebuah titik pusat dengan sudut tertentu, baik searah maupun berlawanan dengan arah jarum jam. Misalnya, citra dapat diputar 90 derajat searah jarum jam untuk menyesuaikan orientasinya.

Lalu ada skala , yaitu operasi yang mengubah ukuran citra dengan cara memperbesar atau memperkecil secara proporsional. Sebagai contoh, citra bisa diperbesar dua kali lipat dari ukuran aslinya agar lebih jelas saat diproses lebih lanjut. Ketiga transformasi ini sering digunakan dalam tahap prapemrosesan citra, misalnya untuk meluruskan dokumen yang miring atau menyamakan ukuran objek dalam berbagai citra agar analisis menjadi lebih konsisten dan akurat.

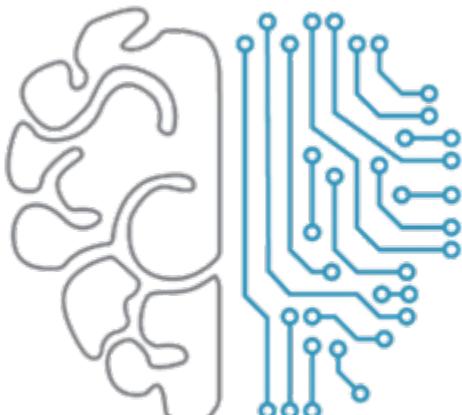
6. Apa yang dimaksud dengan transformasi affine dalam transformasi geometric?

Transformasi affine adalah jenis transformasi geometrik yang lebih kompleks karena mampu menggabungkan beberapa operasi sekaligus, seperti translasi (pergeseran), rotasi (perputaran), skala (pembesaran atau pengecilan), hingga shear atau distorsi bentuk. Intinya, dengan transformasi affine, kita bisa melakukan berbagai manipulasi pada citra secara bersamaan tanpa merusak struktur dasarnya salah satu ciri khas dari transformasi ini adalah bahwa garis lurus tetap lurus, meskipun posisi atau bentuknya berubah. Ini sangat penting dalam pengolahan citra agar objek tetap terlihat proporsional dan mudah dikenali setelah diproses.

Contoh penerapan transformasi affine yang sering dijumpai adalah dalam perbaikan dokumen hasil scan. Misalnya, ketika kita memindai sebuah dokumen dan hasilnya miring (tidak lurus), maka kita bisa menggunakan transformasi affine untuk meluruskannya proses ini biasa disebut deskew. Selain itu, transformasi affine juga digunakan untuk menyesuaikan bentuk objek dalam citra, seperti mengubah area persegi panjang yang tampak miring akibat sudut pengambilan gambar menjadi persegi sempurna. Contoh ini banyak diterapkan dalam aplikasi seperti OCR (Optical Character Recognition) atau deteksi plat nomor kendaraan. Secara umum, transformasi affine sangat membantu dalam membuat citra siap untuk dianalisis lebih lanjut dengan akurasi yang tinggi.

# **LAPORAN PRAKTIKUM**

## **PENGOLAHAN CITRA**



**INTELLIGENT COMPUTING**

NAMA : Sabrina Herman

NIM : 202331241

KELAS : A

DOSEN : Dr. Dra. Dwina Kuswardani, M.Kom.

NO.PC : 32

ASISTEN : 1. Clarenca Sweetdiva Pereira

2. Viana Salsabila Fairuz Syahla

3. Kashrina Masyid Azka

4. Sasikirana Ramadhanty Setiawan Putri

**INSTITUT TEKNOLOGI PLN**

**TEKNIK INFORMATIKA**

**2025**

1. Import Library

## Import Library

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
#202331241_Sabrina Herman
```

Mengimpor library cv2, yang digunakan untuk membaca dan memproses gambar, dan matplotlib.pyplot, yang digunakan untuk menampilkan gambar dan histogram. numpy, yang digunakan untuk operasi array numerik.

2. Membaca Data Gambar

## MEMBACA DATA GAMBAR

```
: img = cv2.imread('Capybara.jpg')
#202331241_Sabrina Herman

: img.shape
#202331241_Sabrina Herman

: (4000, 4000, 3)

: img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#202331241_Sabrina Herman
```

Membaca gambar dan mengetahui ukuran dari gambar

3. Convert BGR to Gray

```
[6]: img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#202331241_Sabrina Herman
```

OpenCV menggunakan BGR secara default; untuk tampilan yang lebih cocok digunakan untuk praktikum dikonversi ke Gray

## 4. Ketetanggaan

Pixel

**KETETANGGAAN PIKSEL**

```
[8]: copyCitra = img.copy().astype(float)

m1,n1 = copyCitra.shape
output1 = np.empty([m1,n1])

print("Shape copyCitra : ", copyCitra.shape)
print("Shape output1 : ", output1.shape)

print('m1 : ', m1)
print('n1 : ', n1)
print()
#202331241_Sabrina Herman
```

Shape copyCitra : (4000, 4000)  
 Shape output1 : (4000, 4000)  
 m1 : 4000  
 n1 : 4000

Membuat salinan citra asli (img) ke dalam variabel copyCitra dengan tipe data float agar siap untuk operasi matematika.

Mendapatkan ukuran citra (jumlah baris dan kolom) menggunakan .shape.

Membuat array kosong output1 dengan ukuran yang sama untuk menyimpan hasil pemrosesan nanti.

Menampilkan dimensi citra dan jumlah baris/kolomnya.

## 5. Membuat filter median

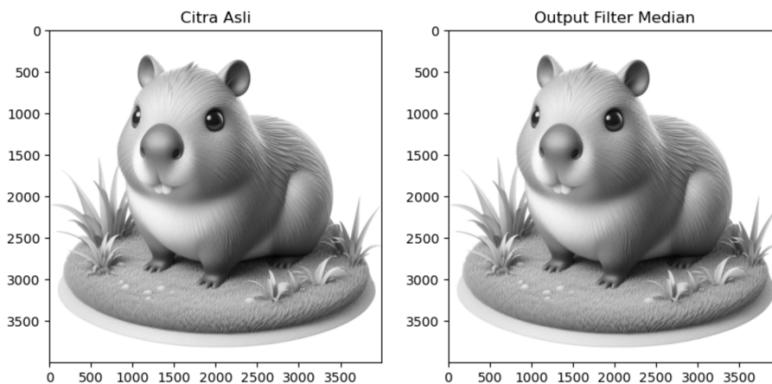
## MEMBUAT FILTER MEDIAN

```
[13]: img_median = img.copy()
img_median = cv2.cvtColor(img_median, cv2.COLOR_BGR2RGB)
img_median_after = cv2.medianBlur(img_median, 5)

fig, axs = plt.subplots(1,2, figsize =(10,10))
ax = axs.ravel()

ax[0].imshow(img, cmap='gray')
ax[0].set_title('Citra Asli')

ax[1].imshow(img_median_after, cmap='gray')
ax[1].set_title('Output Filter Median')
plt.show()
#202331241_Sabrina Herman
```



Codingan ini bertujuan untuk menerapkan filter median pada citra, yang berguna untuk menghilangkan noise (gangguan) seperti *saltandpepper noise* tanpa merusak detail penting

## Laporan 1

dalam citra. Hasilnya kemudian ditampilkan secara visual untuk membandingkan antara citra asli dan hasil pengolahan menggunakan filter median.

### 6. Membuat Filter RataRata

#### MEMBUAT FILTER RATA\_RATA

```
[10]: for baris in range (0, m1-1):
    for kolom in range(0, n1-1):
        a = baris
        b = kolom

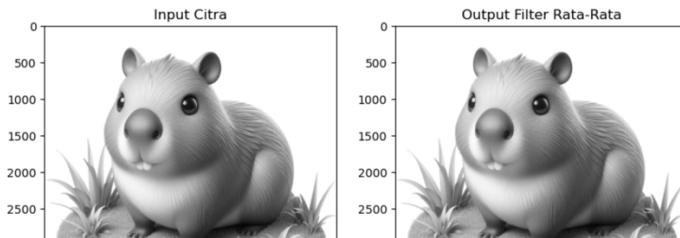
        jumlah = copyCitra[a-1,b-1] + copyCitra[a-1,b] + copyCitra[a-1,b+1] +\
                 copyCitra[a,b-1] + copyCitra[a,b] + copyCitra[a,b+1] +\
                 copyCitra[a+1,b-1] + copyCitra[a+1,b] + copyCitra[a+1, b+1]

        output1[a,b] = 1/9*jumlah
#202331241_Sabrina Herman
```

```
[11]: fig, axs = plt.subplots(1,2, figsize=(10,10))
ax = axs.ravel()

ax[0].imshow(img, cmap='gray')
ax[0].set_title('Input Citra')

ax[1].imshow(output1, cmap='gray')
ax[1].set_title('Output Filter Rata-Rata')
plt.show()
#202331241_Sabrina Herman
```



Menerapkan filter ratarata (mean filter) pada citra. Filter ratarata digunakan untuk menghaluskan citra dengan cara menggantikan nilai setiap pixel dengan ratarata nilai pixel di sekitarnya. Hasilnya kemudian ditampilkan secara visual untuk membandingkan antara citra asli dan hasil pengolahan menggunakan filter ratarata.

Menambahkan nilai kecerahan ke setiap piksel (sebanyak +40) untuk membuat citra menjadi lebih terang. Ini menyebabkan histogram bergerak ke kanan.

### 7. Membuat Filter Batas

#### MEMBUAT FILTER BATAS

```
citra1 = img.copy().astype(float)

for baris in range (0, m1-1):
    for kolom in range (0, n1-1):
        a = baris
        b = kolom

        arr = np.array ([citra1[a-1,b-1], citra1[a-1,b], citra1[a-1,b+1],
                        citra1[a,b-1], citra1[a,b], citra1[a,b+1],
                        citra1[a+1,b-1], citra1[a+1,b], citra1[a+1,b+1]])

        minPiksel = np.amin(arr)
        maxPiksel = np.amax(arr)

        if citra1[baris, kolom] < minPiksel:
            output1[baris, kolom] < minPiksel
        else :
            if citra1[baris, kolom] < maxPiksel:
                output1[baris, kolom] < maxPiksel
            else :
                output1[baris, kolom] = citra1[baris, kolom]

fig, axs = plt.subplots (1,2, figsize=(10,10))
ax = axs.ravel()

ax[0].imshow(citra1, cmap ='gray')
ax[0].set_title('Citra 1')

ax[1].imshow(output1, cmap='gray')
ax[1].set_title('Citra Hasil Filter Batas')
plt.show()
#202331241_Sabrina Herman
```

## Laporan 1

Menerapkan filter batas (thresholding) pada citra. Filter batas digunakan untuk memisahkan objek dalam citra berdasarkan nilai intensitas pixel, biasanya dengan cara mengganti pixel yang lebih kecil atau lebih besar dari suatu ambang batas tertentu.

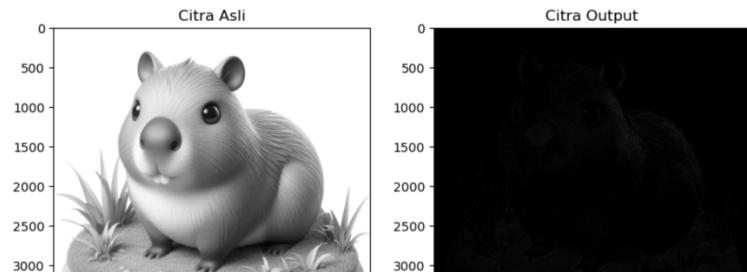
### 8. Konvolusi

menyiapkan kernel konvulsional

```
[20]: kernel = np.array ([[ -1, 0, -1],  
[ 0, 4, 0],  
[-1, 0, -1]])  
#202331241_Sabrina Herman
```

memproses citra dengan konvolusi menggunakan FILTER2D

```
[22]: outputcapy = cv2.filter2D(capy, -1, kernel)  
  
fig, axs = plt.subplots(1,2, figsize=(10,10))  
ax = axs.ravel()  
  
ax[0].imshow(capy, cmap="gray")  
ax[0].set_title("Citra Asli")  
  
ax[1].imshow(outputcapy, cmap="gray")  
ax[1].set_title("Citra Output")  
  
plt.show()  
#202331241_Sabrina Herman
```



Menerapkan operasi konvolusi pada citra menggunakan fungsi cv2.filter2D dari OpenCV. Konvolusi adalah salah satu teknik dasar dalam pengolahan citra yang digunakan untuk memproses citra dengan kernel (filter) tertentu.

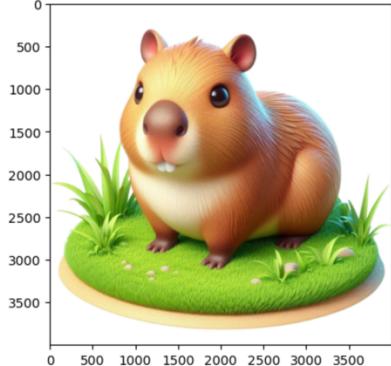
Hasil konvolusi menunjukkan bahwa beberapa bagian citra menjadi lebih gelap atau lebih terang, terutama di daerah tepi atau transisi intensitas yang tajam. Ini menunjukkan bahwa kernel berhasil mendeteksi tepi atau detail penting dalam citra.

### 9. Merubah citra ke RGB untuk persiapan penambahan noise

membaca dan menampilkan data Capybara.jpg

```
[24]: img = cv2.imread("Capybara.jpg")  
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
  
plt.imshow(img)  
plt.show  
#202331241_Sabrina Herman
```

```
[24]: <function matplotlib.pyplot.show(close=None, block=None)>
```



### 10. Import random noise dari library skimage

```
: from skimage.util import random_noise
#202331241_Sabrina Herman
```

## 11. Penambahan noise pada citra

```
[26]: noise_img_snp = random_noise(img, mode= "s&p", amount = 0.2)

#add gaussian noise to the image
noise_img_gaussian = random_noise(img, mode= "gaussian", mean=0, var=0.01)

#add speckle noise to noise the image
noise_img_speckle = random_noise(img, mode= "speckle")

noise_img_snp = np.array(255*noise_img_snp, dtype = 'uint8')
noise_img_gaussian = np.array(255*noise_img_gaussian, dtype = 'uint8')
noise_img_speckle = np.array(255*noise_img_speckle, dtype = 'uint8')
#202331241_Sabrina Herman
```

Codingan ini bertujuan untuk menambahkan berbagai jenis noise (gangguan) ke dalam citra menggunakan modul random\_noise dari library skimage.util. Noise yang ditambahkan meliputi:

1. Saltandpepper noise (s&p)
2. Gaussian noise
3. Speckle noise

Setelah menambahkan noise, nilai pixel diubah menjadi tipe data uint8 agar sesuai dengan format citra digital biasanya.

## 12. Menampilkan hasil penambahan noise pada citra

```
[27]: fig, axs = plt.subplots(4,1, figsize=(10,15))
ax = axs.ravel()

ax[0].imshow(img, cmap="gray")
ax[0].set_title("Citra Asli")

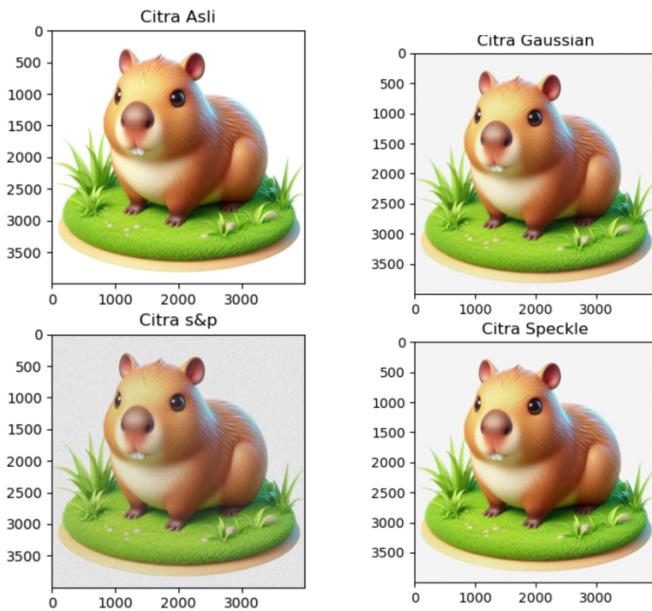
ax[1].imshow(noise_img_snp, cmap="gray")
ax[1].set_title("Citra s&p")

ax[2].imshow(noise_img_gaussian, cmap="gray")
ax[2].set_title("Citra Gaussian")

ax[3].imshow(noise_img_speckle, cmap="gray")
ax[3].set_title("Citra Speckle")

plt.show()
#202331241_Sabrina Herman
```

# Laporan 1



13. Membuat kernel 3x3 dengan nilai yang sama (1/9) , yang sering digunakan dalam operasi konvolusi untuk menghaluskan citra atau melakukan filtrasi ratarata.

**membuat kernel 3\*3 yang bernilai 1/9**

**image reduction**

```
[9]: kernel_3_3 = np.ones((3,3), np.float32)/9  
#202331241_Sabrina Herman
```

14. Melakukan Konvolusi

▼ melakukan konvolusi untuk mengatasi noise salt dan pepper dengan function filter

```
[31]: img_snp_avg_filter = cv2.filter2D(noise_img_snp, cv2.CV_8U, kernel_3_3, (-1,-1), delta=0,  
borderType = cv2.BORDER_DEFAULT)  
img_snp_median_filter = cv2.medianBlur(noise_img_snp, 3)  
#202331241_Sabrina Herman
```

**melakukan konvolusi untuk mengatasi noise gaussian dengan function filter**

```
[33]: img_gaussian_avg_filter = cv2.filter2D(noise_img_gaussian, cv2.CV_8U, kernel_3_3, (-1,-1), delta=0,  
borderType = cv2.BORDER_DEFAULT)  
img_gaussian_median_filter = cv2.medianBlur(noise_img_gaussian, 3)  
#202331241_Sabrina Herman
```

**melakukan konvolusi untuk mengatasi noise speckle dengan function filter**

```
[35]: img_speckle_avg_filter = cv2.filter2D(noise_img_speckle, cv2.CV_8U, kernel_3_3, (-1,-1), delta=0,  
borderType = cv2.BORDER_DEFAULT)  
img_speckle_median_filter = cv2.medianBlur(noise_img_speckle, 3)  
#202331241_Sabrina Herman
```

Mengatasi berbagai jenis noise (gangguan) pada citra , termasuk saltandpepper noise , Gaussian noise , dan speckle noise , menggunakan dua pendekatan utama: konvolusi dengan filter ratarata dan filter median . Pertama, untuk mengurangi saltandpepper noise , dilakukan konvolusi menggunakan fungsi cv2.filter2D dengan kernel 3x3 yang memiliki nilai ratarata (1/9), serta penggunaan cv2.medianBlur dengan ukuran kernel 3x3. Kedua, untuk mengatasi Gaussian noise dan speckle noise , langkah serupa diulang dengan menerapkan konvolusi menggunakan kernel ratarata dan filter median. Dalam setiap kasus, parameter seperti tipe data (CV\_8U) dan jenis penanganan batas (BORDER\_DEFAULT) ditentukan untuk memastikan hasil konvolusi sesuai dengan format citra asli.

# Laporan 1

## 15. Menampilkan hasil konvolusi

```
fig, axs = plt.subplots(3, 3, figsize=(15, 15)) # 3 baris x 3 kolom
ax = axs.ravel() # ubah jadi array 1 dimensi untuk akses mudah

# Baris 1: Salt & Pepper
ax[0].imshow(noise_img_snp, cmap="gray")
ax[0].set_title("Noise Salt & Pepper")

ax[1].imshow(img_snp_avg_filter, cmap="gray")
ax[1].set_title("S&P - Filter 2D (Average)")

ax[2].imshow(img_snp_median_filter, cmap="gray")
ax[2].set_title("S&P - Median Filter")

# Baris 2: Gaussian
ax[3].imshow(noise_img_gaussian, cmap="gray")
ax[3].set_title("Noise Gaussian")

ax[4].imshow(img_gaussian_avg_filter, cmap="gray")
ax[4].set_title("Gaussian - Filter 2D (Average)")

ax[5].imshow(img_gaussian_median_filter, cmap="gray")
ax[5].set_title("Gaussian - Median Filter")

# Baris 3: Speckle
ax[6].imshow(noise_img_speckle, cmap="gray")
ax[6].set_title("Noise Speckle")

ax[7].imshow(img_speckle_avg_filter, cmap="gray")
ax[7].set_title("Speckle - Filter 2D (Average)")

ax[8].imshow(img_speckle_median_filter, cmap="gray")
ax[8].set_title("Speckle - Median Filter")

plt.tight_layout()
plt.show()
#202331241_Sabrina Herman
```



# Laporan 1

## 16. Deteksi Tepi

```
[6]: cv.imshow("Gambar Parkir", image)
cv.waitKey(10)
cv.destroyAllWindows()
#202331241_Sabrina Herman

Mendeteksi Tepi Gambar

[8]: gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray, 75, 150)
#202331241_Sabrina Herman

[9]: cv.imshow("Gambar Parkir", gray)
cv.waitKey(10)
cv.destroyAllWindows()
#202331241_Sabrina Herman

[10]: fig, axes = plt.subplots(1,3, figsize = (10,10))
ax = axes.ravel()

axes[0].imshow(image,cmap = "gray")
axes[0].set_title("Parkir RGB")

axes[1].imshow(gray,cmap = "gray")
axes[1].set_title("Parkir Grayscale")

axes[2].imshow(edges,cmap = "gray")
axes[2].set_title("Parkir Canny")
#202331241_Sabrina Herman

[10]: Text(0.5, 1.0, 'Parkir Canny')
```



Codingan ini bertujuan untuk mengimpor, menampilkan, dan mendeteksi tepi pada citra menggunakan library OpenCV. Berikut adalah penjelasan langkah demi langkah:

### 1. Import Library

**import cv2 as cv**

**import numpy as np**

**import matplotlib.pyplot as plt**

cv2 (OpenCV)

Digunakan untuk pengolahan citra, seperti membaca gambar, konversi warna, dan deteksi tepi.

Numpy

Digunakan untuk manipulasi array, yang sering digunakan dalam pengolahan citra.

matplotlib.pyplot

Digunakan untuk visualisasi data, seperti menampilkan citra

### 2. Mengimpor Gambar

`image = cv.imread("parkir.jpg")`

`cv.imread(...)`

Fungsi OpenCV untuk membaca file citra ke dalam format array NumPy.

"parkir.jpg": Nama file citra yang akan dibaca.

Citra akan diubah menjadi array NumPy dengan tipe data uint8, dimana setiap pixel memiliki nilai antara 0 hingga 255.

### 3. Menampilkan Gambar

```
cv.imshow("Gambar Parkir", image)
cv.waitKey(10)
cv.destroyAllWindows()
```

cv.imshow(...): Menampilkan citra ke layar.

"Gambar Parkir": Judul jendela tampilan.

image: Citra yang akan ditampilkan.

cv.waitKey(10): Menunggu input selama 10 milidetik sebelum melanjutkan eksekusi kode. Nilai 10 berarti jendela tampilan akan ditutup setelah 10 milidetik.

cv.destroyAllWindows(): Menutup semua jendela tampilan setelah waktu tertentu

#### 4. Mendeteksi Tepi Gambar

Langkah 1: Konversi Citra ke Skala Abuabu

```
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
cv.cvtColor(...)
```

Mengonversi citra dari format BGR (yang digunakan oleh OpenCV) menjadi skala abuabu.

**Image**

Citra asli dalam format BGR.

**cv.COLOR\_BGR2GRAY**

Flag untuk mengonversi ke skala abuabu.

Langkah 2: Deteksi Tepi Menggunakan Algoritma Canny

```
edges = cv.Canny(gray, 75, 150)
```

**cv.Canny(...):** Algoritma Canny untuk mendeteksi tepi dalam citra.

gray: Citra skala abuabu yang akan diproses.

75: Threshold rendah (low threshold).

150: Threshold tinggi (high threshold). Pixel dengan intensitas lebih besar dari high threshold dianggap sebagai tepi definitif, sedangkan pixel dengan intensitas antara kedua threshold dapat dianggap sebagai tepi jika terhubung dengan tepi definitif.

Langkah 3: Menampilkan Hasil Deteksi Tepi

```
cv.imshow("Gambar Parkir", gray)
cv.waitKey(10)
cv.destroyAllWindows()
```

cv.imshow(...)

Menampilkan hasil deteksi tepi.

"Gambar Parkir"

Judul jendela tampilan.

Gray

Citra hasil deteksi tepi.  
cv.waitKey(10)  
Menunggu input selama 10 milidetik.  
cv.destroyAllWindows()  
Menutup semua jendela tampilan setelah waktu tertentu.

#### Deteksi Garis Lurus

```
[12]: lines = cv.HoughLinesP(edges, 1, np.pi/180, 30, maxLineGap = 250)
image_line = image.copy()
#202331241_Sabrina Herman
```

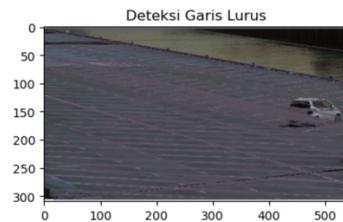
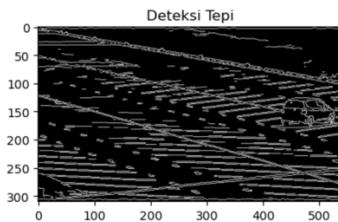
```
[13]: for line in lines:
    x1, y1, x2, y2 = line[0]
    cv.line(image_line, (x1,y1), (x2,y2), (90,78,95), 1)
#202331241_Sabrina Herman
```

```
[14]: fig, axs = plt.subplots(1,2, figsize = (10,10))
ax = axs.ravel()
```

```
axs[0].imshow(edges,cmap = "gray")
axs[0].set_title("Deteksi Tepi")
```

```
axs[1].imshow(image_line, cmap = "gray")
axs[1].set_title("Deteksi Garis Lurus")
#202331241_Sabrina Herman
```

```
[14]: Text(0.5, 1.0, 'Deteksi Garis Lurus')
```



Pertama, deteksi garis lurus dilakukan dengan menggunakan fungsi cv.HoughLinesP() dari OpenCV. Fungsi ini menerapkan Probabilistic Hough Transform , yang memungkinkan deteksi garis lurus pada citra tepi (edge image) hasil algoritma Canny. Parameter yang digunakan meliputi:

- edges: Citra tepi yang akan diproses.
- 1: Jarak resolusi rho (jarak antara garis dan titik asal koordinat polar), dengan nilai 1 piksel.
- np.pi/180: Resolusi theta (sudut polar), yang mengindikasikan bahwa sudut diproses dengan interval 1 derajat ( $\pi/180$ radian).
- 30: Threshold minimum untuk jumlah voting. Hanya garis dengan jumlah voting lebih besar atau sama dengan 30 yang akan dideteksi.
- maxLineGap=250: Batas maksimal jarak antar titik pada garis yang masih dianggap sebagai satu garis. Jika dua titik terdeteksi berada dalam jarak kurang dari 250, mereka dianggap bagian dari garis yang sama.

Hasil dari cv.HoughLinesP() adalah daftar garis yang terdeteksi, dimana setiap garis direpresentasikan oleh empat nilai:  $(x_1, y_1, x_2, y_2)$ , yang menyatakan koordinat dua titik ujung garis tersebut.

Selanjutnya, salinan citra dibuat menggunakan image.copy() agar perubahan tidak memengaruhi citra aslinya. Variabel image\_line digunakan untuk menandai garis-garis yang terdeteksi tanpa merusak citra asli.

Setelah itu, gambar garis terdeteksi pada citra menggunakan loop for line in lines. Setiap garis yang terdeteksi diambil dari lines, dan koordinat dua titik ujung garis (x1, y1 dan x2, y2) digunakan untuk menggambar garis menggunakan fungsi cv.line(). Garis-garis ini ditandai pada citra image\_line dengan warna abu-abu tua (BGR: (90, 78, 95)).

Terakhir, hasil ditampilkan secara visual menggunakan matplotlib. Dua subplot dibuat:

1. Subplot pertama menampilkan citra tepi (hasil deteksi tepi menggunakan algoritma Canny), dengan judul "Deteksi Tepi".
2. Subplot kedua menampilkan citra asli dengan garis terdeteksi , dengan judul "Deteksi Garis Lurus".

Hasil akhir adalah visualisasi yang menunjukkan garis-garis lurus yang berhasil ditemukan dalam citra, yang berguna dalam aplikasi seperti pengenalan pola geometris atau navigasi robot.

## 17. Geometrik

### Geometrik

```
[66]: import cv2 as cv
plat_asli = cv.imread('plat.jpg')
plat_rgb = cv.cvtColor(plat_asli, cv.COLOR_BGR2RGB)
plt.imshow(plat_rgb)
plt.show()
#202331241_Sabrina Herman
```



```
[68]: from skimage import io, transform
#202331241_Sabrina Herman
```

Membaca, mengonversi, dan menampilkan citra menggunakan library OpenCV. Berikut adalah penjelasan langkah demi langkah:

#### 1. Mengimpor Library

**import cv2 as cv**

cv2 (OpenCV) digunakan untuk membaca, memproses, dan menampilkan citra. Fungsi-fungsi dari OpenCV sangat berguna dalam pengolahan citra digital.

#### 2. Membaca Citra Asli

**plat\_asli = cv.imread('plat.jpg')**

cv.imread(...): Fungsi OpenCV untuk membaca file citra ke dalam format array NumPy.

- 'plat.jpg': Nama file citra yang akan dibaca.

- Citra akan diubah menjadi array NumPy dengan tipe data uint8, dimana setiap pixel memiliki nilai antara 0 hingga 255.
- Format warna default yang digunakan oleh OpenCV adalah BGR (Blue-Green-Red).

### 3. Mengonversi Warna dari BGR ke RGB

**plat\_rgb = cv.cvtColor(plat\_asli, cv.COLOR\_BGR2RGB)**

cv.cvtColor(...): Fungsi OpenCV untuk mengonversi format warna citra.

- plat\_asli: Citra asli dalam format BGR.
- cv.COLOR\_BGR2RGB: Flag untuk mengonversi citra dari format BGR menjadi RGB. Ini dilakukan karena format RGB lebih umum digunakan dalam banyak aplikasi visualisasi, seperti matplotlib.
- Hasil konversi disimpan dalam variabel plat\_rgb.

### 4. Menampilkan Citra

**plt.imshow(plat\_rgb)**

**plt.show()**

- plt.imshow(...): Fungsi matplotlib untuk menampilkan citra.
- plat\_rgb: Citra RGB yang akan ditampilkan.
- plt.show(): Menampilkan jendela visualisasi citra.

Setelah codingan dijalankan, citra plat nomor kendaraan (plat.jpg) akan ditampilkan dalam format RGB. Gambar yang ditampilkan menunjukkan beberapa motor dengan plat nomor terlihat jelas.

```
*[68]: from skimage import io, transform
#202331241_Sabrina Herman

[74]: src = np.array([
    [0,0],
    [0,50],
    [300,50],
    [300,0]
])

crop = np.array([
    [373,191],
    [346,257],
    [516,273],
    [542,195],
])

tform = transform.ProjectiveTransform()
tform.estimate(src,crop)

warped = transform.warp(plat_rgb, tform, output_shape=(50,300))

fig, axs = plt.subplots(1,2, figsize = (10,5))
ax = axs.ravel()

axs[0].imshow(warped)
axs[0].set_title("Gambar Img 1")

axs[1].imshow(plat_rgb)
axs[1].plot(crop[:,0], crop[:,1], '.r')
axs[1].set_title("Gambar Asli")

for a in axs:
    a.axis('Off')

plt.tight_layout()
plt.show
#202331241_Sabrina Herman
```

## Laporan 1

melakukan transformasi geometrik proyektif pada citra menggunakan library skimage. Langkah-langkahnya meliputi:

### 1. Mendefinisikan Titik Sumber dan Tujuan :

- o Dua array, src dan crop, dibuat untuk menyimpan koordinat titik-titik yang akan digunakan dalam transformasi proyektif.

- src: Merupakan koordinat titik-titik di citra asli yang akan dihubungkan dengan titik-titik tujuan.
- crop: Merupakan koordinat titik-titik di citra hasil yang ingin dicapai setelah transformasi.

### 2. Estimasi Transformasi Proyektif :

- o Fungsi `transform.ProjectiveTransform()` dari skimage digunakan untuk membuat objek transformasi proyektif.
- o Metode `estimate()` kemudian digunakan untuk menghitung parameter transformasi berdasarkan pasangan titik sumber (src) dan tujuan (crop).

### 3. Melakukan Transformasi Citra :

- o Fungsi `transform.warp()` digunakan untuk menerapkan transformasi proyektif ke citra `plat_rgb`.
- o Parameter `output_shape=(50, 300)` menentukan ukuran citra hasil transformasi.

### 4. Menampilkan Hasil :

- o Subplot dibuat untuk membandingkan citra hasil transformasi (warped) dengan citra asli (`plat_rgb`).
- o Pada subplot pertama, citra hasil transformasi ditampilkan dengan judul "Gambar Img 1".
- o Pada subplot kedua, citra asli ditampilkan dengan titik-titik koordinat crop ditandai menggunakan titik merah (dengan marker '!').
- o Axis pada kedua subplot dimatikan untuk meningkatkan visualisasi.

### 5. Penyesuaian Tampilan :

- o Fungsi `plt.tight_layout()` digunakan untuk mengatur layout agar tampilan lebih rapi.
- o Akhirnya, `plt.show()` dipanggil untuk menampilkan semua subplot.

Hasilnya seperti berikut

```
[74]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Laporan 1