

# **BAB I**

## **PENDAHULUAN**

### **Segmentasi**

Segmentasi dalam pengolahan citra digital adalah proses pemisahan atau pembagian citra menjadi beberapa bagian atau objek yang memiliki makna tertentu. Tujuan utama dari segmentasi adalah untuk menyederhanakan atau mengubah representasi citra agar lebih mudah dianalisis, sehingga fitur-fitur penting seperti objek, batas, atau pola dalam citra tersebut bisa diidentifikasi dengan lebih baik.

### **K-Means**

K-Means adalah algoritma klusterisasi yang digunakan untuk mengelompokkan data ke dalam beberapa klaster berdasarkan kemiripan antara elemen-elemen data tersebut. Dalam konteks pengolahan citra digital, K-Means sering digunakan untuk segmentasi citra, yaitu membagi citra menjadi beberapa bagian yang memiliki karakteristik serupa (misalnya intensitas, warna, atau tekstur).

### **Tepi Sobel**

Menggunakan dua kernel 3x3 (horizontal dan vertikal) untuk menghitung turunan pertama dari intensitas piksel dalam dua arah (x dan y). Dengan menghitung gradien intensitas dalam kedua arah, Sobel dapat mendeteksi tepi secara kasar.

### **Tepi Prewitt**

Mirip dengan Sobel, menggunakan kernel 3x3, tetapi bobot pada kernel Prewitt lebih sederhana, tanpa penguatan bobot pada elemen tengah. Prewitt juga menghitung gradien intensitas dalam dua arah (horizontal dan vertikal).

### **Tepi Canny**

Salah satu metode deteksi tepi yang paling populer dan kuat. Canny terdiri dari beberapa tahap, termasuk smoothing (penghalusan) dengan Gaussian filter, menghitung gradien, non-maximum suppression, dan thresholding ganda untuk mendeteksi tepi yang kuat dan menghubungkan tepi yang lemah.

### **Tepi Laplacian of Gaussian (LoG)**

LoG menggabungkan dua langkah: smoothing citra menggunakan Gaussian filter untuk mengurangi noise, dan kemudian menerapkan Laplacian (deteksi tepi) untuk mendeteksi perubahan intensitas.

### **Tepi Roberts Cross**

Menggunakan dua kernel 2x2 untuk menghitung gradien intensitas di sepanjang sumbu diagonal (bukan horizontal dan vertikal). Metode ini sederhana dan cepat.

## **Tepi Scharr**

Scharr adalah modifikasi dari Sobel, tetapi dengan kernel yang dioptimalkan untuk memberikan hasil yang lebih baik dalam mendeteksi tepi diagonal. Scharr bekerja dengan lebih baik untuk menangani noise dan perubahan intensitas yang kecil.

## **BAB II**

### **Tugas Khusus**

#### **Source Code**

##### **Segmentasi**

```
import cv2

import numpy as np

# Threshold untuk nilai HSV

threshold = 0.2

# Kernel untuk operasi morfologi

kernel5 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (20, 20))

# Inisialisasi koordinat dan nilai HSV

x_co = 0

y_co = 0

hsv = None

H = 0

S = 0

V = 0

# Threshold untuk H, S, dan V

thr_H = 180 * threshold

thr_S = 255 * threshold

thr_V = 255 * threshold

# Fungsi callback untuk mouse

def on_mouse(event, x, y, flag, param):

    global x_co, y_co, H, S, V, hsv

    if event == cv2.EVENT_LBUTTONDOWN:

        x_co = x

        y_co = y

        p_sel = hsv[y_co][x_co]

        H = p_sel[0]

        S = p_sel[1]

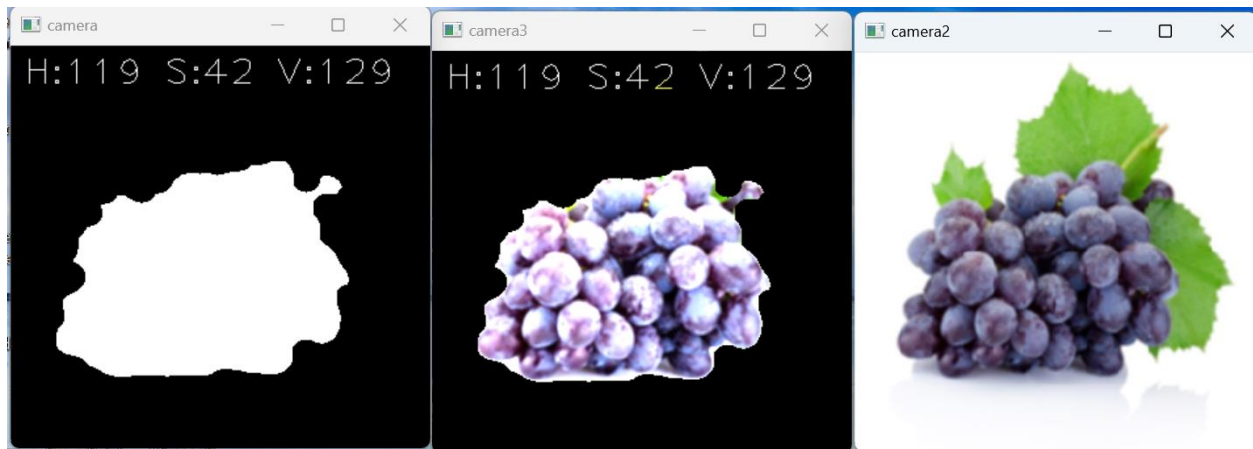
        V = p_sel[2]

# Membaca gambar
```

```

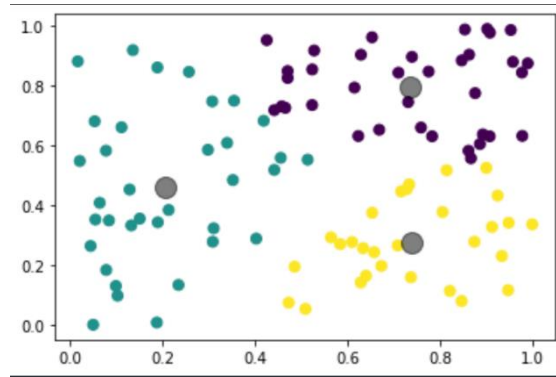
img = cv2.imread("C:/Users/Lenovo/Pictures/anggur.jpeg")
# Mengkonversi gambar dari BGR ke HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# Membuat jendela yang dinamai
cv2.namedWindow("kamera", 1)
cv2.namedWindow("kamera2", 2)
cv2.namedWindow("kamera3", 3)
# Mengatur callback mouse untuk jendela "kamera2"
cv2.setMouseCallback("kamera2", on_mouse, 0)
while True:
    # Memburamkan gambar
    src = cv2.blur(img, (3, 3))
    # Mendefinisikan rentang warna yang dipilih dalam HSV
    min_color = np.array([H - thr_H, S - thr_S, V - thr_V])
    max_color = np.array([H + thr_H, S + thr_S, V + thr_V])
    # Thresholding gambar HSV untuk mendapatkan hanya warna yang dipilih
    mask = cv2.inRange(hsv, min_color, max_color)
    # Menerapkan operasi morfologi closing
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel5)
    # Menambahkan teks pada gambar mask untuk menampilkan nilai H, S, V
    cv2.putText(mask, "H:" + str(H) + " S:" + str(S) + " V:" + str(V),
                (10, 30), cv2.FONT_HERSHEY_PLAIN, 2.0, (255, 255, 255),
                thickness=1)
    # Menampilkan gambar-gambar
    cv2.imshow("kamera", mask)
    cv2.imshow("kamera2", src)
    # Menambahkan mask segmen ke gambar asli
    src_segmented = cv2.add(src, src, mask=mask)
    cv2.imshow("kamera3", src_segmented)
    # Keluar dari loop jika tombol 'Esc' ditekan
    if cv2.waitKey(10) == 27:

```



## K-Means

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Menghasilkan data sampel
X = np.random.rand(100, 2)
# Membuat instance KMeans dengan 3 klaster
kmeans = KMeans(n_clusters=3, random_state=0)
# Melatih model dengan data
kmeans.fit(X)
# Mendapatkan label klaster
y_kmeans = kmeans.predict(X)
# Visualisasi klaster
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.show()
```



## Tepi Sobel

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Oct 21 20:57:53 2024
```

```
@author: Lenovo
```

```
"""
```

```
import cv2
```

```
import numpy as np
```

```
# Membaca gambar dari file
```

```
img = cv2.imread('C:/Users/Lenovo/Pictures/Camera  
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)
```

```
# Menggunakan operator Sobel untuk mendeteksi tepi
```

```
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3) # Gradien tepi pada arah  
X
```

```
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3) # Gradien tepi pada arah  
Y
```

```
# Menghitung magnitudo gradien
```

```
sobel_combined = cv2.magnitude(sobelx, sobely)
```

```
# Mengkonversi hasil ke format 8-bit untuk visualisasi
```

```
sobelx = cv2.convertScaleAbs(sobelx)
```

```
sobely = cv2.convertScaleAbs(sobely)
```

```
sobel_combined = cv2.convertScaleAbs(sobel_combined)
```

```
# Menampilkan gambar asli dan hasil deteksi tepi Sobel
```

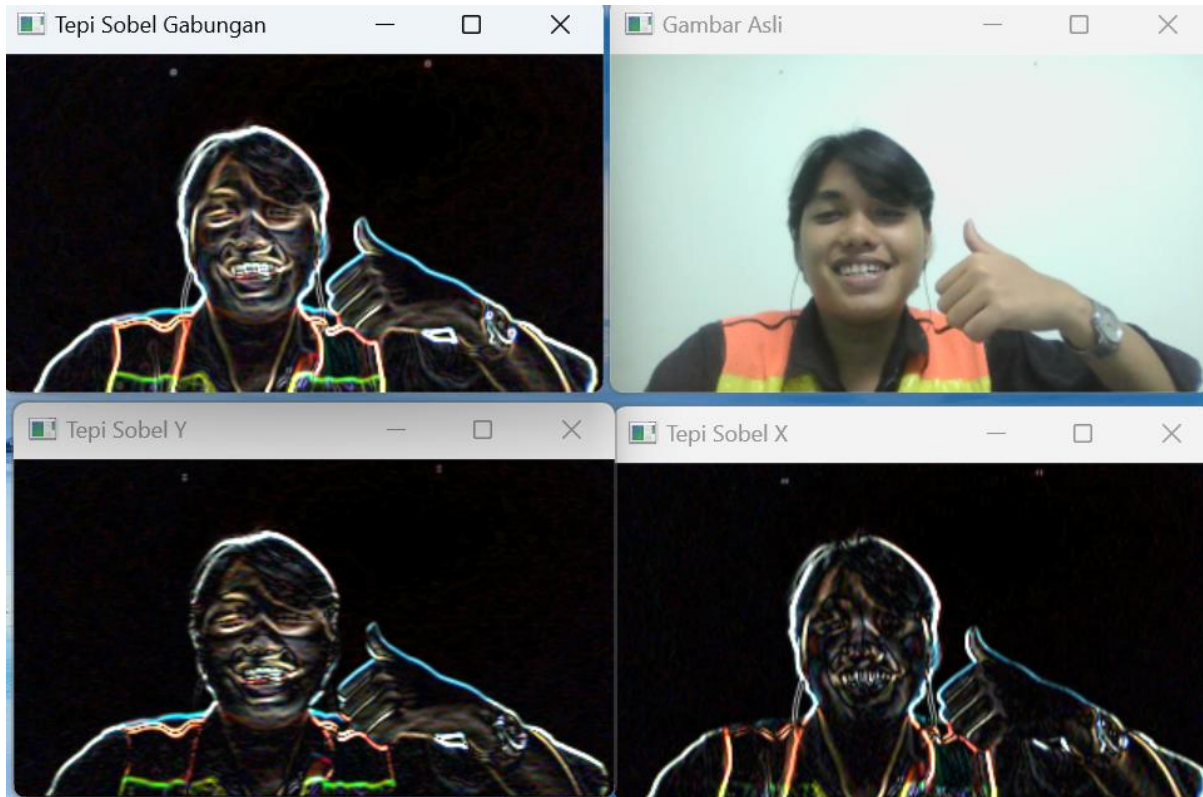
```
cv2.imshow('Gambar Asli', img)
```

```
cv2.imshow('Tepi Sobel X', sobelx)
```

```

cv2.imshow('Tepi Sobel Y', sobely)
cv2.imshow('Tepi Sobel Gabungan', sobel_combined)
# Menunggu hingga tombol ditekan dan menutup semua jendela
cv2.waitKey(0)
cv2.destroyAllWindows()

```



## Tepi Prewitt

```

# -*- coding: utf-8 -*-

```

```

"""

```

```

Created on Mon Oct 21 23:25:26 2024

```

```

@author: Lenovo

```

```

"""

```

```

import cv2

```

```

import numpy as np

```

```

# Membaca citra grayscale

```

```

img = cv2.imread('C:/Users/Lenovo/Pictures/Camera
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)

```

```

# Mendefinisikan kernel Prewitt
kernel_x = np.array([[ -1,  0,  1], [ -1,  0,  1], [ -1,  0,  1]])
kernel_y = np.array([[ -1, -1, -1], [ 0,  0,  0], [ 1,  1,  1]])

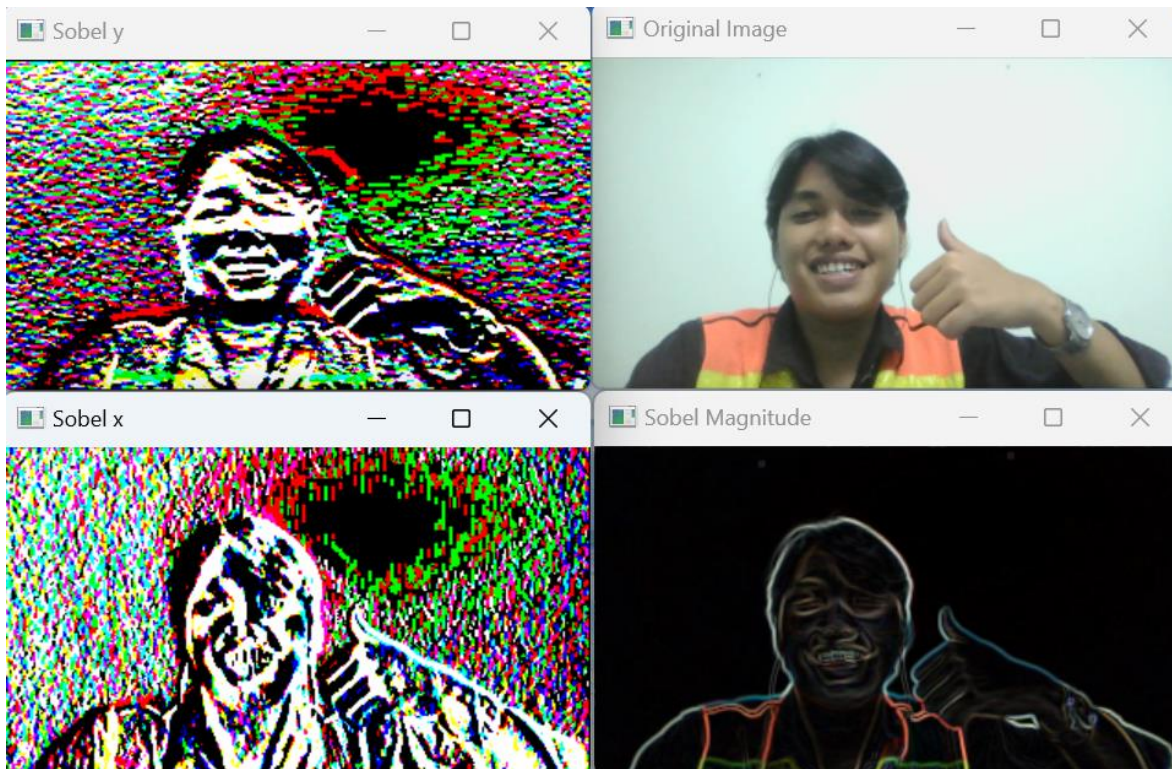
# Konvolusi dengan kernel
sobelx = cv2.filter2D(img, cv2.CV_64F, kernel_x)
sobely = cv2.filter2D(img, cv2.CV_64F, kernel_y)

# Menghitung magnitudo gradien
mag = np.sqrt(sobelx**2 + sobely**2)

# Normalisasi
mag = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX)
mag = mag.astype(np.uint8)

# Menampilkan hasil
cv2.imshow('Original Image', img)
cv2.imshow('Sobel x', sobelx)
cv2.imshow('Sobel y', sobely)
cv2.imshow('Sobel Magnitude', mag)
cv2.waitKey(0)
cv2.destroyAllWindows()

```





## Tepi Canny

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Oct 21 23:29:06 2024
```

```
@author: Lenovo
```

```
"""
```

```
import cv2
```

```
import numpy as np
```

```
# Membaca citra grayscale
```

```
img = cv2.imread('C:/Users/Lenovo/Pictures/Camera  
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)
```

```
# Deteksi tepi menggunakan Canny
```

```
canny = cv2.Canny(img, 100, 200)
```

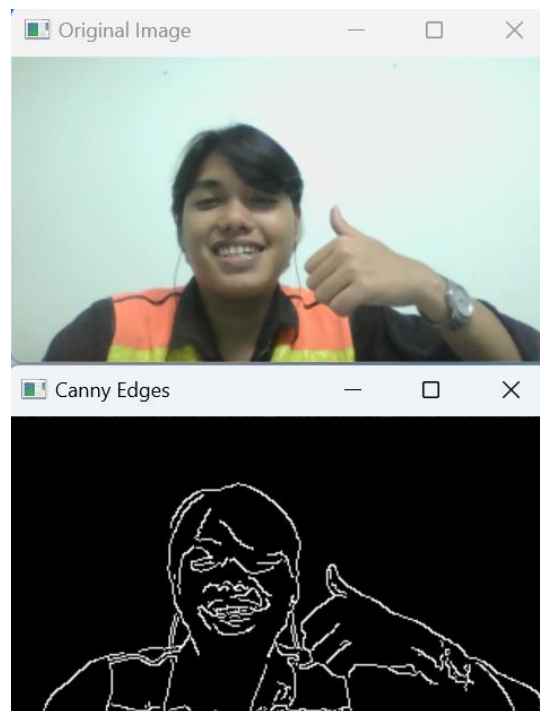
```
# Menampilkan hasil
```

```
cv2.imshow('Original Image', img)
```

```
cv2.imshow('Canny Edges', canny)
```

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

```
cv2.destroyAllWindows()
```



## Tepi Laplacian of Gaussian(LoG)

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Oct 21 23:34:18 2024
```

```
@author: Lenovo
```

```
"""
```

```
import cv2
```

```
import numpy as np
```

```
def LoG(img, sigma):
```

```
    # Membuat filter Gaussian
```

```
    size = 2*int(3*sigma) + 1
```

```
    x, y = np.meshgrid(np.arange(-size//2 + 1, size//2 + 1),  
                        np.arange(-size//2 + 1, size//2 + 1))
```

```
    gaussian = np.exp(-(x**2 + y**2) / (2.0 * sigma**2))
```

```
    # Membuat filter Laplacian
```

```
    laplacian = cv2.Laplacian(gaussian, cv2.CV_64F)
```

```
    # Konvolusi
```

```
    img_filtered = cv2.filter2D(img, cv2.CV_64F, laplacian)
```

```
    # Ambang batas (adjust sesuai kebutuhan)
```

```
    thresh = np.mean(img_filtered)
```

```
    img_binary = np.where(img_filtered > thresh, 255, 0).astype(np.uint8)
```

```
    return img_binary
```

```
# Membaca citra grayscale
```

```
img = cv2.imread('C:/Users/Lenovo/Pictures/Camera  
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)
```

```
# Menentukan nilai sigma
```

```
sigma = 2
```

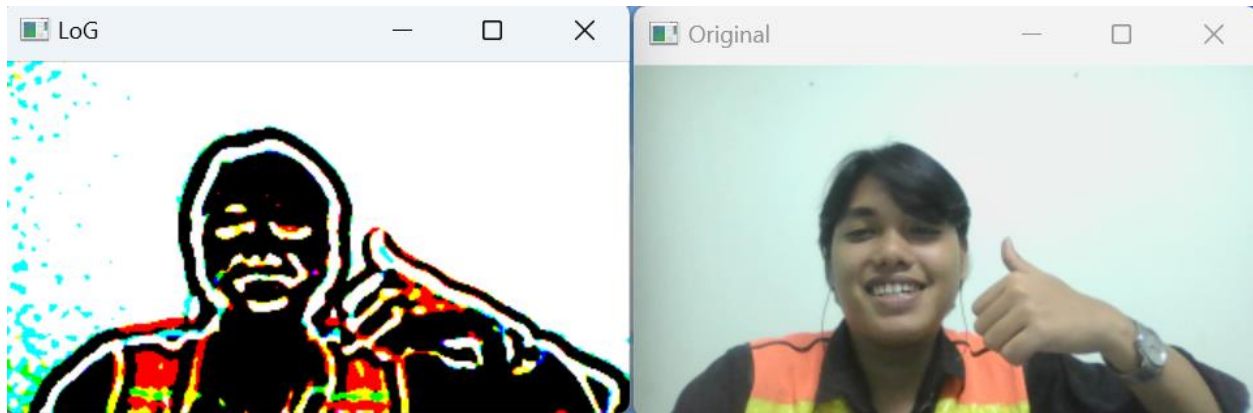
```
# Menerapkan filter LoG
```

```
result = LoG(img, sigma)
```

```
# Menampilkan hasil
```

```
cv2.imshow('Original', img)
```

```
cv2.imshow('LoG', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



## Tepi Robert Cross

```
import cv2

import numpy as np

# Membaca gambar dari file
img = cv2.imread('C:/Users/Lenovo/Pictures/Camera
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)

# Kernel untuk operator Roberts Cross
roberts_cross_x = np.array([[1, 0], [0, -1]], dtype=np.float32) # Kernel
untuk arah X
roberts_cross_y = np.array([[0, 1], [-1, 0]], dtype=np.float32) # Kernel
untuk arah Y

# Melakukan filter dengan kernel Roberts Cross
edge_x = cv2.filter2D(img, cv2.CV_64F, roberts_cross_x)
edge_y = cv2.filter2D(img, cv2.CV_64F, roberts_cross_y)

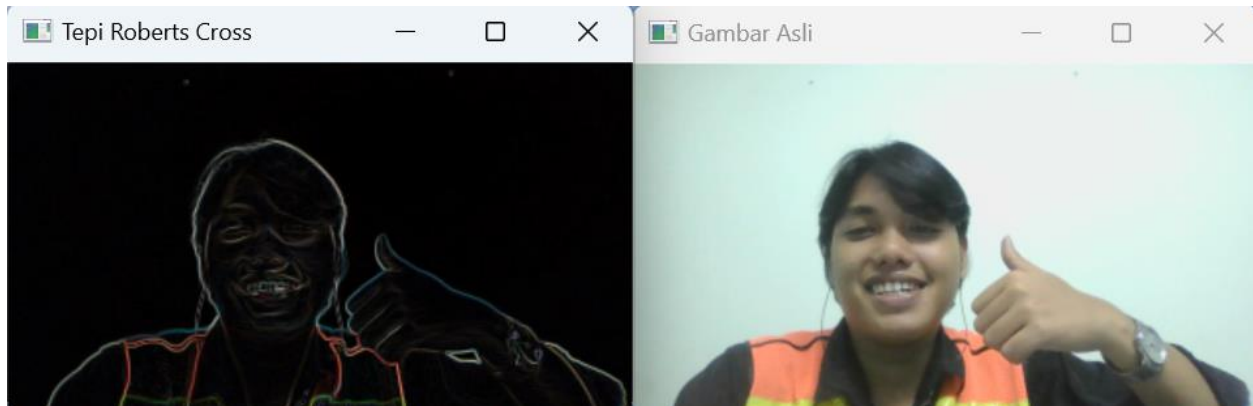
# Menghitung magnitudo gradien
edge_roberts = np.sqrt(np.square(edge_x) + np.square(edge_y))

# Mengonversi hasil ke format 8-bit untuk visualisasi
edge_roberts = cv2.convertScaleAbs(edge_roberts)

# Menampilkan gambar asli dan hasil deteksi tepi Roberts Cross
cv2.imshow('Gambar Asli', img)
cv2.imshow('Tepi Roberts Cross', edge_roberts)

# Menunggu hingga tombol ditekan dan menutup semua jendela
```

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



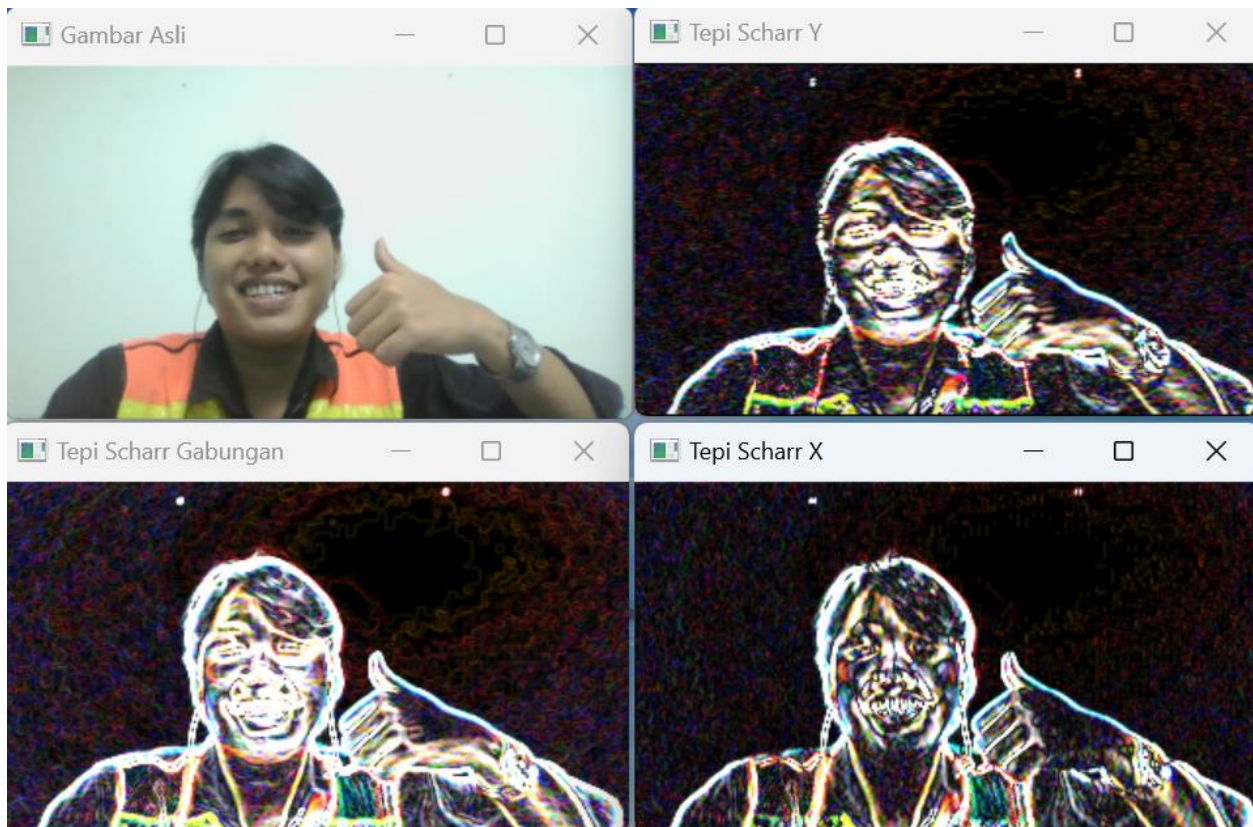
## Tepi Scharr

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 21 21:58:29 2024
@author: Lenovo
"""
import cv2
import numpy as np

# Membaca gambar dari file
img = cv2.imread('C:/Users/Lenovo/Pictures/Camera
Roll/WIN_20240625_08_15_09_Pro.jpg', 100)

# Menggunakan operator Scharr untuk mendeteksi tepi
scharr_x = cv2.Scharr(img, cv2.CV_64F, 1, 0) # Gradien pada arah X
scharr_y = cv2.Scharr(img, cv2.CV_64F, 0, 1) # Gradien pada arah Y
# Menghitung magnitudo gradien (menggabungkan X dan Y)
scharr_combined = cv2.magnitude(scharr_x, scharr_y)
# Mengonversi hasil ke format 8-bit untuk visualisasi
scharr_x = cv2.convertScaleAbs(scharr_x)
scharr_y = cv2.convertScaleAbs(scharr_y)
scharr_combined = cv2.convertScaleAbs(scharr_combined)
# Menampilkan gambar asli dan hasil deteksi tepi Scharr
cv2.imshow('Gambar Asli', img)
```

```
cv2.imshow('Tepi Scharr X', scharr_x)
cv2.imshow('Tepi Scharr Y', scharr_y)
cv2.imshow('Tepi Scharr Gabungan', scharr_combined)
# Menunggu hingga tombol ditekan dan menutup semua jendela
cv2.waitKey(0)
cv2.destroyAllWindows()
```



## **BAB III**

### **KESIMPULAN**

#### **Segmentasi**

Secara singkat, kode ini memungkinkan Anda memilih warna pada gambar secara interaktif dan memvisualisasikan bagian mana dari gambar yang sesuai dengan warna tersebut. Ini adalah cara yang rapi untuk melakukan segmentasi berbasis warna.

#### **K-Means**

Hasil akhirnya adalah sebuah scatter plot yang menunjukkan data sampel yang telah dikelompokkan ke dalam tiga kluster berbeda dengan pusat kluster yang ditampilkan. Ini memberikan visualisasi yang jelas tentang bagaimana KMeans mengelompokkan data ke dalam kluster berdasarkan kedekatan titik-titik data dalam dua dimensi.

#### **Tepi Sobel**

Intinya, kode ini memberikan cara yang efektif untuk mendeteksi dan memvisualisasikan tepi dalam gambar, yang merupakan langkah penting dalam berbagai aplikasi pemrosesan citra.

#### **Tepi Prewitt**

mendeteksi dan memvisualisasikan tepi dalam gambar menggunakan metode Prewitt, memberikan gambaran yang jelas tentang lokasi dan intensitas tepi dalam gambar.

#### **Tepi Canny**

Kode ini bertujuan untuk melakukan deteksi tepi pada gambar dengan menggunakan algoritma Canny Edge Detection. Hasil deteksi tepi ini menyoroti perbedaan intensitas antara piksel yang kuat, yang biasanya menunjukkan batas objek dalam gambar.

#### **Tepi Laplacian of Gaussian(LoG)**

Kode ini menerapkan filter Laplacian of Gaussian (LoG) untuk mendeteksi tepi dalam gambar. Dengan menggunakan filter Gaussian untuk smoothing dan filter Laplacian untuk mendeteksi tepi, metode ini memberikan hasil yang lebih halus dan lebih terfokus pada tepi dibandingkan metode deteksi tepi sederhana lainnya. Hasil akhir ditampilkan dalam bentuk gambar biner yang menunjukkan tepi yang terdeteksi.

#### **Tepi Robert Cross**

Kode ini menerapkan deteksi tepi menggunakan metode Roberts Cross, yang merupakan teknik untuk mendeteksi perubahan intensitas dalam gambar. Dengan menggunakan dua kernel untuk menghitung gradien horizontal dan vertikal, kode ini menghasilkan gambar yang menunjukkan tepi yang terdeteksi. Hasilnya ditampilkan berdampingan dengan gambar asli, memberikan visualisasi yang jelas tentang area dengan kontras tinggi dalam gambar.

## **Tepi Scharr**

Kode ini menerapkan deteksi tepi menggunakan operator Scharr, yang merupakan metode efektif untuk mendeteksi perubahan intensitas pada gambar. Dengan menghitung gradien dalam arah horizontal dan vertikal serta menggabungkannya, hasilnya memberikan informasi yang lebih detail tentang tepi yang ada dalam gambar. Hasil akhir ditampilkan di beberapa jendela, memungkinkan pengguna untuk menganalisis tepi secara mendetail dan membandingkannya dengan gambar asli.