PENGOLAHAN CITRA DIGITAL

Disusun Oleh :

Baralvi Abdullah Putra (226201042)

POLITEKNIK NEGERI SAMARINDA

2024

BAB I
PENDAHULUAN

1.1. Segmentasi

Segmentasi citra pada pengolahan citra digital merupakan proses menyaring dan mengambil daerah yang menarik untuk analisa suatu citra. Segmentasi citra juga dapat didefinisikan sebagai pembagian dan sub pembagian dari sebuah citra menjadi beberapa struktur dan sub struktur.

Untuk membuat struktur atau daerah tersebut bermakna dan berguna untuk analisa citra dan interpretasi, daerah-daerah ini harus direlasikan lebih kuat ke objek yang diperlihatkan atau fitur yang menarik dari citra tersebut. Segmentasi bermakna merupakan langkah pertama dari pengolahan citra tingkat rendah mengubah citra abu-abu atau berwarna ke satu citra atau lebih menjadi deskripsi citra tingkat tinggi dalam hal fitur, objek, dan skena. Keberhasilan dari analisa citra bergantung pada keandalan segmentasi, tetapi sebuah pembagian yang akurat dari sebuah citra merupakan masalah yang lebih menantang pada umumnya.

Ada beberapa jenis segmentasi, diantaranya :

1.2.1 Threshold Bases Segmentation

Metode ini merupakan salah satu metode paling sederhana dari segmentasi citra. *Threshold based Segmentation* dapat diterapkan langsung ke citra atau juga dapat digabungkan dengan teknik *preprocessing* dan *post processing*. Metode ini dapat digunakan juga untuk mengubah citra menjadi hitam putih atau monokrom yang dapat digunakan untuk membuat citra biner. Jenis segmentasi ini, piksel-piksel diberikan kategori berdasarkan rentang nilai dimana letak piksel berada. Misalnya sebuah citra disegmentasikan dengan piksel kurang dari 128 dapat ditempatkan ke satu kategori dan selain itu dapat ditempatkan ke kategori berbeda. Batasan lebih jauh diantara dua piksel berdekatan dapat ditimpa didalam citra putih atau original. Citra baru ini dapat dikatakan citra *threshold-segmented* dari dua tipe *predominant*.

1.2.1.1 Global Thresholding

Jenis *threshold* ini hanya dapat digunakan ketika distribusi intensitas dari sebuah objek cukup berbeda dari piksel latar belakang. Dalam kasus ini, sebuah nilai *threshold* dapat didefinisikan. Ketika T hanya bergantung pada f(a, b) dan T ini hanya berkaitan dengan karakter dari piksel. Ada beberapa teknik *global thresholding* seperti Otsu, *Optimal Thresholding*, *histogram analysis, iterative thresholding, clustering, multispectral,* dan *multi thresholding*.

1.2.1.2 Local Thresholding

Jenis *threshold* ini hanya dapat digunakan ketika satu nilai atau *threshold* tunggal tidak dapat digunakan dikarenakan iluminasi tidak merata yang mungkin disebabkan oleh bayangan atau arah. Tujuan utama dari metode ini untuk *sub divide* citra utama menjadi citra

nxn lalu memilih *threshold*. Jika *threshold* bergantung pada kedua f(a, b) dan p(a, b), maka jenis *thresholding* disebut dengan *local thresholding*. Didalam jenis ini, tingkat penyaringan abu-abu harus mengeleminasi tingkat abu-abu yang terputus pada sub citra. Beberapa contoh dari teknik *local thresholding* yaitu statistik *thresholding* sederhana, *2D entropy based thresholding*, dan *thresholding* transformasi histogram.

1.2.1.3 Adaptive Thresholding

Teknik ini mengambil warna citra sebagai input dan memberikan hasil citra biner. Pada teknik ini, nilai *threshold* dari setiap piksel dihitung, jika nilai kurang dari nilai *threshold* yang diberikan, maka nilai ini diambil sebagai latar belakang atau *background*, jika tidak nilai tersbut diambil sebagaai latar depan atau *foreground. Adaptive thresholding* digunakan untuk memisahkan objek citra dari *background* menggunakan intensitas piksel dari objek-ojbke citra yang beragam. Kekurangan utama dari teknik ini mahal secara komputasi dan tidak banyak digunakan pada penerapan dunia nyata.

1.2.2 Edge Based Segmentation

Edge atau tepi merujuk ke batas objek. Oleh karena itu, tepi ini merupakan permasalahan dasar yang sangat penting pada pengolahan citra. *Edge detection* membantu dalam mengurangi sejumlah data besar yang tidak diinginkan dan hanya menyaring informasi penting, sehingga struktur properti yang penting dari citra dapat dipertahankan.

1.2.3 Region Based Segmentation

Segmentasi ini merupakan kebalikan dari *edge based segmentation*. Segmentasi ini dimulai dari tengah dan perlahan tumbuh menuju batas or *boundary*. Pada teknik ini, piksel yang berkaitan dengan objek dikelompokkan untuk segmentasi lebih lanjut. Sangatlah penting ketika area yang terdeteksi untuk segmentasi harus ditutup. Teknik ini juga dikenal sebagai *similarity-based segmentation*. Pada umumnya, batas-batas diidentifikasikan untuk segmentasi. Dalam setiap langkah, setidaknya satu piksel berkaitan dengan area yang harus dipertimbangkan. Selanjutnya, setelah mengidentifikasi perubahan warna dan tekstur, aliran tepi dikonversi menjadi vektor. Tepi yang diambil digunakan untuk segmentasi lanjut.

Pada *region-based segmentation*, citra yang mirip disegmen ke beragam daerah. Teknik ini juga digunakan untuk menentukan daerah ssecara langsung. Menggunakan nilai abu-abu dari piksel citra sudah dapat melakukan partisi. Ada dua jenis dasar dari teknik ini yaitu metode *Region Growing* dan *Splitting and Merging*.

1.2.4 Morphological Method

Dalam pengolahan citra, morfologi domain merupakan sebuah istilah yang diberikan untuk identifikasi dan analisa struktur inheren secara geometris didalam sebuah objek. Metode ini memeriksa dan memverifikasi citra dengan templat kecil yang sudah didefinisikan sebelumnya yang dikenal sebagai *structuring element*, yang dapat diterapkan ke semua lokasi yang memungkinkan dari input citra dan memberikan hasil dengan ukuran yang sama. Sebuah citra biner baru dihasilkan dengan operasi ini yang dimana jika pengujian

berhasil, citra ini akan memiliki nilai piksel *non-zero* pada lokasi tersebut didalam sebuah input citra. Ada beragam kategori dari *structuring element* seperti *diamond shaped, cross shaped*, dan *square shaped*.

Dilasi, erosi, *opening,* dan *closing* merupakan inti dari operasi morfologi. Operasi morfolofi dapat menjadi inti untuk membangun *morphology filtering* yang lebih cocok untuk analisa bentuk dibandingkan dengan *linera filter* yang standar. Jenis lain dari teknik berbasis morfologi adalah *morphology watershed* yang bukan di bentuk dari operasi morfologi primitif. H Disable dan C Lantujuel memberikan konsep awal dari tool *watershed*. Sebagian besar dari *watershed* menggambarkan batasan antara penangkapan berdekatan. Jika tingkat abu-abu pada setiap titik dari sebuah ketinggian adalah gabungan, *watershed* dapat didefinisikan sebagai jembatan diantara dua batasan *watershed*.

1.2.5 Model Based Segmentation

Segmentasi ini juga dikenal sebagai *Markov Random Field* (MRF). Pada teknik ini kondisi daerah bawaan digunakan untuk segmentasi warna. Berbagai komponen dari warna piksel dapat dipertimbangkan sebagai variabel independen yang dapat digunakan untuk pengolahan lebih jauh. Teknik ini dapat mendeteksi dan menepi lebih jelas dan tepat ketika digabungan dengan *edge detection*.

1.2.6 Clustering Technique

Teknik *clustering* juga dapat digunakan untuk segmentasi citra. Beberapa algoritma dasar seperti *K-means* kebanyakan digunakan untuk citra bertekstur. *K-means* mengelompokkan piksel yang berkaitan untuk segmentasi citra. Pada kasus ini, segmentasi sepenuhnya begantung pada ciri-ciri dari citra.

Teknik *Fuzzy clustering* merupakan salah satu metode *clustering* kebanyakan diugnakan untuk citra warna. Pada metode ini, teknik *clustering* digunakan sehingga memberikan *cluster* warna menggunakan fungsi keanggotaan *fuzzy* dalam ruang warna terhadap ruang citra. Teknik ini sangat berguna dalam mengenali daerah berwarna. Setelah mendapatkan citra kasar disegmentasi menggunakan *multithresholding*, citra ini disempurnakan lebih lanjut oleh *fuzzy c-mean clustering*, teknik ini dapat diterapkan ke jenis dari citra *multispectral* apapun.

Terkadang metode *clustering* mencoba mengelompokkan pola yang terlihat sama secara bersamaan, sehingga metode ini dapat dikatakan sebagai salah satu *tool* terkuat dari segmentasi citra. Analisa lebih lanjut dilakukan untuk *subdivide* dataset citra menjadi sejumlah *cluster*. *K-means clustering* merupakan salah satu metode populer karena kesederhanaannya dan efisiensi dalam komputasi.

Clustering juga merupakan algoritma terpopuler dalam segmentasi citra. Algoritma ini mengklasifikasi data dan pola-pola dalam kategori. Dua contoh metode *clustering* yang sering digunakan yaitu *K-means* dan *fuzzy c-means*. Kedua metode mampu menghasilkan

pembagian citra dibawah kondisi tertentu termasuk angka *cluster* dan untuk memulihkan algoritma *cluster*, angka dan pusat *cluster* bergantung pada grafik keputusan.

1.2. K-Means Clustering

K-means clustering memisahkan sejumlah data menjadi kelompok *k-number* dari data. Algoritma ini mengklasifikasikan data set yang diberikan menjadi *k-number* dari *cluster* yang terputus. *K-means clustering* terdiri dari dua fase terpisah. Fase pertama menghitung sentroid k dan fase kedua algoritma ini mengambil setiap titik ke *cluster* yang mempunyai sentroid terdekat terhadap titik data.

Ada beberapa metode berbeda untuk menentukan jarak dari sentroid terdekat dan salah satu metode yang sering digunakan yaitu *Euclidean distance*. Setelah pengelompokkan selesai, algoritma ini menghitung ulang sentroid baru dari setiap *cluster* dan berdasarkan dari sentroid tersebut, jarak Euclidean baru dihitung diantara setiap pusat dan setiap titik data dan menetapkan titik-titik kedalam *cluster* yang mempunyai jarak Euclidean minimum. Setiap *cluster* didalam partisi didefinisikan oleh anggota objek dan oleh sentroidnya. Sentroid untuk setiap *cluster* merupakan titik ke dimana jumlah jarak dari semua objek ke *cluster* tersebut diminimalisir. Jadi, *K-means* merupakan sebuah algoritma iteratif yagn dimana algoritma tersebut meminimalisirkan jumlah jarak dari setiap objek ke sentroid *cluster*, pada semua *cluster*.

1.3. Garis Tepi

1.3.1 Canny Edge Detection

Canny edge detetction operator merupakan algoritma *multi-stage* yang mencari berbagai macam tepi pada citra. Metode ini digunakan untuk menentukan tepi-tepi citra pada berbagai arah seperti horisontal, vertikal, dan tepi diagonal pada citra buram. Metode ini terdiri dari empat *filter* untuk mendeteksi batas dari sebuah citra. Operator deteksi tepi mengembalikan sebuah nilai untuk derivatif pertama pada arah horisontal (Gx) dan arah vertical (Gy). Dari gradien dan arah tepi ini dapat dituliskan kedalam bentuk model matematikanya :

$$G = \sqrt{G_x^2 + G_y^2} \quad (1)$$

$$Sudut(\theta) = tan^{-1}(\frac{G_y}{G_x}) \quad (1)$$

1.3.2 Sobel Edge Detection

Sobel edge detection diturunkan menjadi dua 3x3 kernel. Kernel-kernel ini dikonvolusikan dengan citra awal untuk menghitung perkiraan derivatif, satu untuk horisontal dan satu untuk vertikal. Misal A adalah sumber citra, Gx dan Gy merupakan dua citra pada setiap titik berisikan derivatif perkiraan terhadap horisontal dan vertikal.

$$G_x = [P_7 + CP_8 + P_9] - [P_1 + CP_2 + P_3] \quad (2)$$

$$G_y = [P_3 + CP_6 + P_9] - [P_1 + CP_4 + P_7] \quad (2)$$

Dengan nilai konstanta jepi c = 2. Dengan ini, persamaan ini dapat digambarkan sebagai berikut.

$$Gx = \{[(x+1, y-1) + 2(x+1, y) + f(x+1, y+1\,[(x-1, y-1) + 2(x-1, y) + f(x-1, y+1)]\} \quad (3)$$

$$Gy = [(x-1, y+1) + 2(x, y+1) + f(x+1, y)] - [f(x-1, y-1) + 2f(x, y-1) + f(x+1, y-1)] \quad (4)$$

Pada setiap titik dalam citra, hasil perkiraan gradien dapat digabungkan, dan memberikan magnitudo gradien, menggunakan:

$$G = \sqrt{G_x^2 + G_y^2} \quad (5)$$

Menggunakan persamaan-6, arah gradien dapat dihitung dengan :

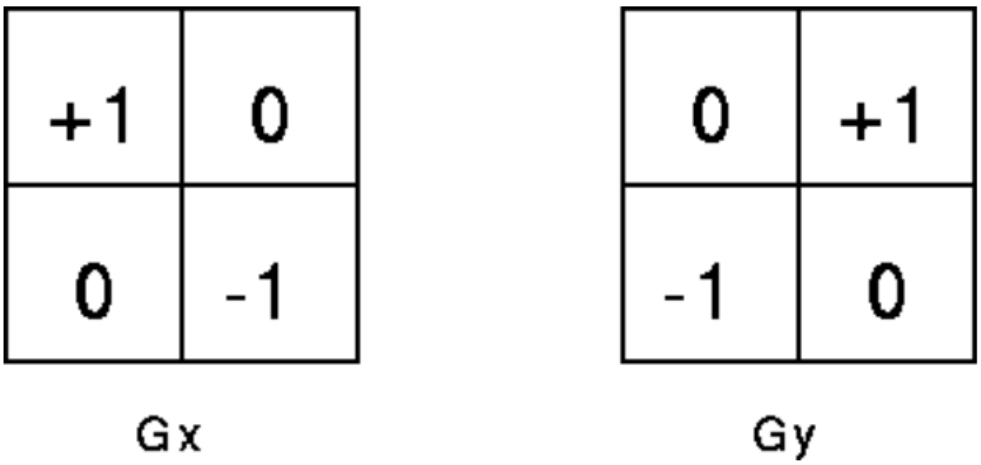
$$\theta = arctan\, arc\, \tan\left(\frac{G_y}{G_x}\right) \quad (6)$$

1.3.3 Laplacian of Gaussian Edge Detector

1.3.4 Prewitt Edge Detection

1.3.5 Robert Cross Edge Detection

Robert Cross edge detection merupakan operator deteksi tepi pada pengolahan citra dan *computer vision* untuk deteksi tepi. Operator ini terdiri dari dua 2x2 kernel konvolusi seperti pada gambar berikut. Salah satu kernel merupakan kernel lainnya yang diputar 90 derajat. Operator ini hampir sama dengan *Sobel edge detection.*



Gambar 1.1 Kernel konvolusi Robert Cross

Kernel-kernel ini dirancang untuk merespon secara maksimal ke tepi-tepi yang berjalan pada 45 derajat ke piksel *gris*, satu kernel untuk masing-masing dari dua orientasi tegak lurus. Kernel-kernel ini dapat diterapkan secara terpisah ke input citra, untuk menghasilkan pengukuran terpisah dari komponen gradien di setiap orientasi. Kedua orientasi ini yaitu

horisontal (Gx) dan vertikal (Gy). Kedua orientasi ini nantinya dapat digabungkan untuk mencari magnitudo absolut dari gradien pada setiap titik dan orientasi dari gradien tersebut. Berikut model matematika dari magnitudo gradien :

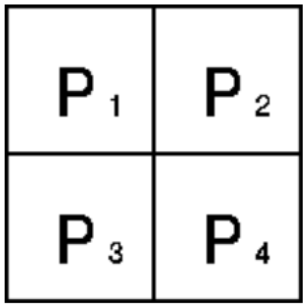
$$|G| = \sqrt{{G_x}^2 + {G_y}^2} \quad \text{(n)}$$

Arah dari gradien dapat dihitung menggunakan persamaan berikut :

$$\theta = arctan\, arc\, \tan\left(\frac{G_y}{G_x}\right) \quad \text{(n)}$$

Pada kasus ini, orientasi 0 diambil dengan maskud bahwa arah dari kontras maksimum dari hitam ke putih berjalan dari kiri ke kanan pada citra, dan sudut lain dihitung searah jarum jam dari sini.

Seringkali, pengguna hanya melihat output magnitudo absolut. Dua komponen dari gradien dihitung secara sederhana dan ditambahkan dalam satu kali *pass* pada input citra menggunakan operator *pseudo-convolution* seperti pada gambar berikut.



Gambar 1.2 Kernel pseudo-convolution

Dengan menggunakan kernel ini, perkiraan magnitudo dapat dihitung dengan persamaan berikut:

$$|G| = |P_1 - P_4| + |P_2 - P_3| \quad \text{(n)}$$

1.3.6 Zero Crossing Edge Detection

Detektor zero crossing mencari tempat dalam Laplacian suatu gambar di mana nilai Laplacian melewati angka nol , yaitu titik-titik di mana Laplacian berubah tanda. Titik-titik seperti itu sering terjadi pada `tepi' pada gambar --- yaitu titik di mana intensitas gambar

berubah dengan cepat, tetapi juga terjadi pada tempat-tempat yang tidak mudah diasosiasikan dengan tepi. Yang terbaik adalah menganggap detektor zero crossing sebagai semacam detektor fitur, bukan sebagai detektor tepi yang spesifik. Persilangan nol selalu terletak pada kontur tertutup, sehingga output dari detektor persilangan nol biasanya berupa gambar biner dengan garis ketebalan piksel tunggal yang menunjukkan posisi titik persilangan nol.

Titik awal untuk detektor zero crossing adalah gambar yang sudah difilter dengan menggunakan filter Laplacian of Gaussian. Persilangan nol yang dihasilkan sangat dipengaruhi oleh ukuran Gaussian yang digunakan untuk tahap penghalusan operator ini. Ketika penghalusan ditingkatkan, maka semakin sedikit kontur zero crossing yang akan ditemukan, dan kontur yang masih ada akan sesuai dengan fitur dengan skala yang lebih besar dan lebih besar pada gambar.

Inti dari detektor zero crossing adalah Laplacian dari filter Gaussian, sehingga pengetahuan tentang operator tersebut diasumsikan di sini. Seperti yang dijelaskan di sana, `tepi' pada gambar menimbulkan zero crossing pada output LoG.

Namun demikian, zero crossing juga terjadi di tempat mana pun di mana gradien intensitas gambar mulai meningkat atau mulai menurun, dan hal ini dapat terjadi di tempat yang tidak jelas tepinya. Sering kali zero crossing ditemukan di wilayah dengan gradien yang sangat rendah, di mana gradien intensitasnya bergoyang-goyang di sekitar nol.

Setelah gambar difilter LoG, tinggal mendeteksi zero crossing. Hal ini bisa dilakukan dengan beberapa cara.

Cara yang paling sederhana adalah dengan hanya membuat ambang batas output LoG pada nol, untuk menghasilkan gambar biner di mana batas-batas antara daerah latar depan dan latar belakang mewakili lokasi titik persilangan nol. Batas-batas ini kemudian dapat dengan mudah dideteksi dan ditandai dalam sekali jalan, misalnya dengan menggunakan beberapa operator morfologi. Sebagai contoh, untuk menemukan semua titik batas, kita hanya perlu menandai setiap titik latar depan yang memiliki setidaknya satu tetangga latar belakang.

Masalah dengan teknik ini adalah, akan cenderung membiaskan lokasi tepi persimpangan nol ke sisi terang dari tepi, atau sisi gelap dari tepi, tergantung pada apakah diputuskan untuk mencari tepi daerah latar depan atau tepi daerah latar belakang.

Teknik yang lebih baik adalah mempertimbangkan titik-titik pada kedua sisi batas ambang, dan memilih salah satu yang memiliki besaran absolut Laplacian yang paling rendah, yang mudah-mudahan akan paling dekat dengan zero crossing.

Karena zero crossing umumnya berada di antara dua piksel pada gambar yang difilter LoG, representasi output alternatif adalah kisi gambar yang secara spasial bergeser setengah piksel ke atas dan setengah piksel ke bawah, relatif terhadap gambar aslinya. Representasi semacam itu dikenal sebagai *kisi ganda*. Tentu saja, cara ini tidak melokalisasi zero crossing secara lebih akurat.

Pendekatan yang lebih akurat adalah melakukan semacam interpolasi untuk memperkirakan posisi zero crossing ke presisi sub-piksel.

BAB II
SOURCE CODE

2.1. Segmentasi

```
import cv2
import numpy as np


threshold = 0.2
kernel5 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(20,20))
x_co = 0
y_co = 0
hsv = None
Hue = 0
Saturation = 0
Value = 0
threshold_Hue = 180*threshold
threshold_Saturation = 255*threshold
threshold_Value = 255*threshold


def on_mouse(event,x,y,flag,param):
  global x_co,y_co,Hue,Saturation,Value,hsv
  if(event==cv2.EVENT_LBUTTONDOWN):
    x_co=x
    y_co=y
    p_sel = hsv[y_co][x_co]
    Hue = p_sel[0]
    Saturation = p_sel[1]
    Value = p_sel[2]


cv2.namedWindow("camera", 1)
cv2.namedWindow("camera2", 2)
```

```
cv2.namedWindow("camera3", 3)

#cam = video.create_capture(0)

image = cv2.imread("C:\\Users\\ASUS\\Pictures\\Fotokku.jpeg")

# image.set(cv2.CAP_PROP_FRAME_WIDTH, 640)

# image.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)


while True:

    # ret, src = image.read()

    source = cv2.blur(image, (3,3))

    hsv = cv2.cvtColor(source, cv2.COLOR_BGR2HSV)

    cv2.setMouseCallback("camera2",on_mouse, 0);


    minColor = np.array([Hue-threshold_Hue,Saturation-
threshold_Saturation,Value-threshold_Value])

    maxColor =
np.array([Hue+threshold_Hue,Saturation+threshold_Saturation,Value+thre
shold_Value])

    mask = cv2.inRange(hsv, minColor, maxColor)

    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel5)


    #res = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)

    cv2.putText(mask,"H:" + str(Hue)+" S:"+str(Saturation)+"
V:"+str(Value), (10,30), cv2.FONT_HERSHEY_PLAIN, 2.0, (255,255,255),
thickness = 1)

    cv2.imshow("camera", mask)

    cv2.imshow("camera2", source)

    source_segmented = cv2.add(source,source,mask=mask)

    cv2.imshow("camera3", source_segmented)

    if cv2.waitKey(10) == ord('q'):

        break


cv2.destroyAllWindows()
```

2.2. K-Means

```
import cv2

import numpy as np


image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg")

image = cv2.resize(image, (500, 500))

image_temp = image.reshape((-1, 3))


image_temp = np.float32(image_temp)


criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10,
1.0)

numberofKcluster = 4


ret, label, center = cv2.kmeans(image_temp, numberofKcluster, None,
criteria, 10, cv2.KMEANS_RANDOM_CENTERS)


center = np.uint8(center)

result = center[label.flatten()]

result2 = result.reshape((image.shape))


cv2.imshow('Original', image)

cv2.imshow('Result', result2)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.3. Canny Edge Detection

```
import cv2

import numpy as np


image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg")


canny = cv2.Canny(image, 150, 100)


cv2.imshow("Original", image)

cv2.imshow("Canny", canny)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.4. Sobel Edge Detection

```
import cv2



image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg"",
cv2.IMREAD_COLOR)

image = cv2.GaussianBlur(image, (3, 3), 0)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


ddepth = cv2.CV_16S

scale = 1

delta = 0


gradientX = cv2.Sobel(gray, ddepth, 1, 0, ksize=3, scale=scale,
delta=delta, borderType=cv2.BORDER_DEFAULT)

gradientY = cv2.Sobel(gray, ddepth, 0, 1, ksize=3, scale=scale,
delta=delta, borderType=cv2.BORDER_DEFAULT)



absoluteGradientX = cv2.convertScaleAbs(gradientX)

absoluteGradientY = cv2.convertScaleAbs(gradientY)
```

```
sobel = cv2.addWeighted(absoluteGradientX, 0.5, absoluteGradientY,
0.5, 0)


cv2.imshow("Original", image)

cv2.imshow("Sobel", sobel)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.5. Laplacian of Gaussian Edge Detector

```
import cv2


image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg"",
cv2.IMREAD_COLOR)

image = cv2.GaussianBlur(image, (3, 3), 0)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


ddepth = cv2.CV_16S

kernel_size = 3

laplacian = cv2.Laplacian(gray, ddepth, ksize=kernel_size)

laplacianAbsolute = cv2.convertScaleAbs(laplacian)


cv2.imshow("Original", image)

cv2.imshow("Laplacian", laplacianAbsolute)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.6. Prewitt Edge Detection

```
import cv2
import numpy as np


image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (3, 3), 0)


kernelX = np.array([
    [-1, 0, 1],
    [-1, 0, 1],
    [-1, 0, 1]
])


kernelY = np.array([
    [-1, -1, -1],
    [0, 0, 0],
    [1, 1, 1]
])


prewittX = cv2.filter2D(gray, -1, kernelX)
prewittY = cv2.filter2D(gray, -1, kernelY)
prewittX = np.float32(prewittX)
prewittY = np.float32(prewittY)


prewitt = cv2.magnitude(prewittX, prewittY)
cv2.imshow("Original", image)
cv2.imshow("Prewitt", prewitt)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.7. Robert Cross Edge Detection

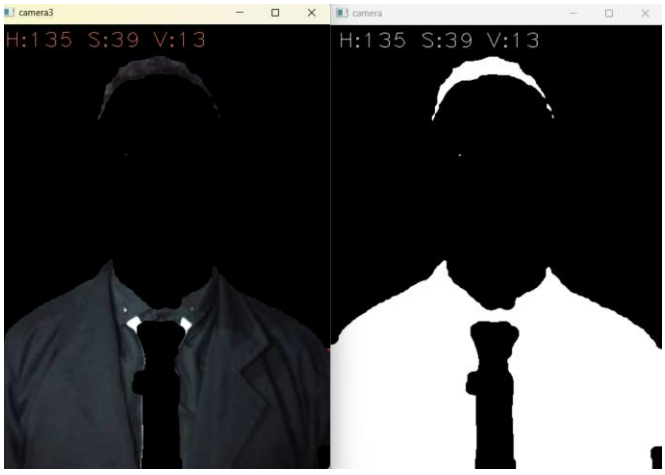
```
import cv2

import numpy as np

from scipy import ndimage

image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg").asty
pe(np.float32)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


kernelX = np.array([[1, 0], [0, -1]])

kernelY = np.array([[0, 1], [-1, 0]])


robertX = ndimage.convolve(gray, kernelX)

robertY = ndimage.convolve(gray, kernelY)

robertX = np.float32(robertX)

robertY = np.float32(robertY)

robert = np.sqrt(np.square(robertX) + np.square(robertY))

_, robert = cv2.threshold(robert, 50, 255, cv2.THRESH_BINARY)


cv2.imshow("Robert Cross.jpg", robert)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.8. Zero Crossing Edge Detection

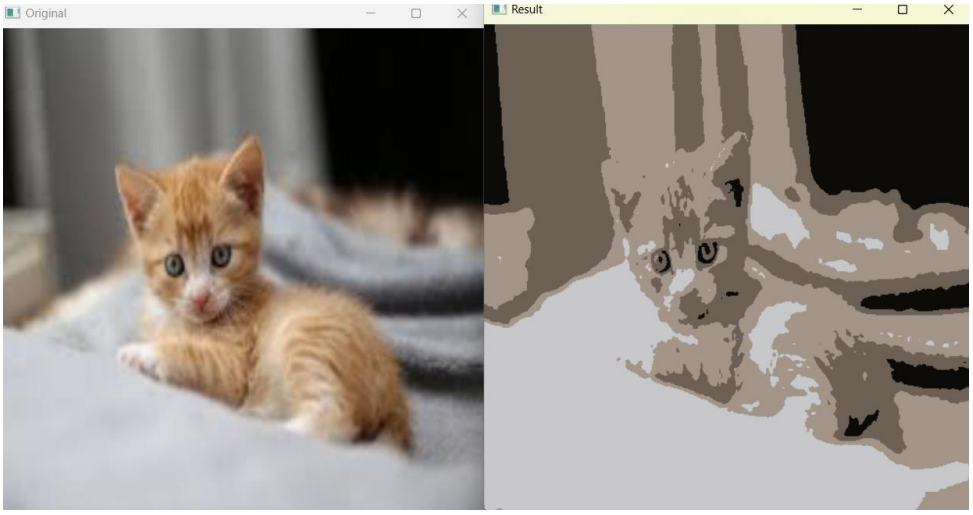
```
import cv2


image =
cv2.imread("C:\\Users\\ASUS\\Downloads\\presentasi\\kitten.jpeg",
cv2.IMREAD_COLOR)

image = cv2.GaussianBlur(image, (3, 3), 0)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


ddepth = cv2.CV_16S

kernel_size = 3

laplacian = cv2.Laplacian(gray, ddepth, ksize=kernel_size)

laplacianAbsolute = cv2.convertScaleAbs(laplacian)

_, zeroCross = cv2.threshold(laplacianAbsolute, 45, 255,
cv2.THRESH_BINARY)


cv2.imshow("Original", image)

cv2.imshow("Zero Crossing", zeroCross)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

2.9.Output

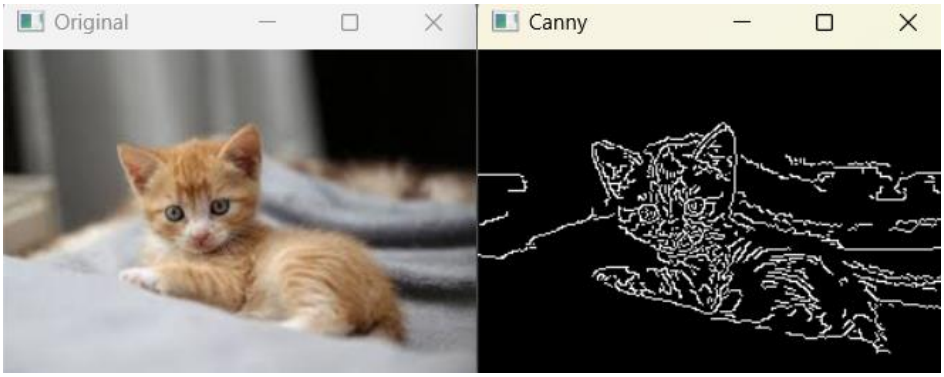
- Segmentasi



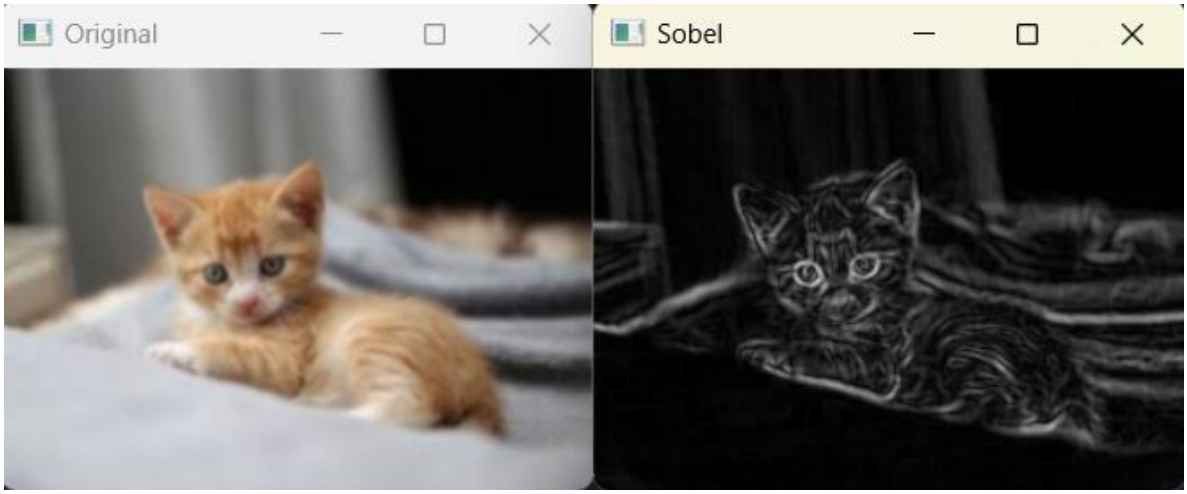
- K-Means



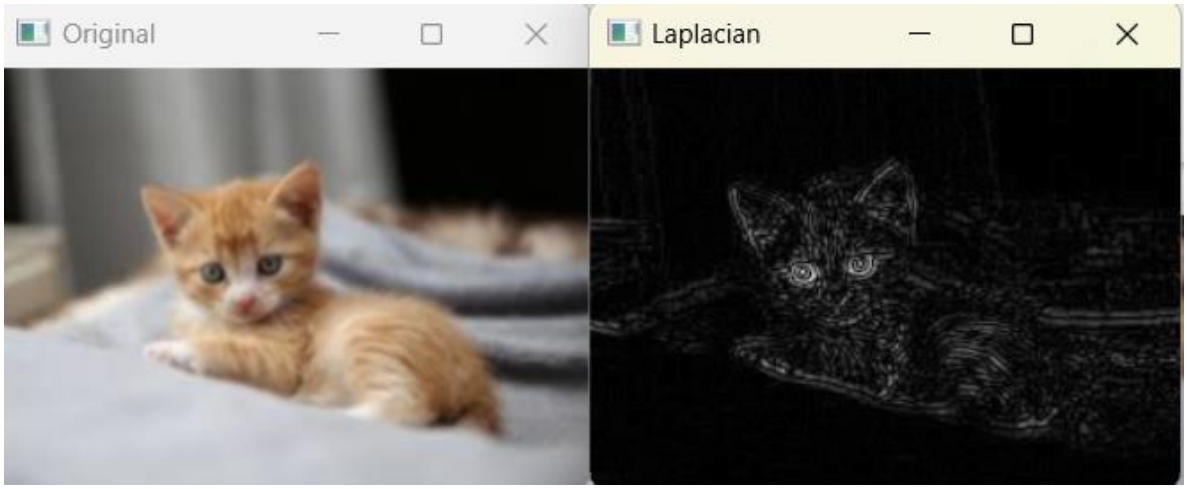
- Canny



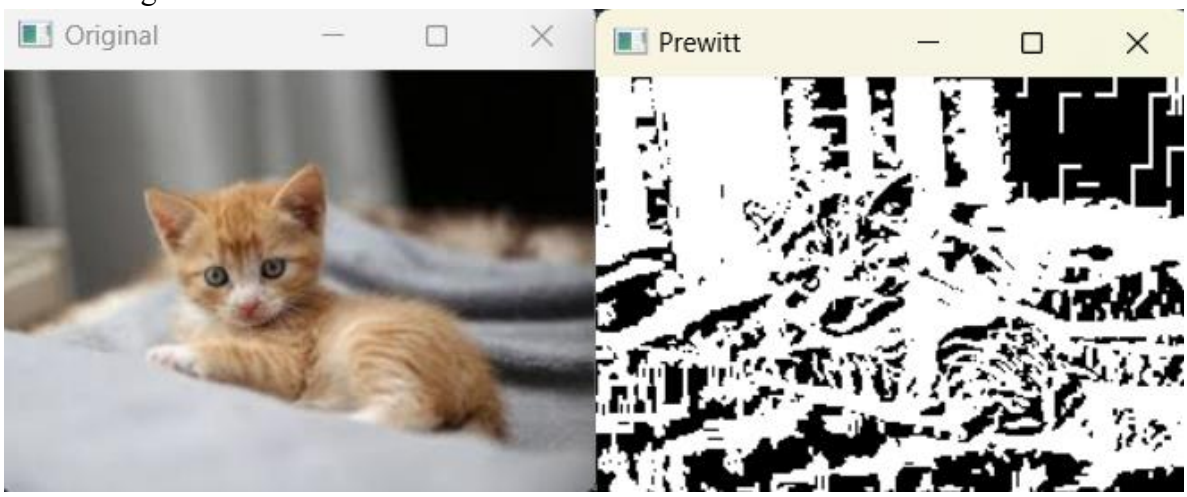
- Sobel



- Laplacian of Gaussian



- Prewitt Edge



- Robert Cross

- Zero Crosssing