LAPORAN

PENGOLAHAN CITRA DIGITAL

DISUSUN OLEH :

Anggi Ramadan

226201032

POLITEKNIK NEGERI SAMARINDA

2024

BAB I

PENDAHULUAN

SEGMENTASI

Segmentasi dalah proses membagi gambar menjadi beberapa bagian atau segmen yang lebih kecil untuk memudahkan analisis. Tujuan dari segmentasi adalah untuk memisahkan objek atau wilayah tertentu dalam citra yang memiliki karakteristik serupa, seperti warna, tekstur, atau intensitas.

K-MEANS

K-means adalah metode clustering yang digunakan dalam pengolahan citra digital untuk mengelompokkan piksel berdasarkan kesamaan fitur, seperti warna atau intensitas. K-means sering digunakan dalam segmentasi citra untuk memisahkan objek dari latar belakang atau mengidentifikasi wilayah dengan karakteristik serupa. Metode ini terkenal karena kesederhanaannya dan kecepatan eksekusi, meskipun bisa kurang efektif pada data yang tidak berbentuk bulat atau memiliki variasi yang besar.

SOBEL

Sobel adalah metode deteksi tepi yang menggunakan dua filter konvolusi, satu untuk mendeteksi perubahan horizontal (Gx) dan satu untuk mendeteksi perubahan vertikal (Gy). Filter ini menghitung gradien intensitas citra, memungkinkan identifikasi area dengan perbedaan yang signifikan dalam intensitas.

CANNY

Canny adalah algoritma deteksi tepi yang melalui beberapa langkah. Pertama, citra dihaluskan menggunakan Gaussian untuk mengurangi noise. Kemudian, gradien citra dihitung untuk menemukan kekuatan dan arah tepi. Langkah berikutnya adalah pengurangan non-maximum, di mana hanya piksel dengan nilai maksimum di sepanjang arah gradien yang dipertahankan. Terakhir, thresholding dilakukan untuk menentukan tepi yang kuat dan lemah.

PREWITT

Prewitt adalah metode yang mirip dengan Sobel, menggunakan dua filter untuk mendeteksi tepi horizontal dan vertikal. Filter ini bekerja dengan cara menghitung gradien intensitas citra dengan pendekatan bobot yang lebih merata, memfokuskan pada perbedaan intensitas di sekitar piksel.

ROBERTS

Roberts menggunakan dua filter kecil yang berfokus pada piksel bersebelahan untuk mendeteksi tepi diagonal. Metode ini menekankan perubahan intensitas yang tajam di antara dua piksel, efektif untuk mendeteksi tepi dengan sudut.

LoG (Laplacian of Gaussian)

Laplacian of Gaussian menggabungkan penghalusan citra dengan Gaussian dan penerapan operator Laplacian. Langkah pertama adalah menghaluskan citra, diikuti oleh penerapan Laplacian untuk menemukan area dengan perubahan intensitas. Metode ini membantu dalam mendeteksi tepi di area dengan noise rendah.

ZERO CROSSING

Zero Crossing adalah metode yang mendeteksi tepi dengan mencari titik di mana nilai fungsi berubah tanda, biasanya setelah menerapkan filter seperti Laplacian. Metode ini berfokus pada perubahan di sekitar piksel dan menghasilkan garis tepi berdasarkan titik perpotongan.

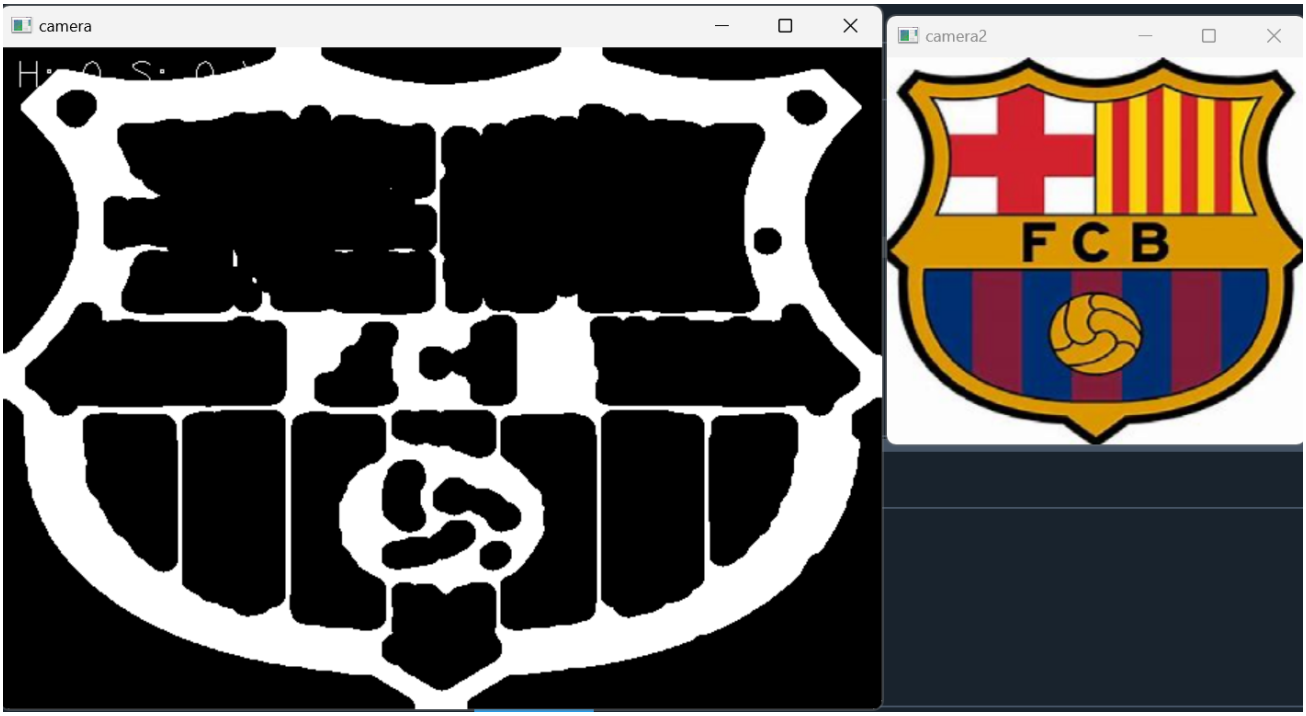
BAB II

SEGMENTASI

```
import cv2  # Mengimpor pustaka OpenCV, yang digunakan untuk pemrosesan gambar dan video.

import numpy as np  # Mengimpor pustaka NumPy, yang digunakan untuk manipulasi array dan operasi numerik.


# Threshold settings

threshold = 0.8  # Menentukan nilai threshold yang akan digunakan untuk menghitung batas warna pada ruang warna HSV.

kernel5 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (20, 20))  # Membuat elemen struktural berbentuk elips dengan ukuran 20x20 untuk operasi morfologi.


# Global variables for color selection

x_co, y_co = 0, 0  # Variabel global untuk menyimpan koordinat x dan y saat pengguna mengklik gambar.

hsv = None  # Variabel global untuk menyimpan citra yang telah dikonversi ke ruang warna HSV.

H, S, V = 0, 0, 0  # Variabel global untuk menyimpan nilai Hue, Saturation, dan Value dari piksel yang dipilih.

thr_H = 180 * threshold  # Threshold untuk rentang nilai Hue (warna), dihitung berdasarkan threshold yang telah ditetapkan.

thr_S = 255 * threshold  # Threshold untuk rentang nilai Saturation (kejenuhan).

thr_V = 255 * threshold  # Threshold untuk rentang nilai Value (kecerahan).


# Mouse callback function to get HSV values

def on_mouse(event, x, y, flags, param):  # Fungsi callback untuk menangani event mouse, digunakan untuk mendapatkan nilai HSV dari piksel yang diklik.

    global x_co, y_co, H, S, V, hsv  # Menggunakan variabel global agar dapat diakses dari dalam fungsi.

    if event == cv2.EVENT_LBUTTONDOWN:  # Mengecek jika tombol kiri mouse ditekan.

        x_co, y_co = x, y  # Menyimpan koordinat x dan y tempat klik terjadi.

        p_sel = hsv[y_co][x_co]  # Mengambil nilai HSV dari koordinat yang diklik.

        H, S, V = p_sel[0], p_sel[1], p_sel[2]  # Menyimpan nilai H, S, dan V dari piksel yang diklik.
```

```
# Load the image instead of capturing from the camera

image_path = 'C:/Users/Asus/Pictures/barca.jpg' # Mengatur path ke gambar yang akan dimuat, ganti dengan path yang sesuai.

src = cv2.imread(image_path)  # Membaca gambar dari file.


if src is None:  # Mengecek apakah gambar berhasil dimuat.

    print("Error: Could not read the image.")  # Menampilkan pesan error jika gambar tidak berhasil dimuat.

    exit()  # Keluar dari program jika gambar tidak terbaca.


# Resize the image if needed

src = cv2.resize(src, (640, 480))  # Mengubah ukuran gambar menjadi 640x480 piksel.


# Create windows for displaying the image

cv2.namedWindow("camera", 1)  # Membuat jendela bernama "camera" untuk menampilkan hasil pemrosesan gambar.

cv2.namedWindow("camera2", 2)  # Membuat jendela bernama "camera2" untuk menampilkan gambar asli.

cv2.setMouseCallback("camera2", on_mouse)  # Mengatur callback mouse pada jendela "camera2", sehingga klik di gambar ini akan memanggil fungsi `on_mouse`.


# Convert BGR to HSV

hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)  # Mengonversi gambar dari format BGR (yang digunakan oleh OpenCV) ke ruang warna HSV.


while True:  # Memulai loop utama.

    # Define color range for masking

    min_color = np.array([H - thr_H, S - thr_S, V - thr_V])  # Menentukan batas bawah warna berdasarkan nilai H, S, dan V yang dipilih, dikurangi dengan threshold.

    max_color = np.array([H + thr_H, S + thr_S, V + thr_V])  # Menentukan batas atas warna berdasarkan nilai H, S, dan V yang dipilih, ditambah dengan threshold.

    mask = cv2.inRange(hsv, min_color, max_color)  # Membuat mask berdasarkan rentang warna yang ditentukan, hanya piksel dalam rentang yang akan disorot.


    # Morphological operations to clean up the mask

    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel5)  # Melakukan operasi morfologi untuk menghilangkan noise pada mask menggunakan elemen struktural `kernel5`.
```

```
    # Display the selected HSV values
    cv2.putText(mask, f"H: {H} S: {S} V: {V}", (10, 30), cv2.FONT_HERSHEY_PLAIN, 2.0, (255, 255, 255), thickness=1)  # Menampilkan nilai H, S, dan V yang dipilih pada gambar.

    # Show the mask and original image
    cv2.imshow("camera", mask)  # Menampilkan mask pada jendela "camera".
    cv2.imshow("camera2", src)  # Menampilkan gambar asli pada jendela "camera2".

    # Exit on pressing 'ESC'
    if cv2.waitKey(10) == 27:  # Mengecek jika tombol 'ESC' ditekan untuk keluar dari loop.
        break  # Keluar dari loop jika tombol 'ESC' ditekan.

# Close windows
cv2.destroyAllWindows()  # Menutup semua jendela OpenCV.
```



K-MEANS

```
import numpy as np

import matplotlib.pyplot as plt

import cv2


# Membaca gambar

image_path = r' C:/Users/Asus/Pictures/barca.jpg

image = cv2.imread(image_path)


# Mengonversi BGR ke RGB (OpenCV membaca gambar dalam format BGR,
sementara matplotlib mengharapkan format RGB)

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


# Menampilkan gambar asli

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(image)

plt.title('Gambar Asli')


# Mengubah bentuk gambar menjadi array 2D dengan nilai-nilai 3 komponen
warna (RGB)

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


# Mengonversi tipe data ke float untuk k-means

pixel_vals = np.float32(pixel_vals)


# Mendefinisikan kriteria penghentian untuk k-means:

# Penghentian jika 100 iterasi telah dijalankan, atau akurasi (epsilon)
mencapai 0,85

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.85)


# Melakukan k-means clustering dengan jumlah cluster yang didefinisikan
sebagai 3 (untuk segmentasi)
```

```
k = 3  # Jumlah cluster (dapat diubah sesuai kebutuhan)
_, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

# Mengonversi pusat (centers) ke nilai 8-bit
centers = np.uint8(centers)

# Memetakan label ke pusat (warna) untuk setiap pixel
segmented_data = centers[labels.flatten()]

# Mengubah bentuk data yang telah disegmentasi ke dimensi gambar asli
segmented_image = segmented_data.reshape((image.shape))

# Menampilkan gambar hasil segmentasi
plt.subplot(1, 2, 2)
plt.imshow(segmented_image)
plt.title('Segmentasi gambar dengan K-Means')

# Menampilkan kedua gambar
plt.show()
```

SOBEL

```
import cv2  # Kita perlu mengimpor OpenCV untuk memproses gambar
import numpy as np  # Numpy digunakan untuk operasi numerik dan array
import matplotlib.pyplot as plt  # Matplotlib digunakan untuk menampilkan gambar

# Baca gambar aslinya dan ubah menjadi grayscale
# Fungsi ini akan mengambil gambar dari path yang kita berikan, dan mengubahnya jadi gambar hitam putih (grayscale)
original_image = cv2.imread(' C:/Users/Asus/Pictures/barca.jpg', cv2.IMREAD_GRAYSCALE)

# Sobel X ini akan menghitung perubahan intensitas (gradien) di sepanjang sumbu X (horizontal)
# Sobel menggunakan kernel 5x5 untuk memproses setiap piksel
sobel_x = cv2.Sobel(original_image, cv2.CV_64F, 1, 0, ksize=5)

# Sobel Y ini akan menghitung perubahan intensitas di sepanjang sumbu Y (vertikal)
sobel_y = cv2.Sobel(original_image, cv2.CV_64F, 0, 1, ksize=5)

# Fungsi cv2.magnitude() akan menghitung seberapa besar perubahan gradien pada kedua arah (X dan Y)
sobel_combined = cv2.magnitude(sobel_x, sobel_y)

# Pertama, kita buat jendela (figure) dengan ukuran 10x5 inci
plt.figure(figsize=(10,5))

plt.subplot(1, 3, 1)  # Mengatur posisi gambar di grid 1 baris dan 3 kolom
plt.imshow(original_image, cmap='gray')  # Menampilkan gambar dalam skala abu-abu
plt.title('Original')  # Memberikan judul "Original" pada gambar ini
plt.axis('off')  # Mematikan tampilan sumbu agar terlihat lebih bersih

# Di kolom kedua, kita tampilkan hasil Sobel X (deteksi tepi horizontal)
plt.subplot(1, 3, 2)
```

```
plt.imshow(sobel_x, cmap='gray')  # Menampilkan hasil Sobel di arah X
plt.title('Sobel X')  # Judulnya "Sobel X"
plt.axis('off')  # Sumbu juga kita matikan di sini

# Di kolom ketiga, kita tampilkan hasil Sobel Y (deteksi tepi vertikal)
plt.subplot(1, 3, 3)
plt.imshow(sobel_y, cmap='gray')  # Menampilkan hasil Sobel di arah Y
plt.title('Sobel Y')  # Judulnya "Sobel Y"
plt.axis('off')  # Mematikan sumbu di sini juga

# Untuk menggabungkan Sobel X dan Y, kita buat jendela baru
plt.figure()
plt.imshow(sobel_combined, cmap='gray')  # Menampilkan hasil kombinasi Sobel X dan Y
plt.title('Sobel Combined (Magnitude)')  # Judulnya "Sobel Combined"
plt.axis('off')  # Mematikan sumbu lagi agar terlihat lebih bersih

# Terakhir, kita tampilkan semua gambar yang sudah disiapkan
plt.show()
```

CANNY

```
import cv2  # Mengimpor OpenCV untuk pemrosesan gambar
import matplotlib.pyplot as plt  # Mengimpor Matplotlib untuk menampilkan gambar

# cv2.imread() membaca gambar dari disk, dan dengan cv2.IMREAD_GRAYSCALE kita konversi gambar jadi grayscale
original_image = cv2.imread(' C:/Users/Asus/Pictures/barca.jpg', cv2.IMREAD_GRAYSCALE)

# Fungsi Canny menggunakan dua nilai threshold. Threshold pertama (100) menentukan batas bawah,
# threshold kedua (200) adalah batas atas. Piksel dengan nilai gradien di antara keduanya dianggap tepi.
canny_edges = cv2.Canny(original_image, 100, 200)

# plt.figure(figsize=(10,5)) membuat jendela tampilan dengan ukuran 10x5 inci
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)  # plt.subplot(1, 2, 1) artinya 1 baris, 2 kolom, dan gambar di posisi pertama
plt.imshow(original_image, cmap='gray')  # Tampilkan gambar dalam skala abu-abu
plt.title('Original Image')  # Judul gambar: "Original Image"
plt.axis('off')  # Matikan tampilan sumbu (untuk tampilan lebih rapi)

# Di bagian kanan, kita tampilkan hasil dari Canny
plt.subplot(1, 2, 2)  # plt.subplot(1, 2, 2) artinya 1 baris, 2 kolom, dan gambar di posisi kedua
plt.imshow(canny_edges, cmap='gray')  # Tampilkan hasil deteksi tepi menggunakan filter Canny
plt.title('Canny Edges')  # Judul gambar: "Canny Edges"
plt.axis('off')  # Matikan tampilan sumbu juga untuk hasil yang lebih rapi

# Menampilkan kedua gambar
plt.show()
```

PREWITT

```
import cv2 # Kita pakai OpenCV untuk membaca dan memproses gambar
import numpy as np  # NumPy digunakan untuk membuat kernel Prewitt
import matplotlib.pyplot as plt # Matplotlib kita pakai untuk menampilkan gambar


# Pertama, kita baca gambar asli dalam format grayscale
# Fungsi cv2.imread() digunakan untuk membaca gambar, dan kita ubah ke grayscale agar lebih mudah mendeteksi tepinya
original_image = cv2.imread('C:/Users/Asus/Pictures/barca.jpg', cv2.IMREAD_GRAYSCALE)


# Sekarang kita buat filter Prewitt untuk mendeteksi tepi di arah X (horizontal)
# Kernel Prewitt di arah X mendeteksi perubahan intensitas piksel di sepanjang sumbu X
# Kernel ini memiliki bentuk 3x3 dengan nilai-nilai yang sudah spesifik untuk mendeteksi perubahan di arah horizontal
prewitt_kernel_x = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
# Fungsi cv2.filter2D() digunakan untuk menerapkan kernel ini pada gambar
prewitt_x = cv2.filter2D(original_image, -1, prewitt_kernel_x)


# Lanjut, kita buat filter Prewitt untuk mendeteksi tepi di arah Y (vertikal)
# Kernel Prewitt di arah Y mendeteksi perubahan intensitas piksel di sepanjang sumbu Y
# Kernel ini memiliki nilai untuk mendeteksi perubahan vertikal
prewitt_kernel_y = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
```

Kita juga terapkan kernel ini pada gambar dengan fungsi filter2D

```
prewitt_y = cv2.filter2D(original_image, -1, prewitt_kernel_y)
```

Setelah kita punya hasil untuk Prewitt di arah X dan Y, kita gabungkan keduanya

cv2.magnitude() menghitung magnitude dari hasil Prewitt X dan Y, yang akan memberikan kita kombinasi keduanya

```
prewitt_combined = cv2.magnitude(prewitt_x.astype(float),
prewitt_y.astype(float))
```

Sekarang kita tampilkan hasilnya

Kita buat jendela tampilan ukuran 10x5 inci untuk gambar asli dan hasil Prewitt

```
plt.figure(figsize=(10, 5))
```

Menampilkan gambar asli

```
plt.subplot(1, 3, 1) # Ini artinya gambar akan ditempatkan di kolom pertama dari grid 1 baris dan 3 kolom

plt.imshow(original_image, cmap='gray') # Gambar ditampilkan dalam skala abu-abu

plt.title('Original') # Judul untuk gambar asli

plt.axis('off') # Matikan sumbu agar tidak terlihat
```

Menampilkan hasil Prewitt X (deteksi tepi horizontal)

```
plt.subplot(1, 3, 2) # Ini gambar akan ditempatkan di kolom kedua

plt.imshow(prewitt_x, cmap='gray') # Tampilkan hasil Prewitt X

plt.title('Prewitt X') # Judulnya Prewitt X

plt.axis('off') # Matikan sumbu
```

Menampilkan hasil Prewitt Y (deteksi tepi vertikal)

```
plt.subplot(1, 3, 3) # Gambar di kolom ketiga

plt.imshow(prewitt_y, cmap='gray') # Tampilkan hasil Prewitt Y

plt.title('Prewitt Y') # Judul Prewitt Y

plt.axis('off') # Matikan sumbu
```

Terakhir, kita tampilkan gabungan Prewitt X dan Y

```
plt.figure() # Buat jendela baru
plt.imshow(prewitt_combined, cmap='gray') # Tampilkan hasil gabungan Prewitt X dan Y
plt.title('Prewitt Combined') # Judul untuk gambar gabungan
plt.axis('off')  # Matikan sumbu
```

Menampilkan semua gambar

```
plt.show()
```

ROBERT

```
import cv2 # OpenCV untuk membaca dan memproses gambar
import numpy as np # NumPy digunakan untuk membuat kernel
import matplotlib.pyplot as plt # Matplotlib untuk menampilkan gambar

# Baca gambar asli dalam grayscale
original_image = cv2.imread(' C:/Users/Asus/Pictures/barca.jpg', cv2.IMREAD_GRAYSCALE)

# Kernel Roberts Cross yang sudah langsung menggabungkan arah X dan Y
# Kernel ini bekerja mendeteksi tepi dari perubahan intensitas di kedua arah diagonal
roberts_kernel_1 = np.array([[1, 0], [0, -1]]) # Roberts di diagonal 1
roberts_kernel_2 = np.array([[0, 1], [-1, 0]]) # Roberts di diagonal 2

# Terapkan kedua kernel Roberts langsung pada gambar
roberts_edge_1 = cv2.filter2D(original_image, -1, roberts_kernel_1) # Deteksi tepi diagonal pertama
roberts_edge_2 = cv2.filter2D(original_image, -1, roberts_kernel_2) # Deteksi tepi diagonal kedua

# Gabungkan kedua hasil untuk mendapatkan deteksi tepi yang lengkap
roberts_combined = cv2.magnitude(roberts_edge_1.astype(float), roberts_edge_2.astype(float))

# Tampilkan gambar hasil deteksi tepi menggunakan Matplotlib
plt.figure(figsize=(10, 5))

# Gambar asli
plt.subplot(1, 2, 1) # 1 baris, 2 kolom, gambar di posisi pertama
plt.imshow(original_image, cmap='gray')  # Tampilkan gambar dalam skala abu-abu
plt.title('Original Image') # Judul gambar asli
plt.axis('off') # Matikan tampilan sumbu
```

Gambar hasil deteksi tepi dengan Roberts

```
plt.subplot(1, 2, 2) # Gambar di posisi kedua

plt.imshow(roberts_combined, cmap='gray') # Tampilkan hasil deteksi tepi

plt.title('Roberts Edge Detection') # Judul gambar deteksi tepi

plt.axis('off') # Matikan tampilan sumbu
```

Menampilkan kedua gambar

```
plt.show()
```

LoG

```
import cv2 # Mengimpor OpenCV untuk membaca dan memproses gambar

import numpy as np # Mengimpor NumPy untuk manipulasi array

import matplotlib.pyplot as plt # Mengimpor Matplotlib untuk menampilkan gambar
```

Baca gambar asli dalam grayscale

Kita menggunakan cv2.imread() untuk membaca gambar dan mengonversinya menjadi grayscale agar lebih mudah dalam analisis

```
original_image = cv2.imread(' C:/Users/Asus/Pictures/barca.jpg',
cv2.IMREAD_GRAYSCALE)
```

Terapkan Gaussian Blur untuk mengurangi noise

Kita menggunakan cv2.GaussianBlur() untuk membuat gambar lebih halus sebelum mendeteksi tepi

Dengan menggunakan kernel berukuran 5x5, kita mengurangi noise yang dapat mengganggu proses deteksi tepi

```
gaussian_blur = cv2.GaussianBlur(original_image, (5, 5), 0)
```

Terapkan Laplacian untuk mendeteksi tepi

Fungsi cv2.Laplacian() digunakan untuk menerapkan operator Laplacian pada gambar yang sudah di-blur

Operator ini akan memberikan nilai besar di area yang memiliki perubahan intensitas yang signifikan, yaitu tepi

```
laplacian_edges = cv2.Laplacian(gaussian_blur, cv2.CV_64F)
```

Ambil nilai absolut untuk menghindari nilai negatif

Kita mengambil nilai absolut dari hasil Laplacian agar semua tepi terdeteksi sebagai nilai positif

```
laplacian_edges = np.abs(laplacian_edges)
```

Tampilkan gambar

Kita menggunakan Matplotlib untuk menampilkan gambar asli dan hasil deteksi tepi

```
plt.figure(figsize=(10, 5))
```

Menampilkan gambar asli

```
plt.subplot(1, 2, 1) # Grid 1 baris, 2 kolom, gambar di posisi pertama
plt.imshow(original_image, cmap='gray') # Tampilkan gambar dalam skala abu-abu
plt.title('Original Image') # Judul gambar asli
plt.axis('off') # Matikan tampilan sumbu agar tampilan lebih bersih
```

Menampilkan hasil deteksi tepi menggunakan LoG

```
plt.subplot(1, 2, 2) # Gambar di posisi kedua
plt.imshow(laplacian_edges, cmap='gray') # Tampilkan hasil deteksi tepi
plt.title('Laplacian of Gaussian (LoG)') # Judul gambar hasil deteksi tepi
plt.axis('off') # Matikan tampilan sumbu
```

Menampilkan kedua gambar

```
plt.show()
```

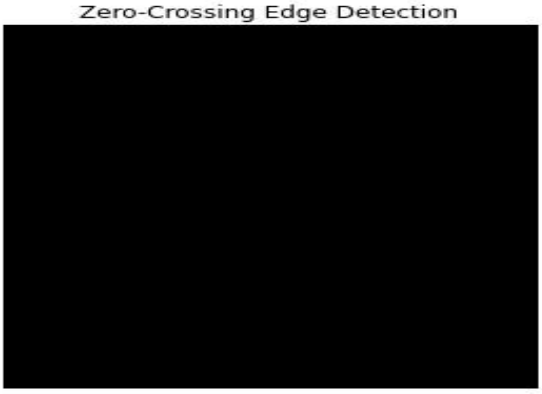
ZERRO CROSSING

```
import cv2 # Mengimpor OpenCV untuk membaca dan memproses gambar
import numpy as np # Mengimpor NumPy untuk manipulasi array
import matplotlib.pyplot as plt # Mengimpor Matplotlib untuk menampilkan gambar

# Baca gambar asli dalam grayscale
original_image = cv2.imread('C:/Users/Asus/Pictures/barca.jpg', cv2.IMREAD_GRAYSCALE)

# Terapkan Gaussian Blur untuk mengurangi noise
gaussian_blur = cv2.GaussianBlur(original_image, (5, 5), 0)

# Terapkan Laplacian untuk mendeteksi tepi
laplacian_edges = cv2.Laplacian(gaussian_blur, cv2.CV_64F)

# Ambil nilai absolut dari Laplacian
laplacian_edges = np.abs(laplacian_edges)

# Buat gambar untuk Zero-Crossing
zero_crossing = np.zeros_like(laplacian_edges, dtype=np.uint8)
```

```
# Deteksi zero-crossing
for i in range(1, laplacian_edges.shape[0] - 1):
    for j in range(1, laplacian_edges.shape[1] - 1):
    # Periksa jika ada perubahan tanda
        if (laplacian_edges[i, j] > 0 and
            (laplacian_edges[i-1, j] < 0 or
             laplacian_edges[i+1, j] < 0 or
             laplacian_edges[i, j-1] < 0 or
             laplacian_edges[i, j+1] < 0)):
            zero_crossing[i, j] = 255  # Tandai titik zero-crossing

# Tampilkan gambar
plt.figure(figsize=(10, 5))

# Menampilkan gambar asli
plt.subplot(1, 2, 1) # Grid 1 baris, 2 kolom, gambar di posisi pertama
plt.imshow(original_image, cmap='gray') # Tampilkan gambar dalam skala abu-abu
plt.title('Original Image') # Judul gambar asli
plt.axis('off') # Matikan tampilan sumbu agar tampilan lebih bersih

# Menampilkan hasil deteksi tepi menggunakan Zero-Crossing
plt.subplot(1, 2, 2) # Gambar di posisi kedua
plt.imshow(zero_crossing, cmap='gray') # Tampilkan hasil deteksi tepi
plt.title('Zero-Crossing Edge Detection') # Judul gambar hasil deteksi tepi
plt.axis('off') # Matikan tampilan sumbu

# Menampilkan kedua gambar
plt.show()
```

Original Image

Zero-Crossing Edge Detection



BAB III

KESIMPULAN

Bab I memperkenalkan berbagai metode segmentasi dan deteksi tepi dalam pengolahan citra digital. Segmentasi memisahkan citra menjadi bagian-bagian berdasarkan karakteristik tertentu, seperti warna atau tekstur K-means: Algoritma clustering yang digunakan untuk mengelompokkan piksel berdasarkan kemiripan warna atau intensitas.

Bab II menjelaskan implementasi teknis dari berbagai metode segmentasi dan deteksi tepi menggunakan bahasa pemrograman Python dengan pustaka OpenCV dan NumPy, yang disertai langkah-langkah pemrograman seperti pengolahan gambar, penerapan algoritma K-means, serta berbagai filter untuk deteksi tepi (Sobel, Canny, Prewitt, dan lainnya).