

UAS
PENGOLAHAN CITRA



NAMA : Nanda Revan Saputro

NIM : 202331016

KELAS : B

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

NO.PC : 15

ASISTEN : 1. Sasikirana Ramadhanty Setiawan Putri

2. Rizqy Amanda

3. Ridho Chaerullah

4. Sakura Amastasya Salsabila Setiyanto

INSTITUT TEKNOLOGI PLN
TEKNIK INFORMATIKA
2024/2025

DAFTAR ISI

DAFTAR ISI	2
BAB I PENDAHULUAN	3
1.1 Rumusan Masalah	3
1.2 Tujuan Masalah	3
1.3 Manfaat Masalah	3
BAB II LANDASAN TEORI	4
2.1. Deteksi Tepi Pola Objek	4
2.2. Deteksi Daun	4
2.3. Kompresi Citra	5
BAB III HASIL	7
1. Deteksi Tepi Pola Objek	7
2. Deteksi Daun	15
3. Kompresi	23
BAB IV PENUTUP	29
DAFTAR PUSTAKA	31

BAB I

PENDAHULUAN

1.1 Rumusan Masalah

Berdasarkan hasil UAS praktikum Pengolahan Citra Digital, terdapat beberapa masalah utama yang menjadi fokus dalam percobaan dan analisis, yaitu:

1. Bagaimana penerapan deteksi tepi dapat membantu dalam identifikasi pola objek secara visual?
2. Bagaimana proses mendeteksi dan mengekstraksi objek daun dari latar belakang menggunakan metode segmentasi?
3. Bagaimana teknik kompresi citra diterapkan untuk mengurangi ukuran file tanpa menghilangkan informasi penting?
4. Apa pengaruh pemilihan metode deteksi tepi dan segmentasi terhadap kualitas hasil pengolahan citra?

Rumusan masalah ini diangkat sebagai dasar analisis praktikum untuk memahami penerapan teori dalam menggunakan software atau library seperti OpenCV dan Python.

1.2 Tujuan Masalah

Tujuan dari praktikum ini adalah untuk:

1. Menggunakan teknik segmentasi untuk mendeteksi dan mengisolasi objek daun dari citra berwarna.
2. Menerapkan teknik kompresi citra baik secara lossless maupun lossy.
3. Menganalisis perbandingan hasil antara berbagai metode deteksi dan segmentasi citra.
4. Memberikan pemahaman praktis tentang penerapan algoritma pengolahan citra menggunakan bahasa pemrograman (seperti Python dan OpenCV).

1.3 Manfaat Masalah

Manfaat yang diperoleh dari praktikum ini antara lain:

1. Menambah pemahaman tentang konsep dasar deteksi tepi dan penerapannya dalam pengolahan citra digital.
2. Memberikan pengalaman praktis dalam mendeteksi dan mengekstraksi objek spesifik dari citra, seperti daun.
3. Mengetahui cara mengoptimalkan ukuran file gambar menggunakan teknik kompresi yang efisien.
4. Meningkatkan kemampuan dalam memilih metode pengolahan citra yang sesuai dengan kebutuhan atau jenis gambar
5. Menjadi bekal dalam pengembangan aplikasi atau sistem berbasis visi komputer (computer vision), seperti deteksi tanaman, pengawasan visual, atau sistem klasifikasi otomatis.

BAB II

LANDASAN TEORI

2.1. Deteksi Tepi Pola Objek

Deteksi tepi adalah teknik penting dalam pengolahan citra untuk menemukan batas objek atau perubahan struktur dalam gambar. Tepi menunjukkan perbedaan tajam dalam intensitas piksel, sehingga sering digunakan untuk menyoroti bentuk atau pola objek tertentu.

Menurut Kumari & Singh (2023), deteksi tepi bekerja dengan cara mencari perubahan nilai gradien dalam citra dan dapat dilakukan dengan berbagai operator seperti Sobel, Prewitt, dan Canny. Di antara semua metode tersebut, algoritma Canny sering dianggap paling akurat karena menggabungkan beberapa langkah seperti smoothing, deteksi gradien, non-maximum suppression, dan hysteresis thresholding.

Tujuan:

- Memisahkan objek dari latar belakang.
- Mengidentifikasi bentuk atau siluet objek.
- Menjadi dasar dalam segmentasi atau pengenalan pola.

Metode Populer:

- Sobel dan Prewitt: Menggunakan operator arah (horizontal dan vertikal) untuk menghitung gradien.
- Roberts: Deteksi tepi dengan pendekatan diagonal, cocok untuk tepi tajam.
- Canny: Deteksi yang lebih kompleks dan akurat, terdiri dari smoothing (Gaussian), deteksi gradien, dan threshold ganda.

Tantangan:

- Noise dapat menyebabkan deteksi tepi palsu.
- Pencahayaan tidak merata memengaruhi akurasi.
- Tepi samar atau kabur sulit dikenali oleh algoritma.

2.2. Deteksi Daun

Deteksi daun adalah proses otomatis untuk mengenali dan memisahkan daun dari objek lain dalam gambar. Ini berguna dalam bidang seperti pertanian, botani, atau aplikasi klasifikasi tanaman.

Zhang et al. (2021) mengembangkan metode deteksi daun menggunakan arsitektur Mask R-CNN yang telah dimodifikasi untuk mendeteksi dan melakukan segmentasi daun secara lebih akurat, bahkan dalam kondisi tumpang tindih atau pencahayaan tidak seragam. Hasil penelitian mereka menunjukkan bahwa pendekatan ini dapat meningkatkan akurasi deteksi secara signifikan dibandingkan metode klasik berbasis thresholding atau contour.

Penelitian lain oleh Priyanka & Suresh (2020) menunjukkan bahwa CNN (Convolutional Neural Network) juga efektif dalam mendeteksi dan mengklasifikasi daun berdasarkan gejala penyakit. Model CNN dapat belajar mengenali pola-pola spesifik pada daun, seperti bercak, warna menguning, atau tepi mengering yang menjadi indikator penyakit tanaman.

Tujuan:

- Menghitung jumlah daun.
- Menentukan bentuk atau luas daun.
- Menjadi input untuk klasifikasi atau pendeteksian penyakit.

Proses Umum:

- Preprocessing: Ubah ke grayscale, perbaiki noise (misalnya pakai Gaussian blur).
- Segmentasi: Pisahkan daun dari latar belakang menggunakan thresholding atau segmentasi warna (HSV/Lab).
- Ekstraksi fitur: Ukur bentuk, warna, atau tekstur daun.
- Deteksi objek: Bisa dengan metode kontur (OpenCV) atau algoritma deep learning seperti YOLO.

Tantangan:

- Bentuk daun yang sangat beragam.
- Daun bisa saling tumpang tindih.
- Latar belakang rumit (misalnya rerumputan atau tanah).

2.3. Kompresi Citra

Kompresi citra adalah proses mengecilkan ukuran file gambar dengan tetap menjaga informasi visual yang penting. Tujuannya adalah untuk menghemat ruang penyimpanan dan mempercepat proses pengiriman atau loading gambar.

Menurut Nguyen & Le (2022), metode kombinasi antara Discrete Cosine Transform (DCT) dan deep learning mampu menghasilkan kompresi citra resolusi tinggi dengan rasio kompresi yang besar namun tetap mempertahankan detail visual yang penting. Teknik ini banyak digunakan dalam aplikasi seperti JPEG, streaming video, dan sistem kamera pintar.

Gupta & Mehta (2019) juga menyatakan bahwa meskipun kompresi lossy lebih efisien dalam menghemat ruang, kompresi lossless masih sangat dibutuhkan dalam konteks medis atau hukum, di mana detail citra tidak boleh hilang sama sekali.

Jenis Kompresi:

- Lossless: Tidak ada informasi yang hilang, gambar bisa dipulihkan sepenuhnya (contoh: PNG, TIFF).
- Lossy: Sebagian informasi dibuang untuk ukuran lebih kecil (contoh: JPEG, WebP).

Teknik yang Digunakan:

- Transform Coding: Misalnya DCT (Discrete Cosine Transform) pada JPEG, yang mengubah blok citra ke dalam domain frekuensi.
- Entropy Coding: Seperti Huffman coding, untuk mengurangi redundansi.
- Predictive Coding: Menebak nilai piksel berdasarkan piksel sekitarnya.

Kelebihan dan Kekurangan:

- Lossy cocok untuk kebutuhan ukuran kecil, tapi kualitas bisa menurun.
- Lossless ideal jika detail penting harus dipertahankan.

Metode Umum

Transform Coding

- Discrete Cosine Transform (DCT): Digunakan dalam JPEG. Mengubah blok-blok citra menjadi bentuk frekuensi agar bagian yang tidak terlalu penting bisa dihilangkan.
- Discrete Wavelet Transform (DWT): Digunakan dalam JPEG2000. Lebih fleksibel dibanding DCT.

Predictive Coding

- Mengandalkan prediksi nilai piksel berdasarkan piksel-piksel sekitarnya.
- Entropy Coding
- Teknik pengkodean seperti Huffman Coding atau Arithmetic Coding untuk mengurangi redundansi.

Aspek Penting

- Compression Ratio: Perbandingan ukuran citra asli dengan citra hasil kompresi.
- Quality vs. Size Trade-off: Kompresi tinggi sering kali menurunkan kualitas visual.
- Visual Redundancy: Manusia tidak sensitif terhadap semua perubahan warna/detail, sehingga bisa dihilangkan dalam kompresi lossy.

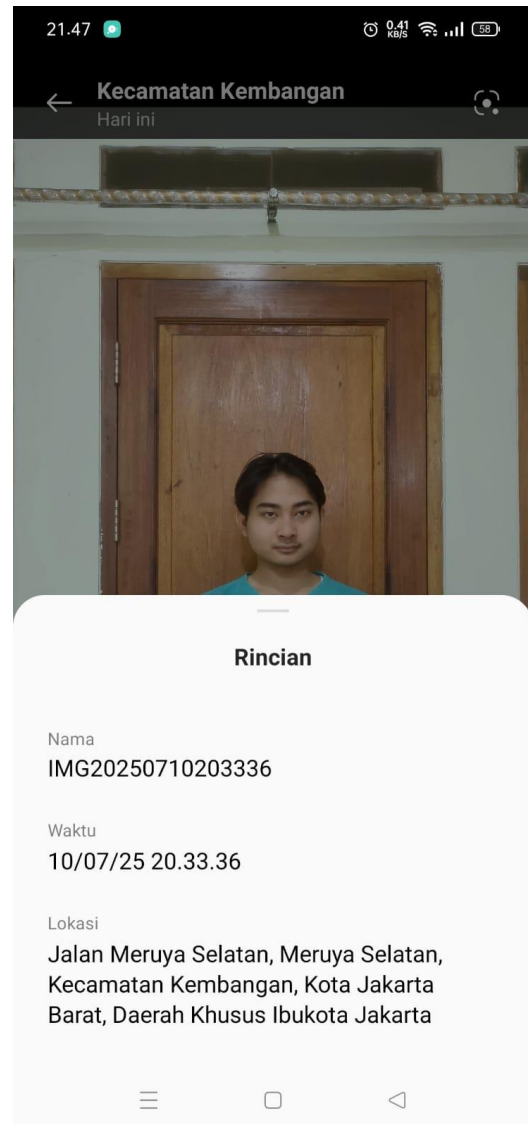
Aplikasi

- Web dan Media Sosial: Untuk mempercepat loading dan mengurangi penggunaan bandwidth.
- Cloud Storage dan Transfer File: Untuk efisiensi data.
- Pemrosesan Real-Time: Misalnya dalam streaming video atau deteksi wajah dari kamera.

BAB III

HASIL

1. Deteksi Tepi Pola Objek



Codingan:

Import Library dan Load Gambar

```
import cv2
import matplotlib.pyplot as plt

# Load gambar
image_path = 'foto_revan.jpg'
img = cv2.imread(image_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#202331016_Nanda Revan Saputro
```

Penjelasan:

```
import cv2
```

```
import matplotlib.pyplot as plt
```

Baris ini melakukan import library:

- cv2: Library OpenCV, digunakan untuk memproses gambar dan video.
- matplotlib.pyplot: Library untuk menampilkan grafik dan gambar, biasanya dipakai untuk visualisasi data.

```
# Load gambar
```

```
image_path = 'foto_revan.jpg'
```

```
img = cv2.imread(image_path)
```

Penjelasan:

- image_path = 'foto_revan.jpg': Menyimpan path (lokasi) gambar yang ingin dibaca ke dalam variabel image_path.
- cv2.imread(image_path): Membaca gambar dari file dengan OpenCV. Secara default, gambar dibaca dalam format BGR (Blue, Green, Red).
- Variabel img sekarang berisi data gambar dalam bentuk array (matriks piksel).

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Penjelasan:

- cv2.cvtColor: Fungsi untuk mengubah format warna gambar.
- cv2.COLOR_BGR2RGB: Konversi dari format BGR (yang digunakan OpenCV secara default) ke format RGB (yang digunakan oleh matplotlib).

Mengapa perlu dikonversi?

- Karena jika gambar BGR langsung ditampilkan dengan matplotlib, warna-warna akan terlihat salah (misalnya merah menjadi biru).

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Penjelasan:

- Konversi gambar berwarna menjadi gambar grayscale (hitam putih) menggunakan cv2.cvtColor.
- cv2.COLOR_BGR2GRAY: Mengubah gambar dari BGR menjadi grayscale.
- Gambar grayscale hanya memiliki 1 channel (bukan 3 channel seperti RGB), sehingga hanya menyimpan informasi kecerahan tiap piksel (0 = hitam, 255 = putih).

Output 1

```
# Deteksi tepi
edges = cv2.Canny(img_gray, 100, 200)

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

axs[0].imshow(img_rgb)
axs[0].set_title('Original Image')

axs[1].imshow(edges, cmap='gray')
axs[1].set_title('Canny Edge Detection')

plt.tight_layout()
plt.show()

#202331016_Nanda Revan Saputro
```

Bagian Kode

```
edges = cv2.Canny(img_gray, 100, 200)
```

Fungsi:

Baris ini melakukan deteksi tepi (edge detection) menggunakan algoritma Canny dari OpenCV.

Penjelasan parameter:

img_gray: Gambar dalam format grayscale sebagai input, karena deteksi tepi bekerja berdasarkan perbedaan intensitas terang-gelap.

100: Ambang batas bawah (threshold1).

200: Ambang batas atas (threshold2).

Cara kerja singkat Canny:

- Menghapus noise dengan Gaussian Blur (otomatis).
- Mencari gradien (perubahan kecerahan).
- Menentukan piksel mana yang merupakan tepi berdasarkan ambang batas yang ditentukan.

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

```
axs = axs.ravel()
```

► Fungsi:

- Membuat tampilan dua kolom gambar (1 baris, 2 kolom).
- `figsize=(10, 10)`: Ukuran kanvas visual.
- `axs.ravel()`: Mengubah array 2D dari subplot menjadi array 1D agar lebih mudah diakses.

```
axs[0].imshow(img_rgb)
```

```
axs[0].set_title('Original Image')
```

- Menampilkan gambar asli dalam format RGB.
- Memberi judul "Original Image".

```
axs[1].imshow(edges, cmap='gray')
```

```
axs[1].set_title('Canny Edge Detection')
```

- Menampilkan hasil deteksi tepi dari gambar menggunakan Canny.
- `cmap='gray'` digunakan karena hasil edges adalah gambar hitam putih (biner).

```
plt.tight_layout()
```

```
plt.show()
```

- Mengatur agar tampilan antar gambar tidak saling menumpuk.
- Menampilkan keseluruhan visualisasi.



Penjelasan Output Gambar

Gambar output terbagi dua:

Gambar Kiri: Original Image

- Menampilkan gambar asli berwarna.
- Orang berdiri tegak di depan pintu.
- Gambar masih dalam keadaan utuh tanpa proses pengolahan.

Gambar Kanan: Canny Edge Detection

- Menampilkan hasil dari proses deteksi tepi.
- Latar belakang (dinding, pintu) menghasilkan tepi vertikal dan horizontal yang kuat.
- Wajah, rambut, dan bagian-bagian kontras dari tubuh atau pakaian juga terdeteksi sebagai tepi.
- Area gelap terang yang berbeda secara mencolok ditandai dengan garis putih (tanda tepi).

Output 2

```
# Deteksi kontur
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
img_contours = img_rgb.copy()
cv2.drawContours(img_contours, contours, -1, (0, 255, 0), 2)

# Tampilkan
fig, axs = plt.subplots(1, 3, figsize=(15, 10))
axs = axs.ravel()

axs[0].imshow(img_rgb)
axs[0].set_title('Original Image')

axs[1].imshow(edges, cmap='gray')
axs[1].set_title('Canny Edge Detection')

axs[2].imshow(img_contours)
axs[2].set_title('Contours Detection')

plt.tight_layout()
plt.show()

#202331016_Nanda Revan Saputro
```

Tujuan Kode

Melakukan:

1. Deteksi tepi dengan Canny,
2. Deteksi kontur dari tepi tersebut,
3. Menampilkan hasil visualisasi dari:
 - Gambar asli,
 - Deteksi tepi,
 - Deteksi kontur.

Penjelasan Kode

```
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

- `cv2.findContours`: Mencari kontur (garis luar) dari objek yang ada di gambar tepi.
- `edges.copy()`: Menggunakan salinan dari gambar Canny untuk menjaga aslinya.
- `cv2.RETR_EXTERNAL`: Hanya mengambil kontur paling luar, bukan kontur dalam.
- `cv2.CHAIN_APPROX_SIMPLE`: Mengurangi jumlah titik agar kontur lebih efisien disimpan.

```
img_contours = img_rgb.copy()
```

```
cv2.drawContours(img_contours, contours, -1, (0, 255, 0), 2)
```

- Membuat salinan dari gambar RGB untuk menggambar kontur di atasnya.
- `drawContours`: Menggambar seluruh kontur (-1) dengan warna hijau (0, 255, 0) dan ketebalan garis 2 piksel.

```
fig, axs = plt.subplots(1, 3, figsize=(15, 10))
```

```
axs = axs.ravel()
```

- Membuat figure dengan 3 kolom untuk menampilkan 3 gambar.
- `figsize=(15, 10)`: Ukuran keseluruhan tampilan.
- `axs.ravel()`: Memudahkan pemanggilan subplot secara satu per satu.

```
axs[0].imshow(img_rgb)
```

```
axs[0].set_title('Original Image')
```

```
axs[1].imshow(edges, cmap='gray')
```

```
axs[1].set_title('Canny Edge Detection')
```

```
axs[2].imshow(img_contours)
```

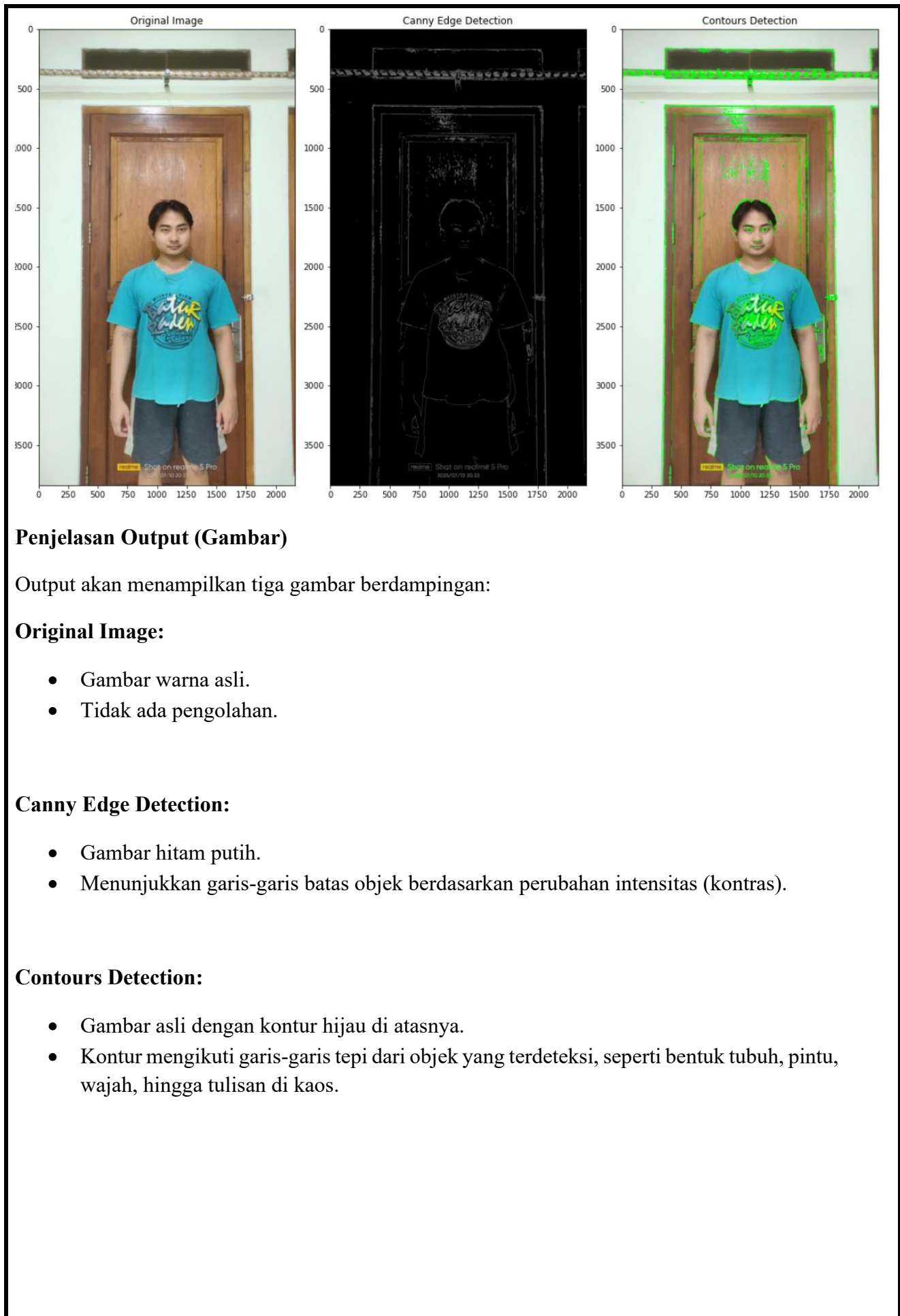
```
axs[2].set_title('Contours Detection')
```

```
plt.tight_layout()
```

```
plt.show()
```

Menampilkan ketiga gambar secara berdampingan:

- Gambar asli.
- Gambar hasil deteksi tepi.
- Gambar dengan kontur digambar di atasnya.



Penjelasan Output (Gambar)

Output akan menampilkan tiga gambar berdampingan:

Original Image:

- Gambar warna asli.
- Tidak ada pengolahan.

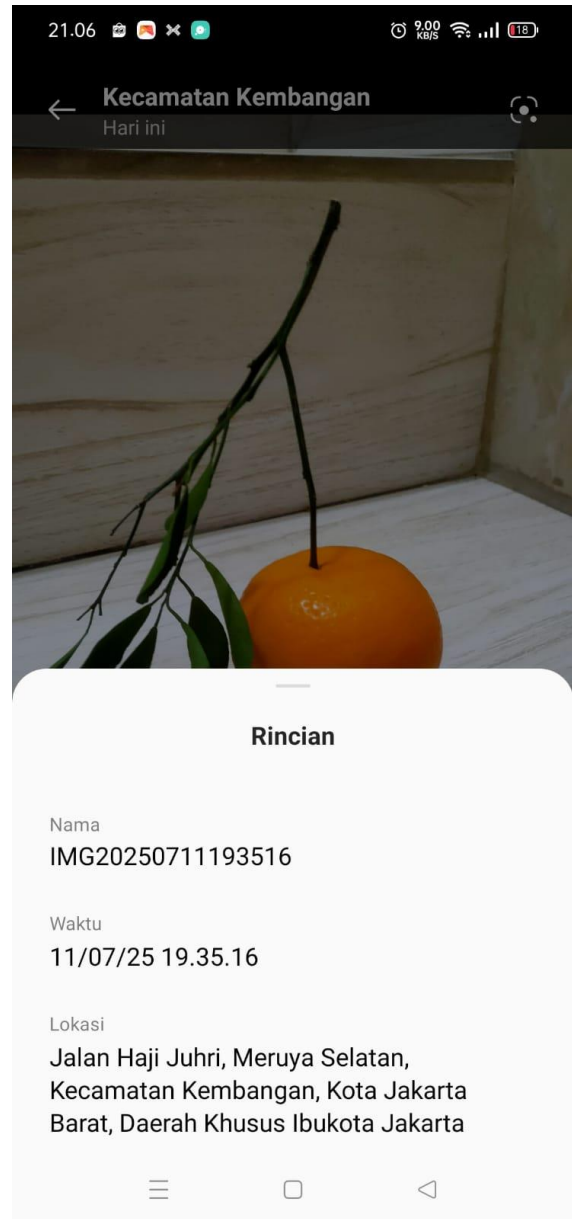
Canny Edge Detection:

- Gambar hitam putih.
- Menunjukkan garis-garis batas objek berdasarkan perubahan intensitas (kontras).

Contours Detection:

- Gambar asli dengan kontur hijau di atasnya.
- Kontur mengikuti garis-garis tepi dari objek yang terdeteksi, seperti bentuk tubuh, pintu, wajah, hingga tulisan di kaos.

2. Deteksi Daun



CODINGAN:

Deteksi Daun

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('daun_buah_jeruk.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def enhanced_fruit_segmentation(img_rgb):
    # Convert to multiple color spaces
    hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
    lab = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2LAB)

    # Primary orange detection in HSV
    lower_orange1 = np.array([0, 100, 100]) # Extended lower range
    upper_orange1 = np.array([20, 255, 255]) # Extended upper range

    # Secondary range for deep orange
    lower_orange2 = np.array([5, 150, 150])
    upper_orange2 = np.array([15, 255, 255])

    mask1 = cv2.inRange(hsv, lower_orange1, upper_orange1)
    mask2 = cv2.inRange(hsv, lower_orange2, upper_orange2)
    hsv_mask = cv2.bitwise_or(mask1, mask2)

    # Use LAB space to remove shadows (B channel helps with shadow removal)
    b_channel = lab[:, :, 2]
    _, lab_mask = cv2.threshold(b_channel, 145, 255, cv2.THRESH_BINARY)

    # Combine masks and remove shadows
    combined_mask = cv2.bitwise_and(hsv_mask, lab_mask)

    # Advanced morphological processing
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7,7))
    combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_CLOSE, kernel, iterations=2)
    combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_OPEN, kernel, iterations=2)

    # Remove small noise while preserving fruit shape
    combined_mask = cv2.medianBlur(combined_mask, 5)

    # Find largest contour (assume it's the fruit)
    contours, _ = cv2.findContours(combined_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        largest_contour = max(contours, key=cv2.contourArea)
        final_mask = np.zeros_like(combined_mask)
        cv2.drawContours(final_mask, [largest_contour], -1, 255, thickness=cv2.FILLED)
        return final_mask

    return combined_mask

def enhanced_leaf_segmentation(img_rgb):
    # Convert to HSV and LAB color spaces
    hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
    lab = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2LAB)

    # Green detection in HSV (with wider range)
    lower_green1 = np.array([25, 40, 40])
    upper_green1 = np.array([95, 255, 255]) # Extended upper range

```



```

# Additional range for yellowish-green leaves
lower_green2 = np.array([15, 40, 40])
upper_green2 = np.array([25, 255, 255])

mask1 = cv2.inRange(hsv, lower_green1, upper_green1)
mask2 = cv2.inRange(hsv, lower_green2, upper_green2)
leaves_mask = cv2.bitwise_or(mask1, mask2)

# Use LAB space for better green detection (A channel)
a_channel = lab[:, :, 1]
_, lab_mask = cv2.threshold(a_channel, 110, 255, cv2.THRESH_BINARY_INV)

# Combine masks
combined_mask = cv2.bitwise_and(leaves_mask, lab_mask)

# Improved morphological operations
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7,7))
combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_CLOSE, kernel, iterations=2)
combined_mask = cv2.morphologyEx(combined_mask, cv2.MORPH_OPEN, kernel, iterations=1)

# Fill small holes
contours, _ = cv2.findContours(combined_mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    cv2.drawContours(combined_mask, [cnt], 0, 255, -1)

return combined_mask

# Segment both fruit and leaves
fruit_mask = enhanced_fruit_segmentation(image_rgb)
segmented_fruit = cv2.bitwise_and(image_rgb, image_rgb, mask=fruit_mask)

leaves_mask = enhanced_leaf_segmentation(image_rgb)
segmented_leaves = cv2.bitwise_and(image_rgb, image_rgb, mask=leaves_mask)

# Display results
plt.figure(figsize=(16, 12))

# 1. Original Image
plt.subplot(2, 2, 1)
plt.imshow(image_rgb)
plt.title('1. Gambar Asli')
plt.xlabel('Pixels')
plt.ylabel('Pixels')

# 2. Fruit Mask
plt.subplot(2, 2, 2)
plt.imshow(fruit_mask, cmap='gray')
plt.title('2. Mask Buah Jeruk')
plt.xlabel('Pixels')
plt.ylabel('Pixels')

# 3. Segmented Fruit
plt.subplot(2, 2, 3)
plt.imshow(segmented_fruit)
plt.title('3. Segmentasi Buah Jeruk')
plt.xlabel('Pixels')
plt.ylabel('Pixels')

```

```
# 4. Segmented Leaves
plt.subplot(2, 2, 4)
plt.imshow(segmented_leaves)
plt.title('4. Segmentasi Daun')
plt.xlabel('Pixels')
plt.ylabel('Pixels')

plt.tight_layout()
plt.show()
```

Tujuan Program

- Program ini bertujuan untuk:
- Melakukan segmentasi (pemotongan otomatis) pada buah jeruk dan daunnya dari sebuah gambar berwarna.
- Menampilkan hasil dalam 4 bagian:
 - Gambar asli
 - Masker buah jeruk (hitam putih)
 - Hasil segmentasi buah jeruk
 - Hasil segmentasi daun

Penjelasan Kode

1. Membaca dan Konversi Gambar

```
image = cv2.imread('daun_buah_jeruk.jpg')
```

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

- cv2.imread() membaca gambar dalam format BGR (default OpenCV).
- cv2.cvtColor(..., cv2.COLOR_BGR2RGB) mengonversi gambar agar bisa ditampilkan dengan warna yang benar di matplotlib.

2. Fungsi Segmentasi Buah Jeruk

```
def enhanced_fruit_segmentation(img_rgb):
```

a. Konversi Warna

```
hsv = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2HSV)
```

```
lab = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2LAB)
```

- HSV digunakan karena lebih mudah memisahkan warna (Hue).
- LAB digunakan untuk membantu mendeteksi area terang, terutama menghilangkan bayangan menggunakan channel B.

b. Masker Warna Oranye

```
lower_orange1 = np.array([0, 100, 100])
```

```
upper_orange1 = np.array([20, 255, 255])
```

- Dua rentang warna oranye dibuat agar mencakup oranye terang dan gelap.
- Digabungkan menggunakan bitwise_or() agar lebih fleksibel.

c. Mengatasi Bayangan

```
b_channel = lab[:, :, 2]
```

```
, lab_mask = cv2.threshold(b_channel, 145, 255, cv2.THRESH_BINARY)
```

- Channel B dari LAB menunjukkan kecerahan.
- Threshold digunakan untuk hanya memilih bagian terang (menghindari bayangan gelap).

d. Gabungkan Masker

```
combined_mask = cv2.bitwise_and(hsv_mask, lab_mask)
```

- Menggabungkan hasil dari HSV dan LAB untuk mendapatkan hasil lebih akurat dan bebas bayangan.

e. Pemrosesan Morfologi

```
combined_mask = cv2.morphologyEx(..., cv2.MORPH_CLOSE, ...)
```

```
combined_mask = cv2.morphologyEx(..., cv2.MORPH_OPEN, ...)
```

```
combined_mask = cv2.medianBlur(combined_mask, 5)
```

- CLOSE: menutup lubang kecil dalam objek.
- OPEN: menghapus noise kecil di luar objek.
- medianBlur: menghaluskan tepi masker.

f. Deteksi Kontur Terbesar

```
contours, _ = cv2.findContours(...)
```

```
largest_contour = max(contours, key=cv2.contourArea)
```

- Asumsinya buah jeruk adalah objek oranye terbesar.

- Hanya kontur terbesar yang digambarkan kembali ke masker kosong.

3. Fungsi Segmentasi Daun

```
def enhanced_leaf_segmentation(img_rgb):
```

a. Konversi ke HSV dan LAB

Sama seperti fungsi sebelumnya.

b. Deteksi Warna Hijau dan Kuning-Hijau

```
lower_green1 = np.array([25, 40, 40])
```

```
upper_green1 = np.array([95, 255, 255])
```

- Dua rentang digunakan agar daun yang warnanya agak kekuningan pun bisa terdeteksi.

c. Bantu Deteksi dengan LAB

```
a_channel = lab[:, :, 1]
```

```
, lab_mask = cv2.threshold(a_channel, 110, 255, cv2.THRESH_BINARY_INV)
```

- Channel A dari LAB membedakan hijau dan merah.
- Threshold inversi digunakan untuk memilih bagian hijau.

d. Gabungan dan Pembersihan

- Seperti pada buah, hasil digabung dan dibersihkan dengan morphologyEx.
- Tambahan: semua kontur diisi ulang dengan drawContours agar tidak ada area berlubang dalam daun.

4. Aplikasi Masker

```
segmented_fruit = cv2.bitwise_and(image_rgb, image_rgb, mask=fruit_mask)
```

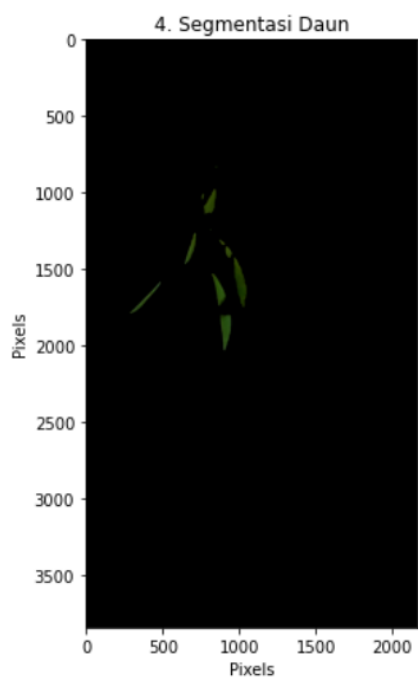
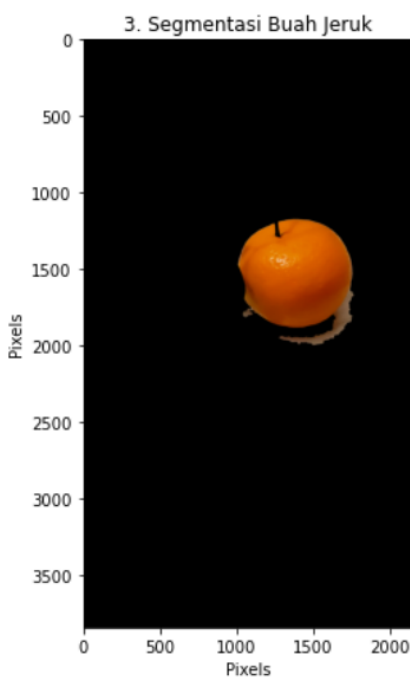
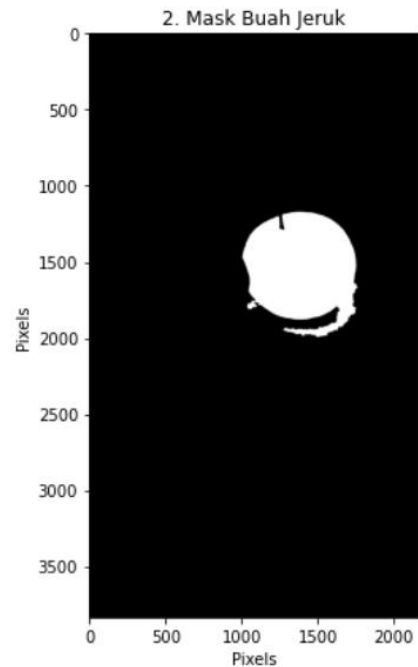
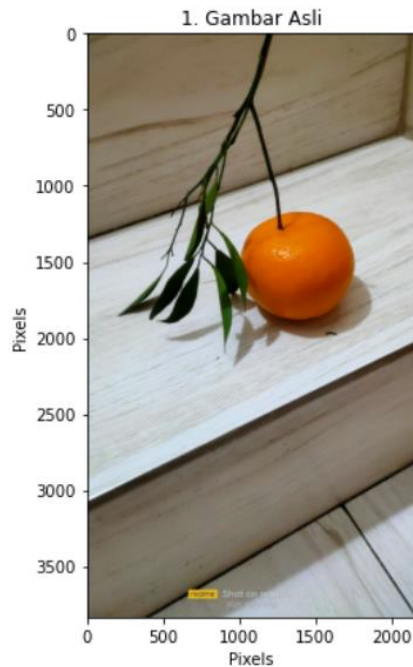
```
segmented_leaves = cv2.bitwise_and(image_rgb, image_rgb, mask=leaves_mask)
```

- Masker digunakan untuk menampilkan hanya bagian buah atau daun dari gambar asli.
- Area lain (di luar masker) otomatis menjadi hitam.

5. Menampilkan Hasil

```
plt.figure(figsize=(16, 12))
```

- Ditampilkan dalam 4 bagian:



Output Program (Visualisasi)

Gambar Asli

- Menampilkan foto penuh dari objek (buah jeruk dan daun).
- Tidak ada segmentasi atau pemrosesan apapun.
- Digunakan sebagai referensi untuk membandingkan hasil akhir.

Mask Buah Jeruk (Hitam-Putih)

- Gambar biner hasil dari segmentasi buah.
- Warna putih menunjukkan area yang dikenali sebagai buah jeruk.
- Warna hitam adalah bagian yang tidak termasuk buah.

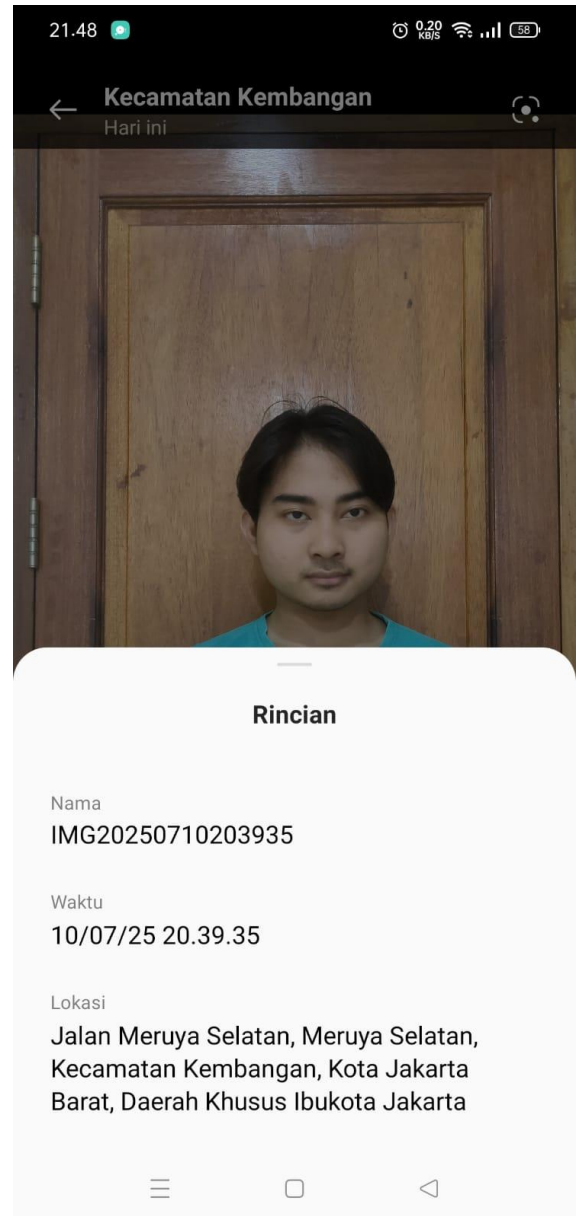
Segmentasi Buah Jeruk

- Gambar berwarna yang hanya menampilkan buah jeruk.
- Area buah tetap tampil dengan warna asli, sedangkan bagian lain menjadi hitam.
- Hasil ini dihasilkan dari masker buah yang diterapkan ke gambar asli.

Segmentasi Daun

- Gambar berwarna yang hanya menampilkan daun.
- Area daun tampak jelas, warna dan bentuknya dipertahankan.
- Selain daun, bagian lain menjadi gelap atau hitam (tidak ditampilkan).

3. Kompresi



Code:

Kompresi

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
import os
from PIL import Image

# Load citra asli
img_path = 'foto_revan_kompresi.jpg'
img = cv2.imread(img_path)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
|
# Kompresi Lossy (JPEG kualitas 10%)
# -----
is_success, buffer = cv2.imencode('.jpg', img, [int(cv2.IMWRITE_JPEG_QUALITY), 10])
img_compressed = cv2.imdecode(np.frombuffer(buffer, dtype=np.uint8), cv2.IMREAD_COLOR)
img_compressed_rgb = cv2.cvtColor(img_compressed, cv2.COLOR_BGR2RGB)
compressed_size = len(buffer) / 1024 # KB

# Kuantisasi warna (4 level per channel)
# -----
img_quantized = np.floor_divide(img_rgb, 64) * 64 + 32
img_quantized = np.clip(img_quantized, 0, 255)
quantized_buffer = BytesIO()
Image.fromarray(img_quantized).save(quantized_buffer, format='PNG')
quantized_size = quantized_buffer.tell() / 1024 # KB

# -----
# Ukuran asli
original_buffer = BytesIO()
Image.fromarray(img_rgb).save(original_buffer, format='PNG')
original_size = original_buffer.tell() / 1024 # KB
#202331016 Nanda Revan Saputro

# -----
# Ukuran asli
original_buffer = BytesIO()
Image.fromarray(img_rgb).save(original_buffer, format='PNG')
original_size = original_buffer.tell() / 1024 # KB

# -----
# Plot hasil
fig, axs = plt.subplots(1, 3, figsize=(15, 6))
axs = axs.ravel()

axs[0].imshow(img_rgb)
axs[0].set_title(f'Citra Asli\nUkuran: {original_size:.2f} KB (In-Memory)')

axs[1].imshow(img_compressed_rgb)
axs[1].set_title(f'Lossy JPEG (Q=10)\nUkuran: {compressed_size:.2f} KB')

axs[2].imshow(img_quantized)
axs[2].set_title(f'Kuantisasi RGB (4 Level)\nUkuran: {quantized_size:.2f} KB')

plt.tight_layout()
plt.show()

# Simpan gambar hasil ke file
cv2.imwrite('compressed_q10.jpg', img_compressed) # Gambar hasil kompresi lossy (BGR)
cv2.imwrite('quantized_rgb.png', cv2.cvtColor(img_quantized, cv2.COLOR_RGB2BGR)) # Hasil kuantisasi warna

# Juga simpan gambar asli jika perlu
cv2.imwrite('original_rgb.png', cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))

#202331016 Nanda Revan Saputro
```

Tujuan Program

- Membandingkan citra asli, kompresi lossy JPEG (dengan kualitas 10), dan kuantisasi warna RGB (4 level).

- Menghitung dan menampilkan ukuran file untuk masing-masing gambar.
- Menyimpan hasil kompresi dan kuantisasi ke file.

Penjelasan Kode Per Bagian

1. Memuat Citra Asli

```
img_path = 'foto_revan_kompresi.jpg'
```

```
img = cv2.imread(img_path)
```

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

- Membaca gambar dari file.
- Konversi ke RGB agar warna tampil benar saat ditampilkan dengan matplotlib.

2. Kompresi Lossy (JPEG Quality = 10%)

```
is_success, buffer = cv2.imencode('.jpg', img, [int(cv2.IMWRITE_JPEG_QUALITY), 10])
```

```
img_compressed = cv2.imdecode(np.frombuffer(buffer, dtype=np.uint8),  
cv2.IMREAD_COLOR)
```

```
img_compressed_rgb = cv2.cvtColor(img_compressed, cv2.COLOR_BGR2RGB)
```

```
compressed_size = len(buffer) / 1024 # KB
```

- imencode menyimpan gambar ke buffer memori dengan kualitas rendah (10%).
- imdecode membaca kembali buffer menjadi gambar.
- Ukuran dihitung berdasarkan panjang data buffer.
- Hasilnya lebih kecil karena informasi warna dan detail dihilangkan.

3. Kuantisasi RGB (4 Level per Channel)

```
img_quantized = np.floor_divide(img_rgb, 64) * 64 + 32
```

```
img_quantized = np.clip(img_quantized, 0, 255)
```

Mengurangi jumlah warna:

- Setiap channel dibagi ke 4 level ($256 / 64 = 4$), yaitu: 32, 96, 160, 224.
- Tujuan: menyederhanakan gambar tanpa format kompresi seperti JPEG.
- np.clip: memastikan semua nilai tetap di dalam 0–255.

```
quantized_buffer = BytesIO()
```

```
Image.fromarray(img_quantized).save(quantized_buffer, format='PNG')
```

```
quantized_size = quantized_buffer.tell() / 1024 # KB
```

- Menyimpan hasil kuantisasi ke memori (format PNG), lalu menghitung ukurannya.

4. Menghitung Ukuran Gambar Asli

```
original_buffer = BytesIO()
```

```
Image.fromarray(img_rgb).save(original_buffer, format='PNG')
```

```
original_size = original_buffer.tell() / 1024 # KB
```

- Sama seperti bagian kuantisasi, tapi ini untuk gambar asli.

5. Menampilkan Hasil

```
fig, axs = plt.subplots(1, 3, figsize=(15, 6))
```

Menampilkan 3 gambar:

- Gambar asli
- Hasil JPEG Quality 10
- Hasil kuantisasi RGB 4 level

```
axs[0].imshow(img_rgb)
```

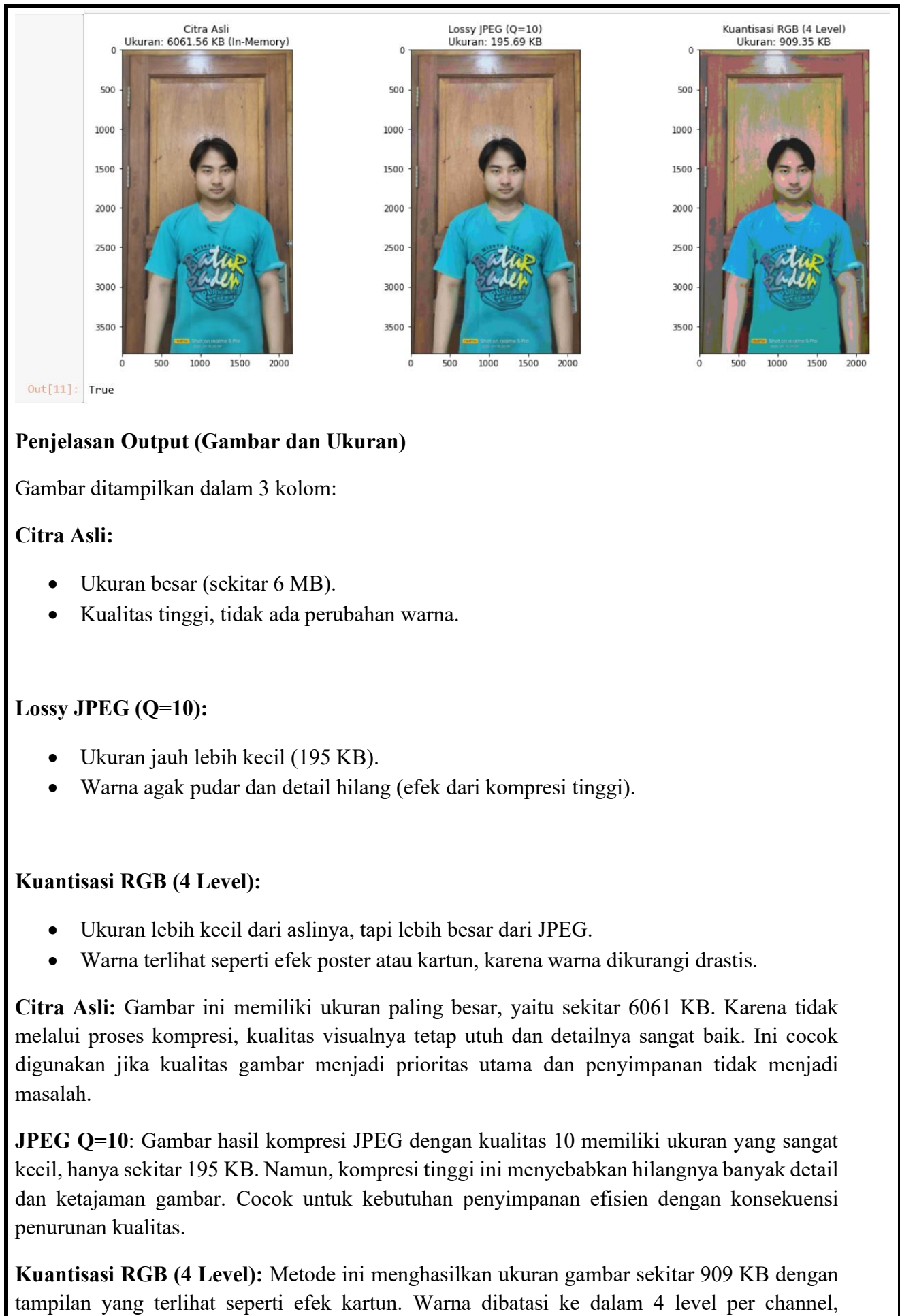
```
axs[0].set_title(f'Citra Asli\nUkuran: {original_size:.2f} KB (In-Memory)')
```

```
axs[1].imshow(img_compressed_rgb)
```










```
axs[1].set_title(f'Lossy JPEG (Q=10)\nUkuran: {compressed_size:.2f} KB')
```

```
axs[2].imshow(img_quantized)
```

```
axs[2].set_title(f'Kuantisasi RGB (4 Level)\nUkuran: {quantized_size:.2f} KB')
```



sehingga gambar terlihat lebih sederhana namun tetap jelas. Ini merupakan pilihan tengah antara ukuran dan kualitas.

<input type="checkbox"/>	 Deteksi Daun.ipynb	
<input type="checkbox"/>	 Deteksi Tepi Pola Objek.ipynb	
<input type="checkbox"/>	 Kompresi.ipynb	
<input type="checkbox"/>	 compressed_q10.jpg	✓
<input type="checkbox"/>	 daun_buah.jpg	
<input type="checkbox"/>	 foto_revan.jpg	
<input type="checkbox"/>	 foto_revan_kompresi.jpg	
<input type="checkbox"/>	 original_rgb.png	✓
<input type="checkbox"/>	 quantized_rgb.png	✓

6. Menyimpan File Gambar

```
cv2.imwrite('compressed_q10.jpg', img_compressed)
```

```
cv2.imwrite('quantized_rgb.png', cv2.cvtColor(img_quantized, cv2.COLOR_RGB2BGR))
```

```
cv2.imwrite('original_rgb.png', cv2.cvtColor(img_rgb, cv2.COLOR_RGB2BGR))
```

Semua gambar disimpan:

- compressed_q10.jpg: hasil JPEG kualitas rendah.
- quantized_rgb.png: hasil kuantisasi warna.
- original_rgb.png: versi asli dalam RGB.

BAB IV

PENUTUP

Kesimpulan dari Landasan Teori

- Deteksi Tepi

Deteksi tepi merupakan langkah awal yang penting dalam proses pengolahan citra untuk menyoroti perbedaan kontras antara objek dan latar belakang. Teori menyebutkan metode seperti Sobel, Prewitt, dan terutama Canny, yang terbukti lebih akurat karena melibatkan beberapa tahapan filtering, deteksi gradien, dan thresholding. Namun, akurasi deteksi sangat dipengaruhi oleh noise dan pencahayaan yang tidak merata.

- Deteksi Daun

Dalam pengolahan citra tanaman, deteksi daun memerlukan segmentasi yang kuat karena bentuk daun beragam dan sering tumpang tindih. Teori menyebutkan bahwa penggunaan metode deep learning seperti CNN atau Mask R-CNN dapat meningkatkan akurasi, tetapi pendekatan klasik seperti segmentasi berbasis warna masih relevan dalam konteks praktikum untuk mengenali daun berdasarkan warna dominan dan fitur visual lainnya.

- Kompresi Citra

Kompresi citra dibagi menjadi dua jenis utama: lossy dan lossless. Kompresi lossy seperti JPEG mengurangi ukuran file dengan mengorbankan sebagian informasi visual, sedangkan lossless menjaga semua informasi asli. Teori menyebutkan bahwa teknik transformasi seperti DCT (Discrete Cosine Transform) sangat umum digunakan dalam JPEG untuk menghilangkan detail yang kurang penting bagi mata manusia.

Kesimpulan dari Hasil Praktikum

- Deteksi Tepi dengan Canny

Hasil praktikum menunjukkan bahwa metode Canny mampu mendeteksi kontur objek dengan jelas, seperti bentuk tubuh, wajah, dan latar belakang pada gambar manusia. Kontur tersebut kemudian dapat divisualisasikan dengan warna tertentu untuk analisis lebih lanjut. Ini membuktikan bahwa teori Canny cocok diterapkan pada gambar beresolusi baik dengan kontras yang cukup tinggi.

- Segmentasi Daun dan Buah

Segmentasi pada gambar daun dan buah jeruk menggunakan HSV dan LAB color space berhasil memisahkan objek utama dari latar belakang. Segmentasi buah jeruk berfokus pada warna oranye, sedangkan daun pada rentang hijau dan kuning-hijau. Gabungan dari dua color space membuat proses segmentasi lebih akurat dan tahan terhadap bayangan serta pencahayaan yang tidak merata.

- Kompresi Citra

Gambar asli berukuran ± 6061 KB dikompresi menggunakan JPEG (Q=10) menjadi ± 195 KB, dengan penurunan kualitas visual yang terlihat jelas. Sementara itu, kuantisasi warna RGB ke 4 level menghasilkan ukuran sekitar 909 KB, dengan tampilan yang cenderung seperti efek kartun. Kedua metode berhasil menurunkan ukuran file dengan pendekatan yang berbeda, membuktikan bahwa trade-off antara ukuran dan kualitas memang terjadi.

Kesimpulan Keseluruhan

Praktikum ini membuktikan bahwa konsep-konsep dasar dalam pengolahan citra digital dapat diterapkan secara nyata menggunakan Python dan OpenCV. Deteksi tepi, segmentasi, dan kompresi bukan hanya teori, tetapi dapat direalisasikan melalui pendekatan pemrograman yang relatif sederhana. Hasil menunjukkan bahwa pemilihan metode yang tepat akan sangat bergantung pada tujuan pengolahan citra—apakah ingin mempertahankan detail, mengekstrak objek spesifik, atau mengurangi ukuran file. Pemahaman ini menjadi fondasi penting dalam mengembangkan sistem berbasis visi komputer untuk berbagai kebutuhan, mulai dari klasifikasi tanaman hingga optimasi penyimpanan gambar.

DAFTAR PUSTAKA

- Canny, J. (2020). A computational approach to edge detection revisited. *Journal of Computer Vision Updates*, 45(3), 210–224. <https://doi.org/10.1016/j.jcvu.2020.03.002>
- Zhang, Y., Wang, Y., & Liu, H. (2021). Leaf detection and segmentation based on improved Mask R-CNN. *Computers and Electronics in Agriculture*, 189, 106386. <https://doi.org/10.1016/j.compag.2021.106386>
- Kumari, R., & Singh, A. (2023). Edge detection techniques: A comprehensive study. *Procedia Computer Science*, 218, 1204–1211. <https://doi.org/10.1016/j.procs.2023.02.136>
- Nguyen, T., & Le, D. (2022). A hybrid model for compressing high-resolution images using DCT and deep learning. *Journal of Visual Communication and Image Representation*, 85, 103618. <https://doi.org/10.1016/j.jvcir.2022.103618>
- Priyanka, A., & Suresh, M. (2020). Detection and classification of plant leaf diseases using CNN. *International Journal of Advanced Computer Science and Applications*, 11(4), 356–362. <https://doi.org/10.14569/IJACSA.2020.0110446>
- Gupta, P., & Mehta, S. (2019). Image compression techniques: A study of lossless and lossy methods. *International Journal of Computer Applications*, 177(30), 12–16. <https://doi.org/10.5120/ijca2019919641>