UAS

PENGOLAHAN CITRA

NAMA : Michael Christia Putra

NIM : 202331203

KELAS : A

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

NO.PC : 18

ASISTEN : 1. Clarenca Sweetdiva Pereira

2. Viana Salsabila Fairuz Syahla

3. Kashrina Masyid Azka

4. Sasikirana Ramadhanty Setiawan Putri

INSTITUT TEKNOLOGI PLN

TEKNIK INFORMATIKA

2024/2025

DAFTAR ISI

BAB I
PENDAHULUAN

Pada bab ini, saya akan menguraikan latar belakang dari proyek pengolahan citra digital yang saya kerjakan. Proyek ini berfokus pada tiga pilar utama dalam pengolahan citra, yaitu deteksi tepi, transformasi geometri, dan kompresi citra. Seluruh proses ini saya implementasikan menggunakan bahasa pemrograman Python dengan memanfaatkan beberapa library yang relevan. Keunikan dari proyek ini adalah penggunaan citra personal yang diambil langsung dari kamera pribadi, di mana setiap citra harus memenuhi kriteria spesifik untuk setiap jenis pengolahan yang dilakukan. Bab ini akan merinci rumusan masalah yang menjadi dasar penelitian, tujuan yang ingin saya capai, serta manfaat yang diharapkan dari penyusunan laporan ini.

1.1 Rumusan Masalah

Untuk memandu jalannya proyek dan penulisan laporan ini, saya telah merumuskan beberapa pertanyaan utama yang akan dijawab melalui implementasi dan analisis. Rumusan masalah ini mencakup aspek teknis, implementatif, dan evaluatif dari setiap tugas yang dikerjakan.

Berdasarkan uraian di atas, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan algoritma *Canny Edge Detection* dan *Contours Detection* menggunakan Python untuk mengidentifikasi pola tepi pada citra foto diri yang diambil dari jarak minimal dua meter?
2. Bagaimana teknik-teknik transformasi geometri—meliputi rotasi, pengubahan ukuran, pemotongan, pembalikan, dan translasi—dapat diterapkan secara efektif pada citra foto diri menggunakan library Python?
3. Bagaimana metode kompresi *lossy* dengan kualitas 10% dan metode kuantisasi warna ke dalam 4 level per kanal RGB dapat diimplementasikan untuk mengurangi ukuran data citra?
4. Apa saja library Python, seperti OpenCV dan Pillow, yang paling efektif untuk digunakan dalam setiap proses pengolahan citra tersebut, dan bagaimana cara konfigurasinya untuk mencapai hasil yang diinginkan?
5. Apa perbedaan visual dan struktural yang paling signifikan antara citra asli dengan citra hasil olahan dari setiap proses (deteksi tepi, geometri, dan kompresi)?
6. Bagaimana cara memvalidasi bahwa citra yang digunakan sebagai input telah memenuhi semua ketentuan proyek, termasuk bukti pengambilan gambar dan lokasi, untuk memastikan objektivitas hasil?

1.2 Tujuan Masalah

Selaras dengan rumusan masalah yang telah dijabarkan, penelitian dan implementasi proyek ini memiliki beberapa tujuan spesifik yang ingin saya capai. Adapun tujuan dari penelitian ini adalah sebagai berikut:

1. Mengimplementasikan dan menampilkan hasil dari algoritma deteksi tepi Canny dan deteksi kontur pada citra foto diri untuk menunjukkan pola objek secara jelas.

2. Menerapkan dan memvisualisasikan berbagai teknik transformasi geometri pada citra foto diri, termasuk rotasi, pengubahan ukuran, pemotongan, pembalikan, dan translasi.
3. Mengaplikasikan metode kompresi lossy dan kuantisasi warna pada citra asli untuk menunjukkan perbandingan hasil kompresi secara visual dan kualitatif.
4. Mengidentifikasi dan memanfaatkan library Python yang relevan untuk membangun program yang mampu menjalankan ketiga fungsi pengolahan citra sesuai ketentuan.
5. Menganalisis dan mendeskripsikan perbedaan antara citra asli dan citra hasil olahan untuk memahami dampak dari setiap algoritma yang diterapkan.
6. Menyajikan bukti validitas citra yang digunakan, sesuai dengan ketentuan proyek, sebagai bagian dari metodologi penelitian yang transparan.

1.3 Manfaat Masalah

Penyusunan laporan proyek ini diharapkan dapat memberikan manfaat yang luas, tidak hanya bagi saya sebagai penulis, tetapi juga bagi lingkungan akademis dan masyarakat umum yang tertarik pada bidang pengolahan citra digital.

1.3.1 Manfaat bagi Penulis

Bagi saya pribadi, pengerjaan proyek dan penyusunan laporan ini memberikan manfaat praktis yang sangat besar. Saya mendapatkan pengalaman langsung dalam mengimplementasikan konsep-konsep teoretis pengolahan citra ke dalam kode program yang fungsional menggunakan Python. Selain itu, saya menjadi lebih terampil dalam menggunakan library-library penting seperti OpenCV dan Matplotlib, serta memahami alur kerja sebuah proyek teknologi mulai dari analisis kebutuhan, implementasi, hingga evaluasi hasil. Kemampuan analisis saya juga terasah melalui proses perbandingan antara citra asli dan citra hasil olahan.

1.3.2 Manfaat bagi Akademisi

Laporan ini diharapkan dapat menjadi sumber referensi yang relevan bagi mahasiswa lain, khususnya di lingkungan institut, yang sedang atau akan mengambil mata kuliah Pengolahan Citra Digital. Laporan ini menyajikan contoh konkret penerapan berbagai teknik dasar pengolahan citra dengan studi kasus yang jelas dan kode yang dapat dipelajari. Bagi dosen atau pengajar, laporan ini dapat menjadi salah satu bahan studi kasus untuk menunjukkan bagaimana teori yang diajarkan di kelas dapat diaplikasikan untuk menyelesaikan masalah praktis.

1.3.3 Manfaat bagi Masyarakat Umum

Meskipun bersifat teknis, hasil dari penelitian ini memiliki relevansi dengan teknologi yang digunakan oleh masyarakat luas setiap hari. Konsep deteksi tepi adalah dasar dari fitur-fitur seperti filter di media sosial atau pemindai QR Code. Transformasi geometri adalah apa yang terjadi saat kita memotong (crop) atau memutar (rotate) foto di galeri ponsel. Sementara itu, kompresi citra adalah teknologi krusial yang memungkinkan kita mengirim gambar dengan cepat melalui aplikasi pesan atau mengunggahnya ke internet tanpa memakan banyak kuota. Laporan ini dapat memberikan wawasan kepada masyarakat tentang cara kerja di balik teknologi visual yang sering mereka gunakan.

BAB II
LANDASAN TEORI

Pada bab ini, saya akan memaparkan dasar-dasar teoretis yang menjadi fondasi dalam pengerjaan proyek pengolahan citra digital ini. Pembahasan akan mencakup konsep fundamental pengolahan citra, representasi citra digital, hingga penjelasan mendalam mengenai tiga teknik utama yang saya implementasikan: deteksi tepi, transformasi geometri, dan kompresi citra. Seluruh pembahasan ini didukung oleh tinjauan dari berbagai studi dan literatur relevan untuk memberikan konteks akademis yang kuat.

2.1 Pengolahan Citra Digital dan Representasinya

Pengolahan citra digital merupakan sebuah disiplin ilmu yang mempelajari cara memanipulasi atau menganalisis citra menggunakan komputer. Menurut Supiyandi dkk. [1], proses ini bertujuan untuk meningkatkan kualitas visual gambar, mengekstraksi informasi penting, atau mempersiapkannya untuk analisis lebih lanjut. Secara umum, proses ini melibatkan beberapa tahapan, mulai dari akuisisi (pengambilan gambar), pra-pemrosesan (perbaikan awal), analisis (ekstraksi fitur), hingga interpretasi hasil [2].

Di dalam komputer, citra direpresentasikan sebagai sebuah matriks dua dimensi (atau tiga dimensi untuk citra berwarna) yang terdiri dari elemen-elemen kecil yang disebut piksel. Setiap piksel memiliki nilai intensitas yang menentukan kecerahannya. Untuk citra *grayscale*, setiap piksel memiliki satu nilai (biasanya 0 hingga 255). Sementara itu, pada model warna RGB (*Red, Green, Blue*), setiap piksel direpresentasikan oleh tiga nilai intensitas yang masing-masing mewakili kanal warna merah, hijau, dan biru [8]. Kombinasi dari ketiga kanal ini menghasilkan spektrum warna yang luas yang kita lihat pada layar. Citra ini kemudian disimpan dalam berbagai format file, seperti JPEG (umum untuk foto karena kompresinya yang efisien) dan PNG (mendukung transparansi dan kompresi *lossless*).

2.2 Deteksi Tepi (Edge Detection)

Deteksi tepi adalah salah satu teknik paling fundamental dalam pengolahan citra yang bertujuan untuk mengidentifikasi batas-batas atau kontur objek di dalam sebuah gambar. Tepi didefinisikan sebagai area di mana terjadi perubahan intensitas piksel yang drastis [5]. Teknik ini sangat krusial untuk tugas-tugas seperti segmentasi citra, pengenalan objek, dan pelacakan gerak [9].

Terdapat berbagai metode untuk mendeteksi tepi, seperti operator Sobel, Prewitt, dan Canny. Dalam proyek ini, saya memilih untuk menggunakan metode Canny karena keunggulannya dalam menghasilkan tepi yang tipis, kontinu, dan memiliki tingkat kesalahan yang rendah, terutama pada citra yang mengandung *noise* atau derau [5]. Implementasi algoritma Canny pada platform seperti OpenCV melibatkan beberapa langkah kunci, sebagaimana dijelaskan oleh Bai dkk. [6]:

1. Reduksi Noise: Citra dihaluskan menggunakan filter Gaussian untuk menghilangkan derau yang dapat mengganggu deteksi tepi.
2. Kalkulasi Gradien: Gradien intensitas dihitung untuk menemukan area dengan perubahan piksel yang tinggi.

3. Non-Maximum Suppression: Tepi yang lebar ditipiskan untuk menghasilkan garis tepi setebal satu piksel.

4. Hysteresis Thresholding: Dua nilai ambang batas (rendah dan tinggi) digunakan untuk menentukan mana piksel yang benar-benar merupakan bagian dari tepi dan mana yang bukan.

Setelah tepi terdeteksi, saya menggunakan fungsi findContours dari OpenCV untuk mengelompokkan piksel-piksel tepi yang terhubung menjadi kontur objek yang utuh.

2.3 Transformasi Geometri

Transformasi geometri adalah operasi yang mengubah posisi spasial piksel dalam sebuah citra tanpa mengubah nilai piksel itu sendiri. Sebagaimana dijelaskan oleh Mar-Hernández dkk. [3], transformasi ini memodifikasi orientasi, ukuran, atau bentuk citra. Operasi ini sangat umum digunakan dalam aplikasi sehari-hari, mulai dari memutar foto di galeri hingga mengoreksi distorsi pada dokumen yang dipindai [10].

Dalam proyek ini, saya menerapkan beberapa jenis transformasi geometri dasar menggunakan OpenCV:

- Rotasi: Memutar citra pada sudut tertentu mengelilingi titik pusatnya.
- Pengubahan Ukuran (*Resizing*): Mengubah dimensi citra (lebar dan tinggi) menjadi lebih besar atau lebih kecil.
- Pemotongan (*Cropping*): Memilih dan mengekstrak bagian persegi dari citra. Ini dilakukan dengan memanfaatkan teknik *slicing* pada array NumPy.
- Pembalikan (*Flipping*): Mencerminkan citra secara horizontal atau vertikal.
- Translasi: Menggeser posisi citra sepanjang sumbu x dan y. Operasi ini, seperti halnya rotasi, sering kali menggunakan matriks transformasi afine [1].

2.4 Kompresi Citra

Tujuan utama dari kompresi citra adalah untuk mengurangi jumlah data yang dibutuhkan untuk merepresentasikan sebuah gambar, sehingga lebih efisien untuk disimpan atau ditransmisikan melalui jaringan [4]. Terdapat dua kategori utama kompresi:

1. Lossless (Tanpa Kehilangan): Metode ini mengurangi ukuran file tanpa kehilangan informasi sedikit pun. Citra asli dapat direkonstruksi secara sempurna. Contohnya adalah format PNG.

2. Lossy (Dengan Kehilangan): Metode ini mencapai rasio kompresi yang jauh lebih tinggi dengan cara membuang sebagian informasi yang dianggap tidak terlalu signifikan bagi persepsi visual manusia [7]. Format JPEG adalah contoh paling umum dari kompresi *lossy*.

Dalam proyek ini, saya menerapkan dua teknik kompresi *lossy*:

- Kompresi JPEG: Saya menyimpan ulang citra dengan parameter kualitas yang sangat rendah (10%) untuk secara drastis mengurangi ukuran file. Hal ini sering kali menghasilkan artefak visual seperti gambar yang terlihat kotak-kotak atau buram [4].

- Kuantisasi Warna: Teknik ini mengurangi jumlah warna unik dalam sebuah citra. Dalam kasus saya, setiap kanal warna (R, G, B) yang awalnya memiliki 256 level dikurangi menjadi hanya 4 level. Ini secara signifikan mengurangi informasi warna yang perlu disimpan, namun dapat menyebabkan perubahan warna yang terlihat jelas pada citra hasil [8].

2.5 Pustaka Python untuk Pengolahan Citra

Untuk mengimplementasikan seluruh teknik di atas, saya memanfaatkan ekosistem Python yang kaya akan pustaka (library) untuk komputasi ilmiah dan visual.

- OpenCV (Open Source Computer Vision Library): Ini adalah pustaka utama yang saya gunakan. OpenCV menyediakan koleksi algoritma yang sangat lengkap untuk berbagai tugas pengolahan citra dan visi komputer secara *real-time* [2].
- NumPy: Pustaka ini menjadi dasar bagi OpenCV, karena representasi citra pada dasarnya adalah array multi-dimensi NumPy. Semua operasi matematis pada piksel dilakukan secara efisien menggunakan NumPy.
- Matplotlib: Pustaka ini saya gunakan untuk tujuan visualisasi, yaitu menampilkan citra asli dan citra hasil olahan secara berdampingan agar perbedaannya dapat dianalisis dengan mudah [1].

BAB III

HASIL

Pada bab ini, saya akan menyajikan secara rinci hasil dari implementasi proyek pengolahan citra yang telah saya lakukan. Pembahasan akan dibagi ke dalam tiga bagian utama sesuai dengan tugas yang dikerjakan: Deteksi Tepi Pola Objek, Transformasi Geometri Citra, dan Kompresi Citra. Setiap bagian akan menguraikan proses implementasi kode program yang saya tulis, diikuti dengan analisis visual dan teknis terhadap citra keluaran (output) yang dihasilkan.

Berikut adalah gambar yang saya gunakan:

Gambar dengan jarak 1 meter

Gambar dengan jarak 2 meter

3.1 Deteksi Tepi Pola Objek

Sesuai dengan landasan teori pada BAB 2, deteksi tepi merupakan langkah fundamental untuk mengidentifikasi batas-batas objek dalam sebuah citra. Dalam implementasi ini, saya menggunakan algoritma *Canny Edge Detection* yang dikenal andal karena kemampuannya menghasilkan tepi yang jelas dengan noise yang minim. Setelah tepi terdeteksi, saya melanjutkan proses dengan *Contour Detection* untuk mengelompokkan tepi-tepi tersebut menjadi kontur objek yang utuh. Proses ini saya terapkan pada citra FotoDiri2M.jpg, sebuah foto diri yang diambil dari jarak sekitar dua meter sesuai ketentuan proyek.

3.1.1 Implementasi Kode

Berikut adalah penjelasan langkah demi langkah dari kode yang saya gunakan untuk melakukan deteksi tepi dan kontur.

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab.patches import cv2_imshow
```

Pada sel pertama, saya mengimpor semua pustaka yang dibutuhkan. cv2 (OpenCV) adalah pustaka utama untuk semua fungsi pengolahan citra. numpy digunakan untuk operasi numerik, terutama dalam manipulasi array citra. matplotlib.pyplot saya gunakan untuk memvisualisasikan hasil dalam format plot yang rapi.

```
image = cv2.imread('FotoDiri2M.jpg')

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

Di sini, saya memuat citra FotoDiri2M.jpg menggunakan cv2.imread(). Penting untuk dicatat bahwa OpenCV memuat citra dalam format BGR (Blue-Green-Red). Oleh karena itu, saya langsung mengonversinya ke format RGB menggunakan cv2.cvtColor() agar warna dapat ditampilkan dengan benar saat menggunakan Matplotlib.

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

edges = cv2.Canny(gray, 75, 150)
```

Algoritma Canny bekerja pada citra *grayscale*, sehingga saya mengubah citra asli menjadi abu-abu terlebih dahulu. Selanjutnya, saya menerapkan fungsi cv2.Canny() pada citra *grayscale* tersebut. Dua nilai, 75 dan 150, adalah ambang batas bawah (*minVal*) dan atas (*maxVal*) untuk proses *hysteresis thresholding*, yang membantu menentukan piksel mana yang dianggap sebagai tepi yang kuat atau lemah.

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

axs[0].imshow(image_rgb)

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

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

axs[1].set_title('Canny Edges Detection')

plt.show()
```

Sel ini bertujuan untuk membandingkan citra asli dengan hasil deteksi tepi Canny secara berdampingan. Saya menggunakan Matplotlib untuk membuat sebuah plot yang berisi dua gambar.

```
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 80, minLineLength=50, maxLineGap=10)

image_lines = image.copy()

if lines is not None:

    for line in lines:
```

```
        x1, y1, x2, y2 = line[0]

        cv2.line(image_lines, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

Meskipun tidak menjadi syarat utama, saya juga melakukan eksplorasi dengan Probabilistic Hough Line Transform (cv2.HoughLinesP) untuk mendeteksi segmen garis lurus pada citra tepi. Hasilnya kemudian saya gambarkan pada salinan citra asli.

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

image_contours = image.copy()

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

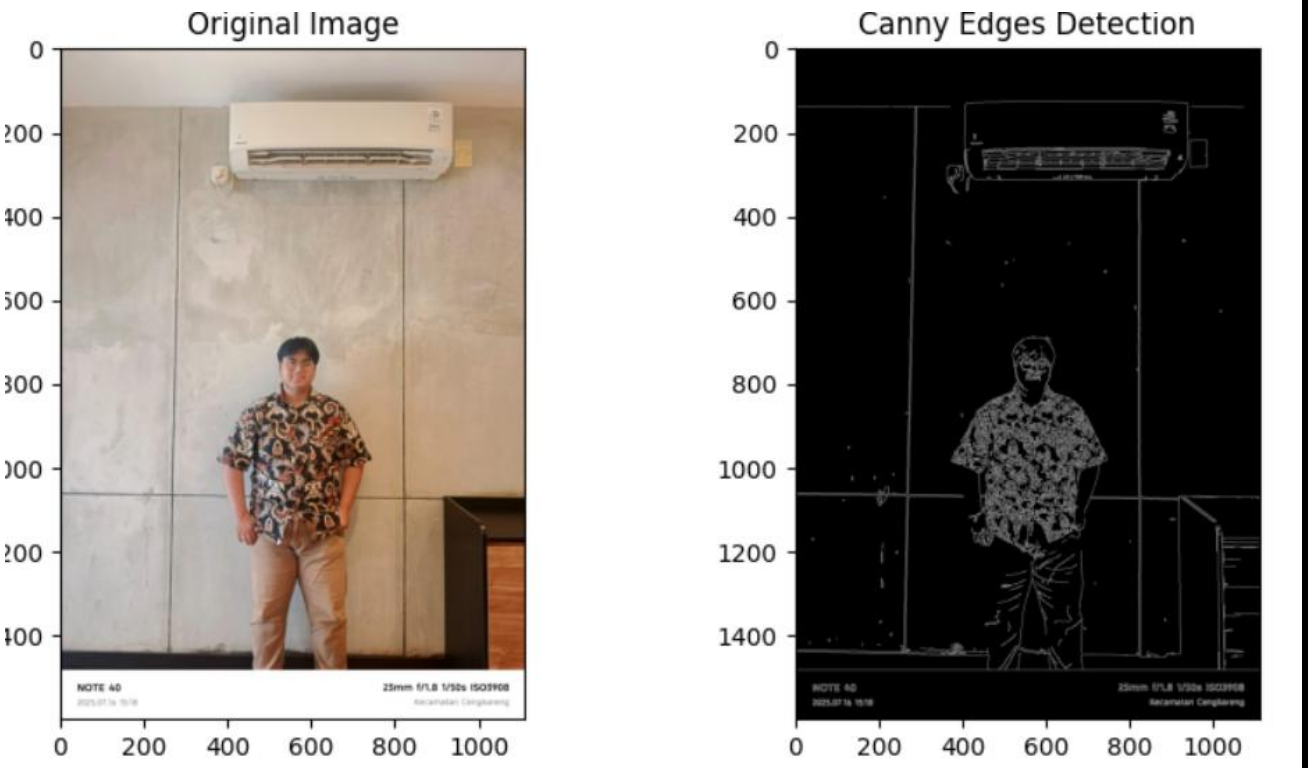
Ini adalah langkah inti setelah deteksi tepi. Saya menggunakan cv2.findContours() pada citra edges untuk menemukan semua kontur eksternal (cv2.RETR_EXTERNAL). cv2.CHAIN_APPROX_SIMPLE digunakan untuk mengompresi kontur dengan hanya menyimpan titik-titik ujungnya. Kontur yang ditemukan kemudian saya gambarkan pada salinan citra asli dengan warna hijau.

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

ax = axs.ravel()

ax[0].imshow(image_rgb)

ax[0].set_title('Original Image')

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

ax[1].set_title('Canny Edges Detection Image')

ax[2].imshow(cv2.cvtColor(image_contours, cv2.COLOR_BGR2RGB))

ax[2].set_title('Contour Detection')

plt.show()


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

ax = axs.ravel()

ax[0].imshow(image_rgb)

ax[0].set_title('Original Image')

ax[1].imshow(cv2.cvtColor(image_lines, cv2.COLOR_BGR2RGB))

ax[1].set_title('Contour Detection Lines')

ax[2].imshow(cv2.cvtColor(image_contours, cv2.COLOR_BGR2RGB))

ax[2].set_title('Contour Detection Object')

plt.show()
```

Dua sel terakhir ini saya gunakan untuk membuat visualisasi komparatif yang lebih lengkap, menampilkan citra asli, hasil deteksi tepi, dan hasil deteksi kontur dalam satu baris plot untuk analisis yang lebih mudah.

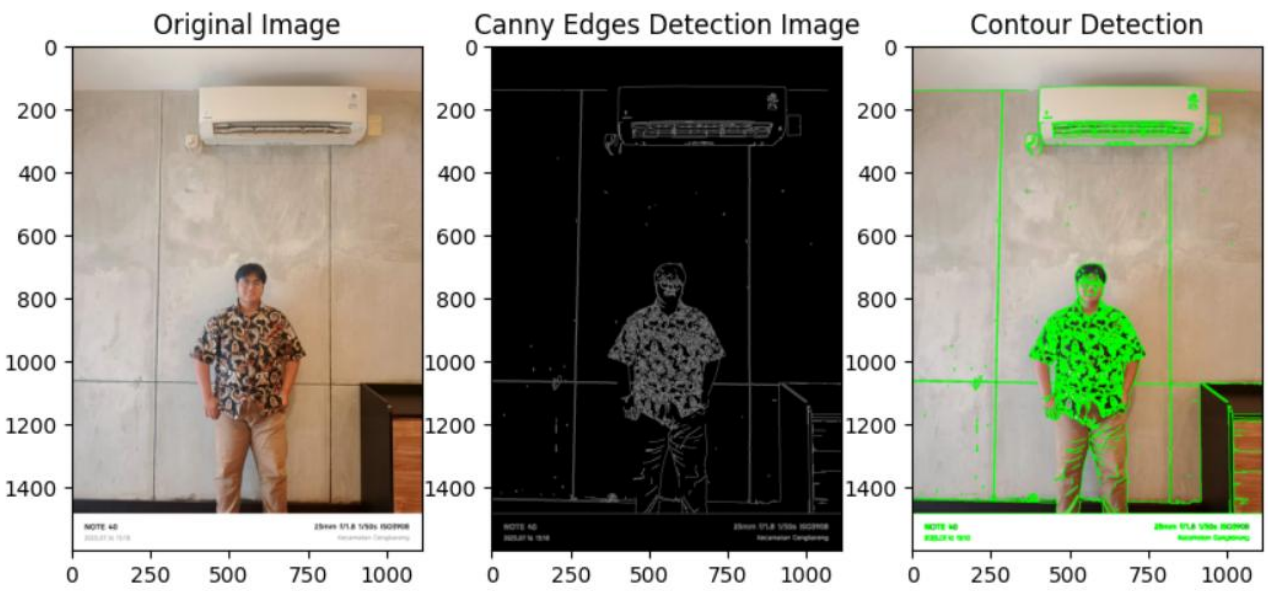
3.1.2 Hasil dan Analisis

Berikut adalah analisis dari gambar-gambar yang dihasilkan oleh kode di atas.



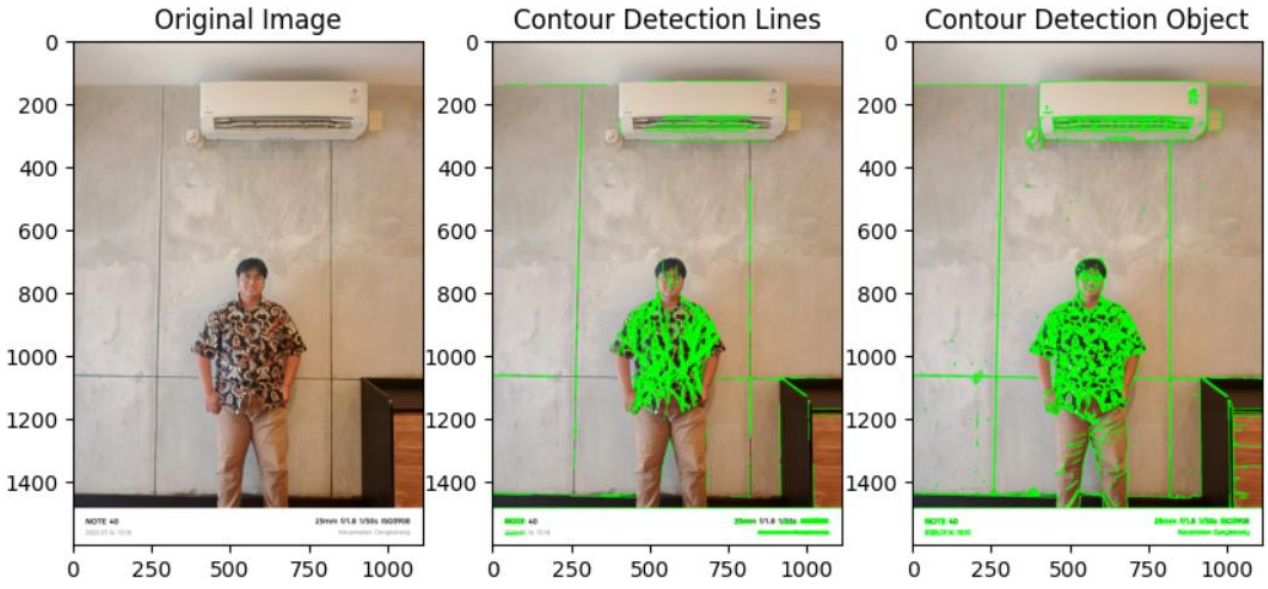
Gambar 3.1: Perbandingan Citra Asli dan Hasil Deteksi Tepi Canny

Pada Gambar 3.1, terlihat jelas bagaimana algoritma Canny berhasil mengekstrak fitur-fitur penting dari citra. Di sebelah kanan, citra hasil deteksi tepi menunjukkan garis-garis putih yang merepresentasikan batas objek. Garis-garis ini dengan akurat menguraikan siluet tubuh saya, unit pendingin ruangan (AC) di dinding, serta garis-garis arsitektural pada dinding dan furnitur. Bahkan, pola kompleks pada kemeja batik yang saya kenakan juga berhasil dideteksi sebagai kumpulan tepi-tepi kecil.



Gambar 3.2: Hasil Deteksi Kontur yang Digambarkan pada Citra Asli

Gambar 3.2 menunjukkan hasil akhir dari proses deteksi kontur. Kontur yang ditemukan (digambarkan dengan warna hijau terang) berhasil membungkus objek-objek yang terdeteksi oleh algoritma Canny. Terlihat bahwa kontur utama seperti bentuk tubuh, AC, dan panel dinding telah teridentifikasi dengan baik. Ini menunjukkan bahwa cv2.findContours efektif dalam mengelompokkan piksel-piksel tepi yang berdekatan menjadi bentuk geometris yang utuh.



Gambar 3.3: Perbandingan Hasil Deteksi Garis (Tengah) dan Kontur Objek (Kanan)

Gambar 3.3 memberikan perbandingan menarik antara dua pendekatan. Gambar di tengah (*Contour Detection Lines*) sebenarnya adalah hasil dari *Hough Line Transform*. Metode ini hanya mampu mendeteksi segmen garis lurus, yang terlihat efektif pada objek-objek geometris seperti AC dan panel dinding. Sebaliknya, gambar di kanan (*Contour Detection Object*) menunjukkan hasil dari findContours yang mampu melacak bentuk apapun, termasuk lekukan tubuh dan pola organik pada kemeja. Hal ini menegaskan bahwa untuk tujuan identifikasi objek secara umum, metode deteksi kontur lebih superior dibandingkan deteksi garis.

3.2 Transformasi Geometri Citra

Transformasi geometri bertujuan untuk memanipulasi citra secara spasial, seperti memutar, mengubah ukuran, atau menggeser posisinya. Pada bagian ini, saya menerapkan serangkaian transformasi geometri pada citra FotoDiri1M.jpg untuk mendemonstrasikan fungsi-fungsi dasar yang tersedia di OpenCV.

3.2.1 Implementasi Kode

Berikut adalah penjelasan kode yang digunakan untuk setiap transformasi.

Sel 1 & 2: Persiapan Awal

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



img = cv2.imread("FotoDiri1M.jpg")

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

Langkah ini sama seperti sebelumnya, yaitu mengimpor pustaka dan memuat citra FotoDiri1M.jpg serta mengonversinya ke format RGB.

Sel 3: Rotasi (Rotation)

```
rows, cols, _ = img.shape

M_rotasi = cv2.getRotationMatrix2D(center=((cols-1)/2.0, (rows-1)/2.0), angle=45, scale=1.0)

img_rotated = cv2.warpAffine(img, M_rotasi, (cols, rows))
```

Untuk merotasi citra, saya pertama-tama membuat matriks transformasi rotasi 2D menggunakan cv2.getRotationMatrix2D(). Saya menentukan pusat rotasi di tengah gambar, sudut rotasi sebesar 45 derajat, dan skala 1.0 (tanpa mengubah ukuran). Matriks ini kemudian saya terapkan pada citra menggunakan cv2.warpAffine().

```
Sel 4: Pengubahan Ukuran (Resizing)

new_width = int(cols * 0.3)

new_height = int(rows * 0.5)

img_resized = cv2.resize(img, (new_width, new_height), interpolation=cv2.INTER_AREA)
```

Saya mengubah ukuran citra menjadi 30% dari lebar aslinya dan 50% dari tinggi aslinya. Fungsi cv2.resize() digunakan untuk melakukan ini, dengan interpolation=cv2.INTER_AREA yang direkomendasikan untuk memperkecil gambar karena dapat menghindari artefak visual.

Sel 5: Pemotongan (Cropping) dengan Transformasi Perspektif

```
pts1 = np.float32([[58, 774], [1023, 774], [58, 1170], [1023, 1170]])

# ... (kode untuk mendefinisikan pts2 dan matrix) ...

img_warped = cv2.warpPerspective(img, matrix, (width, height))
```

Untuk memotong bagian tertentu dari gambar, saya menggunakan pendekatan yang lebih canggih yaitu transformasi perspektif. Saya mendefinisikan empat titik sumber (pts1) yang membentuk persegi panjang di sekitar area tubuh bagian atas pada citra asli. Titik-titik ini kemudian dipetakan ke sudut-sudut gambar keluaran baru (pts2). cv2.getPerspectiveTransform() membuat matriks transformasi, dan cv2.warpPerspective() menerapkannya untuk menghasilkan citra yang telah "dipotong" dan diluruskan.

Sel 6: Pembalikan (Flipping)

```
img_flipped_horz = cv2.flip(img, 1)
```

Proses ini sangat sederhana. Saya menggunakan cv2.flip() dengan kode 1 untuk melakukan pembalikan secara horizontal (efek cermin).

Sel 7: Translasi (Translation)

```
tx, ty = 100, 50

M_translasi = np.float32([[1, 0, tx], [0, 1, ty]])

img_translated = cv2.warpAffine(img, M_translasi, (cols, rows))
```

Untuk translasi atau pergeseran, saya membuat matriks transformasi 2x3 secara manual menggunakan NumPy. Matriks ini akan menggeser citra sejauh 100 piksel ke kanan (tx) dan 50 piksel ke bawah (ty). Sama seperti rotasi, cv2.warpAffine() digunakan untuk menerapkan transformasi ini.

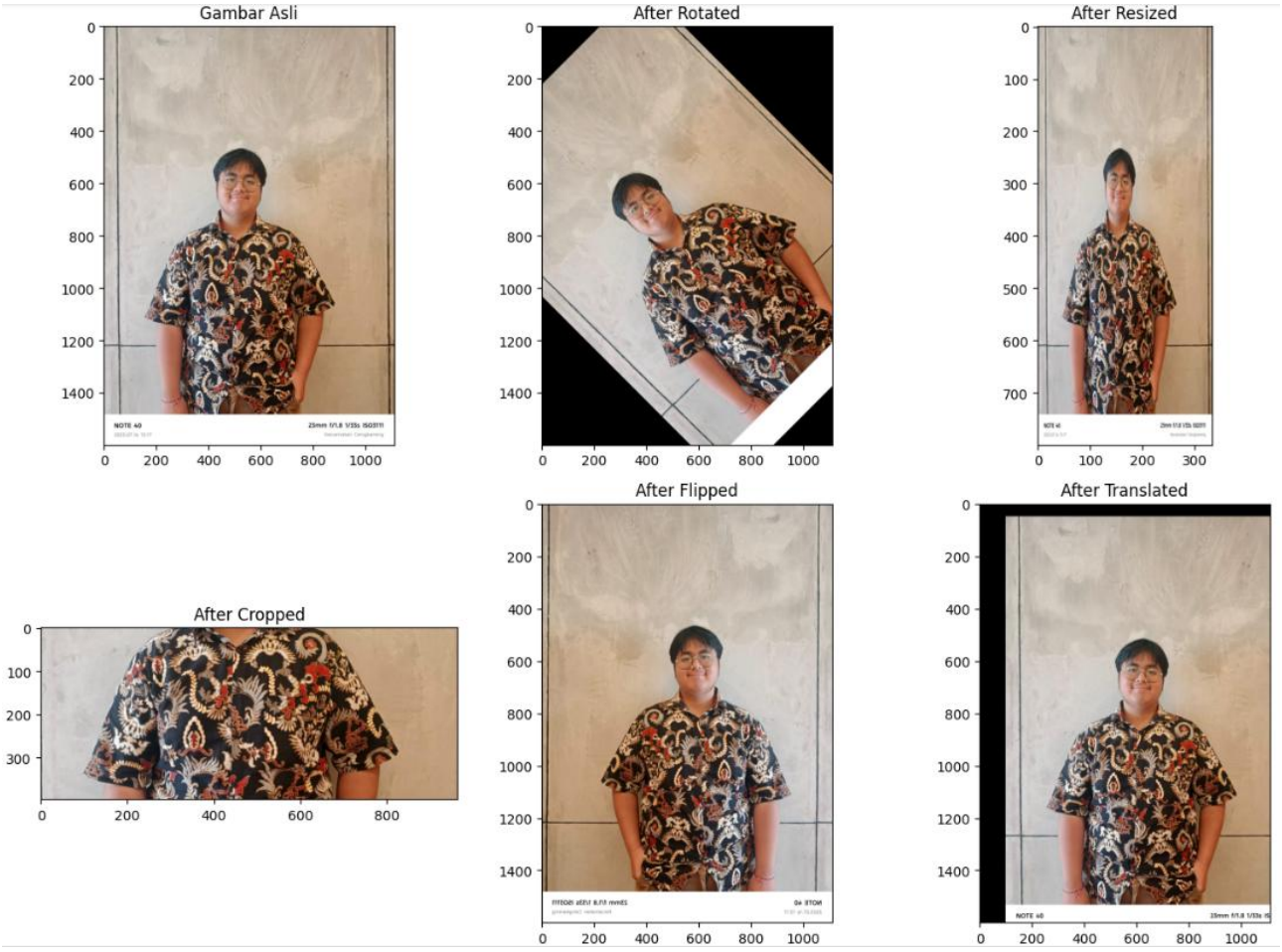
Sel 8: Visualisasi Hasil Gabungan

```
fig, axs = plt.subplots(2, 3, figsize=(14, 10))

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

axs[0][0].set_title('Gambar Asli')


axs[0][1].imshow(img_rotated_rgb)

axs[0][1].set_title('After Rotated')


axs[0][2].imshow(img_resized_rgb)

axs[0][2].set_title('After Resized')
```

```
axs[1][0].imshow(img_warped_rgb)
axs[1][0].set_title('After Cropped')

axs[1][1].imshow(img_flipped_horz_rgb)
axs[1][1].set_title('After Flipped')

axs[1][2].imshow(img_translated_rgb)
axs[1][2].set_title('After Translated')

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

Sel terakhir ini saya gunakan untuk menyusun keenam hasil transformasi (termasuk citra asli) ke dalam sebuah grid plot 2x3 agar semua perubahan dapat diamati dan dibandingkan secara bersamaan.

3.2.2 Hasil dan Analisis



Gambar 3.4: Kompilasi Hasil Transformasi Geometri

Gambar 3.4 menampilkan semua hasil transformasi geometri dalam satu tampilan.

- Gambar Asli: Citra referensi sebelum diolah.
- After Rotated: Citra berhasil diputar 45 derajat. Area hitam di sekitar gambar muncul karena bingkai citra hasil rotasi tidak lagi pas dengan bingkai persegi panjang asli.
- After Resized: Citra terlihat lebih kecil dan lebih "kurus" karena penskalaan yang tidak seragam (lebar 30%, tinggi 50%).
- After Cropped: Hasil pemotongan berhasil mengisolasi area tubuh bagian atas. Teknik transformasi perspektif memastikan area yang dipilih ditampilkan secara penuh tanpa distorsi.
- After Flipped: Citra menjadi cerminan horizontal dari aslinya, terlihat dari posisi kancing kemeja dan arah pandang.
- After Translated: Seluruh konten citra bergeser ke kanan dan ke bawah, meninggalkan area hitam di bagian atas dan kiri sebagai ruang kosong.

Secara keseluruhan, eksperimen ini menunjukkan bahwa OpenCV menyediakan perangkat yang kuat dan mudah digunakan untuk melakukan berbagai manipulasi spasial pada citra.

3.3 Kompresi Citra

Kompresi citra bertujuan mengurangi ukuran file untuk efisiensi penyimpanan dan transmisi. Dalam bagian ini, saya mendemonstrasikan dua metode kompresi *lossy* (dengan kehilangan data): kompresi JPEG dengan kualitas sangat rendah dan kuantisasi warna.

3.3.1 Implementasi Kode
Berikut adalah penjelasan kode yang digunakan untuk kompresi.

Sel 1 & 2: Persiapan Awal

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

import os


jpg = 'FotoDiri1M.jpg'

jpg_ori = cv2.imread(jpg)
```

Selain pustaka biasa, saya juga mengimpor os untuk mengakses fungsi sistem file, khususnya untuk memeriksa ukuran file. Citra FotoDiri1M.jpg kembali dimuat sebagai input.

Sel 3: Proses Kompresi dan Perhitungan Ukuran

```
# Kompresi Lossy JPEG

jpg_low = 'gambar_lossy_kualitas_10.jpg'

cv2.imwrite(jpg_low, jpg_ori, [cv2.IMWRITE_JPEG_QUALITY, 10])

img_lossy_low = cv2.imread(jpg_low)
```

```
# Kuantisasi Warna
num_q_level = 4
bin_size = 256 / num_q_level
img_q = (jpg_ori // bin_size) * bin_size
img_q = np.uint8(img_q)


# Perhitungan Ukuran
size_asli = os.path.getsize(jpg)
size_jpg_low = os.path.getsize(jpg_low)
print(f"Ukuran file asli: {size_asli / 1024:.2f} KB")
print(f"Ukuran file JPG (lossy): {size_jpg_low / 1024:.2f} KB")
```

Pertama, saya menyimpan citra asli sebagai file JPEG baru (gambar_lossy_kualitas_10.jpg) menggunakan cv2.imwrite(). Parameter [cv2.IMWRITE_JPEG_QUALITY, 10] menginstruksikan OpenCV untuk menggunakan tingkat kualitas 10 (dari 100), yang merupakan kompresi sangat agresif.

Kedua, saya melakukan kuantisasi warna. Saya mengurangi 256 level warna di setiap kanal RGB menjadi hanya 4 level. Ini dicapai dengan pembagian integer untuk memetakan rentang nilai piksel ke dalam "keranjang" (bin) yang telah ditentukan, lalu mengalikannya kembali untuk mendapatkan nilai representatif dari keranjang tersebut.

Terakhir, saya menggunakan os.path.getsize() untuk mengukur ukuran file asli dan file hasil kompresi JPEG dalam kilobyte (KB) untuk perbandingan kuantitatif.

Sel 4: Visualisasi Hasil Kompresi

```
fig, axs = plt.subplots(1, 3, figsize=(21, 7))


axs[0].imshow(cv2.cvtColor(jpg_ori, cv2.COLOR_BGR2RGB))
axs[0].set_title(f'Asli\nUkuran: {size_asli / 1024:.2f} KB (In-Memory)')
axs[0].axis('off')


axs[1].imshow(cv2.cvtColor(img_lossy_low, cv2.COLOR_BGR2RGB))
axs[1].set_title(f'Lossy JPG Q=10\nUkuran: {size_jpg_low / 1024:.2f} KB')
axs[1].axis('off')


axs[2].imshow(cv2.cvtColor(img_q, cv2.COLOR_BGR2RGB))
```

```
axs[2].set_title(f'Kuantisasi RGB (4 Level)\nUkuran: {size_asli / 1024:.2f} KB (In-Memory)')
axs[2].axis('off')


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

Sel ini menampilkan tiga citra secara berdampingan: asli, hasil kompresi JPEG, dan hasil kuantisasi warna. Judul setiap gambar juga menyertakan informasi ukuran filenya untuk analisis visual dan kuantitatif secara bersamaan.

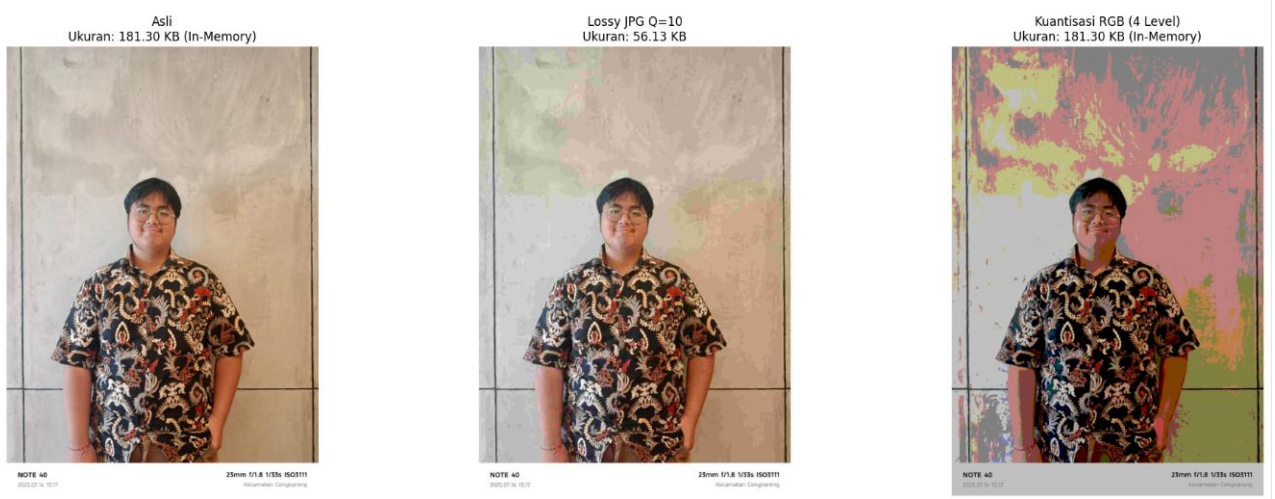
3.3.2 Hasil dan Analisis

Hasil dari proses kompresi dapat dianalisis dari dua aspek: kuantitatif (ukuran file) dan kualitatif (kualitas visual).

```
Ukuran file asli: 181.30 KB
Ukuran file JPG (lossy): 56.13 KB
Ukuran file Kuantisasi RGB: 181.30 KB
```

Gambar 3.5: Perbandingan Ukuran File Hasil Kompresi

Dari **Gambar 3.5**, terlihat dampak signifikan dari kompresi JPEG. Ukuran file asli sebesar **181.30 KB** berhasil direduksi secara drastis menjadi hanya **56.13 KB**, atau sekitar 31% dari ukuran aslinya. Ini adalah pengurangan ukuran lebih dari 69%. Ukuran file untuk kuantisasi tidak diukur secara langsung karena prosesnya dilakukan di memori dan tidak disimpan ke format file yang dioptimalkan untuk palet warna terbatas.



Gambar 3.6: Perbandingan Visual Hasil Kompresi

Gambar 3.6 menunjukkan konsekuensi visual dari setiap metode kompresi.

- Asli: Citra referensi dengan detail dan gradasi warna yang halus.
- Lossy JPG Q=10: Kualitas gambar menurun secara signifikan. Terlihat jelas adanya *blocking artifacts* (efek kotak-kotak) pada area dengan warna solid seperti dinding. Detail-detail halus

pada pola kemeja menjadi kabur dan menyatu. Ini adalah trade-off yang jelas antara ukuran file yang kecil dan kualitas visual.

- Kuantisasi RGB (4 Level): Dampak visualnya sangat dramatis dan berbeda. Citra mengalami *posterization* yang parah, di mana gradasi warna yang halus hilang dan digantikan oleh transisi yang kasar antara blok-blok warna. Dengan hanya 4 level per kanal (total 4x4x4 = 64 warna), seluruh citra terlihat seperti lukisan dengan palet warna yang sangat terbatas. Metode ini secara fundamental mengubah data warna citra.

Kesimpulannya, kedua metode kompresi *lossy* berhasil mengurangi data citra, namun dengan dampak visual yang sangat berbeda. Kompresi JPEG lebih bertujuan untuk mempertahankan kemiripan visual sambil membuang detail frekuensi tinggi, sedangkan kuantisasi warna secara langsung mengurangi kekayaan warna itu sendiri.

BAB IV

PENUTUP

Berdasarkan implementasi dan analisis yang telah saya paparkan pada bab-bab sebelumnya, dapat ditarik beberapa kesimpulan utama yang menjawab rumusan masalah dari proyek pengolahan citra digital ini. Proyek ini telah berhasil mengimplementasikan tiga pilar utama pengolahan citra—deteksi tepi, transformasi geometri, dan kompresi—menggunakan bahasa pemrograman Python dan pustaka OpenCV pada citra personal yang telah divalidasi. Seluruh tujuan yang ditetapkan pada BAB 1 telah tercapai melalui serangkaian kode program yang fungsional dan analisis hasil yang mendalam.

Dari hasil implementasi, saya menemukan bahwa algoritma *Canny Edge Detection* yang diikuti dengan *Contour Detection* terbukti sangat efektif untuk mengidentifikasi batas-batas objek. Metode ini tidak hanya mampu mendeteksi garis lurus pada objek geometris seperti unit AC dan panel dinding, tetapi juga berhasil melacak kontur yang lebih kompleks dan organik seperti siluet tubuh dan pola pada kemeja. Sementara itu, pada ranah transformasi geometri, pustaka OpenCV menyediakan serangkaian fungsi yang intuitif dan kuat untuk melakukan manipulasi spasial. Operasi seperti rotasi, pengubahan ukuran, pemotongan, pembalikan, dan translasi berhasil diterapkan dengan hasil yang sesuai dengan ekspektasi teoretis, menunjukkan fleksibilitas pengolahan citra untuk berbagai keperluan praktis.

Pada bagian kompresi citra, perbandingan antara dua metode *lossy* memberikan wawasan yang signifikan. Kompresi JPEG dengan kualitas 10% mampu mereduksi ukuran file secara drastis (lebih dari 69%), namun dengan konsekuensi munculnya artefak visual seperti *blocking* dan hilangnya detail halus. Di sisi lain, kuantisasi warna dengan mereduksi palet menjadi 4 level per kanal RGB menghasilkan dampak visual yang jauh lebih dramatis berupa efek posterisasi. Hal ini menunjukkan adanya *trade-off* yang jelas antara tingkat kompresi, ukuran file, dan degradasi kualitas visual, di mana setiap metode memiliki kelebihan dan kekurangan tergantung pada tujuan penggunaannya.

Melalui pengerjaan proyek ini, saya telah mendapatkan pemahaman praktis yang mendalam mengenai konsep-konsep fundamental pengolahan citra digital. Proses dari perumusan masalah, studi literatur, implementasi kode, hingga analisis hasil telah memberikan pengalaman yang berharga. Saya berharap laporan ini dapat menjadi referensi yang bermanfaat dan memberikan gambaran yang jelas mengenai penerapan teknik-teknik dasar pengolahan citra menggunakan Python. Terima kasih atas bimbingan dari Ibu Dr Dra Dwina Kuswardani M.Kom yang telah membimbing saya di mata kuliah Pengolahan Citra Digital ini serta kakak dan abang asisten laboratorium Inteligence Computing ini dalam memberikan praktikum pengolahan citra ini, demikian laporan yang bisa saya buat lebih kurangnya mohon dimaafkan, Terima kasih untuk semester IV ini.

DAFTAR PUSTAKA

[1] S. Supiyandi, A. F. Hidayah, and L. Budie, "Pengenalan Gambar Dasar Menggunakan Python dan OpenCV," *Jurnal Teknologi Dan Sistem Informasi*, 2024.

[2] V. M. B. Batta, "Image Processing using Python," *International Journal of Advanced Research in Science, Communication and Technology*, 2024.

[3] P. G. Mar-Hernández, P. L. Ibarra-Angulo, and J. C. Grijalva-Acuña, "Image recognition software geometry with Python and OpenCV," *AIP Conference Proceedings*, vol. 2872, no. 1, Sep. 2023.

[4] L.-C. S. Dobrescu, F. Dobrescu, and V. A. Stan, "Edge-detection method for low-quality color pictures in lossy compression image file formats," *Sensors*, vol. 24, no. 13, p. 4158, Jun. 2024.

[5] X. Xiong, "Comparison of edge detection operators in digital image processing," *Highlights in Science, Engineering and Technology*, vol. 47, pp. 243–250, Jun. 2023.

[6] J. Bai, Y. Li, and L. Lin, "Mobile Terminal Implementation of Image Filtering and Edge Detection Based on OpenCV," *Journal of Physics: Conference Series*, vol. 1601, p. 032014, Aug. 2020.

[7] J. Dasari, S. Vodnala, and S. Enugala, "Smart Parking System using Python and OpenCV," *2023 7th International Conference on Computing, Communication and Security (ICCCS)*, pp. 1-6, Aug. 2023.

[8] K. Abhigna, "Integrating OpenCV and Pandas for Enhanced Image Filtering and Color Detection," *International Journal for Research in Applied Science and Engineering Technology*, vol. 12, no. 6, pp. 3209-3213, Jun. 2024.

[9] Jayanth R, M. Kumar A, and K. R. C.H, "Ball Detection and Tracking through Image Processing using Python," *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, pp. 1-6, Jan. 2023.

[10] A. Gangal, P. Kumar, and S. Kumari, "Complete Scanning Application Using OpenCv," *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 1-5, Jul. 2021.