UAS

PENGOLAHAN CITRA

NAMA : Nathasya Napitupulu

NIM : 202331074

KELAS : F

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

NO.PC : 10

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

BAB I

PENDAHULUAN

Pengolahan citra digital adalah bidang yang penting dalam teknologi modern, dengan aplikasi yang luas dalam pengenalan objek, analisis citra medis, dan banyak lagi. Salah satu teknik kunci dalam pengolahan citra adalah segmentasi warna, yang bertujuan untuk memisahkan objek berdasarkan warna mereka. Ruang warna HSV (Hue, Saturation, Value) menawarkan representasi yang lebih intuitif dibandingkan ruang warna RGB, memungkinkan pemisahan objek yang lebih efektif, terutama dalam kondisi pencahayaan yang bervariasi.

Selain segmentasi warna, filtering warna digunakan untuk meningkatkan atau memanipulasi komponen warna dalam citra, sehingga dapat menyoroti fitur tertentu. Kompresi citra juga merupakan aspek penting, yang bertujuan untuk mengurangi ukuran file citra tanpa mengorbankan kualitas. Terdapat dua tipe kompresi: lossy, yang menghilangkan beberapa informasi, dan lossless, yang mempertahankan semua detail.

Laporan UAS Pengolahan Citra Digital ini akan membahas secara ringkas mengenai segmentasi warna, ruang warna HSV, filtering warna, dan kompresi citra, serta metode dan aplikasi masing-masing dalam pengolahan citra digital.

1.1 Rumusan Masalah

Adapun rumusan masalah dari uas praktikum ini adalah:

1. Bagaimana teknik filtering warna dapat digunakan untuk meningkatkan kualitas citra dalam aplikasi tertentu?
2. Bagaimana cara segmentasi warna dapat meningkatkan akurasi dalam pengenalan objek dalam citra digital?
3. Apa perbedaan antara kompresi citra lossy dan lossless, dan dalam situasi apa masing-masing metode sebaiknya digunakan?

1.2 Tujuan Masalah

Adapun tujuan masalah dari uas praktikum ini adalah:

1. Memahami kompresi citra
2. Meningkatkan kualitas citra
3. Meningkatkan akurasi pengenalan objek

1.3 Manfaat Masalah

Adapun manfaat masalah dari uas praktikum ini adalah:

1. Peningkatan Kualitas Citra
2. Peningkatan Akurasi dalam Pengenalan Objek
3. Pemilihan Metode Kompresi yang Tepat

BAB II
LANDASAN TEORI

2.1 Deteksi Daun

Segmentasi warna adalah proses memisahkan objek dalam citra berdasarkan warna mereka. Ini adalah langkah penting dalam pengolahan citra yang memungkinkan kita untuk mengidentifikasi dan mengekstrak objek tertentu dari latar belakang. Segmentasi warna sering digunakan dalam aplikasi seperti pengenalan objek, analisis citra, dan visi komputer.

Ruang Warna HSV

Ruang warna HSV adalah representasi warna yang lebih intuitif dibandingkan dengan ruang warna RGB. Dalam ruang warna HSV:

- Hue (H): Menunjukkan warna yang sebenarnya, diukur dalam derajat (0-360). Misalnya, 0° adalah merah, 120° adalah hijau, dan 240° adalah biru[1].
- Saturation (S): Menunjukkan kejenuhan warna, diukur dalam persentase (0-100%). Saturasi 0% berarti warna tersebut adalah abu-abu, sedangkan 100% berarti warna tersebut sepenuhnya jenuh.
- Value (V): Menunjukkan kecerahan warna, juga diukur dalam persentase (0-100%). Value 0% berarti hitam, sedangkan 100% berarti warna sepenuhnya cerah.

2.2 Filtering Warna

Filtering warna adalah teknik dalam pengolahan citra digital yang digunakan untuk memanipulasi atau mengekstrak informasi warna dari citra[2]. Proses ini melibatkan penerapan filter untuk mengubah atau memperkuat komponen warna tertentu dalam citra, sehingga dapat meningkatkan kualitas citra atau menyoroti fitur tertentu.

Pentingnya filtering warna agar memperbaiki kontras antara objek dan latar belakang, sehingga objek lebih terlihat, memisahkan objek berdasarkan warna, yang berguna dalam pengenalan objek dan analisis citra, mengoreksi ketidakseimbangan warna dalam citra untuk menghasilkan tampilan yang lebih alami[3].

Sebelum membahas teknik filtering warna, penting untuk memahami ruang warna yang umum digunakan dalam pengolahan citra:

- RGB (Red, Green, Blue)
  Ruang warna paling umum, di mana setiap piksel diwakili oleh kombinasi tiga komponen warna.
- HSV (Hue, Saturation, Value)
  Ruang warna yang lebih intuitif untuk manipulasi warna, yang di mana hue menunjukkan warna (0-360 derajat), saturation untuk menunjukkan kejenuhan warna (0-100%), dan value untuk menunjukkan kecerahan warna (0-100%).

- LAB
  Ruang warna yang dirancang untuk mendekati persepsi manusia terhadap warna, terdiri dari komponen L (lightness), a (warna merah-hijau), dan b (warna kuning-biru).

Metode Filtering Warna

Berikut adalah beberapa metode filtering warna yang umum digunakan:

1. Tresholding Warna
   Thresholding warna adalah teknik untuk memisahkan objek berdasarkan nilai warna tertentu. Proses ini melibatkan:
   - Konversi Ruang Warna
     Mengubah citra dari ruang warna RGB ke ruang warna yang lebih sesuai, seperti HSV atau LAB.
   - Penetapan Ambang
     Menetapkan ambang batas untuk komponen warna tertentu. Misalnya, untuk mengekstrak objek berwarna hijau, kita dapat menetapkan ambang untuk nilai Hue di sekitar 120 derajat.
   - Penerapan Mask
     Menerapkan mask untuk mengekstrak piksel yang memenuhi kriteria ambang.

2. Filter Warna Berbasis Kernel
   Filter warna berbasis kernel melibatkan penerapan kernel (matriks) untuk memanipulasi nilai warna dalam citra. Contoh filter yang dapat digunakan adalah:
   - Filter Gaussian
     Menghaluskan citra dengan mengurangi noise, yang dapat diterapkan pada setiap komponen warna.
   - Filter Median
     Menggunakan nilai median dari piksel di sekitar untuk mengurangi noise sambil mempertahankan tepi.

3. Peningkatan Warna
   Peningkatan warna melibatkan manipulasi komponen warna untuk meningkatkan kontras dan kejenuhan. Beberapa teknik yang umum digunakan adalah:
   - Penyesuaian Histogram
     Mengubah distribusi nilai warna untuk meningkatkan kontras. Ini dapat dilakukan dengan meratakan histogram atau menggunakan metode CLAHE (Contrast Limited Adaptive Histogram Equalization).
   - Peningkatan Saturasi
     Meningkatkan nilai saturasi dalam ruang warna HSV untuk membuat warna lebih hidup.

2.3 Kompresi

Kompresi citra adalah proses mengurangi ukuran file citra tanpa mengorbankan kualitas citra secara signifikan. Tujuan utama dari kompresi citra adalah untuk menghemat ruang penyimpanan dan mempercepat transmisi data, terutama dalam aplikasi yang memerlukan pengiriman citra melalui jaringan[4].

Pentingnya kompresi citra agar mengurangi ukuran file citra memungkinkan lebih banyak citra disimpan dalam ruang penyimpanan yang terbatas, mempercepat pengiriman citra melalui jaringan, yang sangat penting dalam aplikasi web dan multimedia, mengurangi waktu pemrosesan citra karena ukuran data yang lebih kecil.

Kompresi citra dapat dibagi menjadi dua kategori utama:

1. Kompresi Lossy

   Kompresi lossy adalah metode kompresi yang mengurangi ukuran file dengan menghilangkan beberapa informasi dari citra. Meskipun ini dapat menghasilkan pengurangan ukuran file yang signifikan, kualitas citra dapat menurun, dan beberapa detail mungkin hilang secara permanen. Kompresi lossy sering digunakan dalam aplikasi di mana ukuran file yang kecil lebih penting daripada kualitas citra yang sempurna.

   Karakteristik Kompresi Lossy yaitu :

   - Dapat mengurangi ukuran file hingga 90% atau lebih, tergantung pada tingkat kompresi yang diterapkan
   - Beberapa detail citra hilang, dan citra tidak dapat dikembalikan ke bentuk aslinya.
   - Semakin tinggi tingkat kompresi, semakin rendah kualitas citra.

2. Kompresi Lossless

   Kompresi lossless adalah metode kompresi yang memungkinkan citra dikompresi dan kemudian dikembalikan ke bentuk aslinya tanpa kehilangan informasi. Ini penting untuk aplikasi di mana kualitas citra harus dipertahankan, seperti dalam pengolahan citra medis atau penyimpanan arsip.

   Karakteristik Kompresi Lossless yaitu:

   - Citra dapat dikembalikan ke bentuk aslinya tanpa kehilangan detail.
   - Meskipun ukuran file dapat dikurangi, pengurangan ini biasanya tidak sebesar kompresi lossy.
   - Citra tetap dalam kualitas aslinya setelah dekompresi.

BAB III
HASIL

3.1 DETEKSI DAUN

```
[2]: import cv2
     import numpy as np
     import matplotlib.pyplot as plt
```

Yang pertama sekali pastinya saya import library dengan cara import cv2, mengimpor pustaka OpenCV (Open Source Computer Vision Library) yang banyak digunakan untuk tugas-tugas penglihatan komputer seperti pemrosesan gambar dan video, import numpy as np, mengimpor pustaka NumPy, yang merupakan fondasi untuk komputasi numerik di Python dan sangat penting untuk bekerja dengan array dan matriks data, yang seringkali merepresentasikan gambar. Terakhir, import matplotlib.pyplot as plt mengimpor modul pyplot dari pustaka Matplotlib.

```
image_path = 'mangga.jpg'
image = cv2.imread(image_path)
```

Pada bagian ini, image_path = 'mangga.jpg', mendeklarasikan sebuah variabel bernama image_path dan menetapkan nilai string 'mangga.jpg' padanya. Kemudian image = cv2.imread(image_path), menggunakan fungsi imread() dari pustaka OpenCV (cv2) untuk membaca gambar yang ditentukan oleh image_path. Hasil dari pembacaan gambar ini kemudian disimpan ke dalam variabel image.

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

Disini, image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB), mengubah gambar asli yang kemungkinan besar dibaca dalam format BGR (Blue-Green-Red) standar OpenCV, menjadi format RGB (Red-Green-Blue). Kemudian hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV) melakukan perubahan format warna lain. Kali ini, gambar diubah dari BGR menjadi HSV (Hue-Saturation-Value). Perubahan ke HSV ini seringkali sangat berguna karena memisahkan informasi warna seperti "apa warnanya" dan "seberapa pekat warnanya" dari informasi kecerahan "seberapa terang warnanya".

```
lower_mangga = np.array([30, 80, 80])
upper_mangga = np.array([70, 255, 255])
mask_mangga = cv2.inRange(hsv_image, lower_mangga, upper_mangga)
```

Lower dan upper mangga menentukan rentang nilai warna (Hue, Saturation, Value) yang dianggap sebagai target, dalam hal ini, warna yang menyerupai mangga. Kemudian menggunakan rentang warna tersebut pada gambar yang sudah dikonversi ke ruang warna HSV (hsv_image). Hasilnya adalah sebuah mask_mangga, yaitu gambar hitam-putih di mana area yang cocok dengan rentang warna yang ditentukan akan berwarna putih, sementara sisanya berwarna hitam.

```
kernel = np.ones((5,5), np.uint8)
mask_mangga = cv2.morphologyEx(mask_mangga, cv2.MORPH_OPEN, kernel)
mask_mangga = cv2.morphologyEx(mask_mangga, cv2.MORPH_CLOSE, kernel)
```

Berikutnya, fungsi dari kernel1 = np.ones((5,5), np.uint8) adalah membuat sebuah "kernel" berbentuk matriks persegi berukuran 5x5 yang berisi angka 1.

Kemudian mask_mangga = cv2.morphologyEx(mask_mangga, cv2.MORPH_OPEN, kernel1), menerapkan operasi "Opening" pada mask mangga. Operasi ini berfungsi untuk menghilangkan bintik-bintik kecil (noise) di latar belakang dan memisahkan objek yang saling menempel, mirip dengan "membuka" celah kecil.

Selanjutnya, mask_mangga = cv2.morphologyEx(mask_mangga, cv2.MORPH_CLOSE, kernel1) menerapkan operasi "Closing" pada mask yang sudah di-"Opening". Operasi Closing berfungsi mengisi lubang-lubang kecil di dalam objek dan menghaluskan batas objek, mirip dengan "menutup" celah.

```
segmentasi_mangga = cv2.bitwise_and(image_rgb, image_rgb, mask=mask_mangga)
```

Ini berfungsi untuk menampilkan hanya bagian gambar yang sesuai dengan area yang ditentukan oleh mask. Fungsi cv2.bitwise_and akan mengambil piksel dari image_rgb hanya pada area yang bernilai putih (255) pada mask_mangga, dan mengabaikan area lain (yang bernilai hitam atau 0).

```
lower_daun_hijau = np.array([25, 40, 40])
upper_daun_hijau = np.array([85, 255, 255])
mask_daun_hijau = cv2.inRange(hsv_image, lower_daun_hijau, upper_daun_hijau)
```

Lower dan upper daun hijau yaitu batas bawah dan batas atas warna hijau yang ingin dideteksi. Kemudian, fungsi cv2.inRange() digunakan untuk menghasilkan mask_daun_hijau yang menandai piksel-piksel dalam hsv_image yang termasuk dalam rentang warna tersebut.

```
lower_daun_kuning = np.array([15, 100, 100])
upper_daun_kuning = np.array([30, 255, 255])
mask_daun_kuning = cv2.inRange(hsv_image, lower_daun_kuning, upper_daun_kuning)
```

Lower dan upper daun kuning mendefinisikan rentang nilai HSV untuk warna kuning daun. Fungsi cv2.inRange() kemudian digunakan untuk membuat mask_daun_kuning dari hsv_image, yang akan bernilai putih (255) untuk piksel-piksel yang warnanya berada dalam rentang kuning tersebut, dan hitam (0) untuk piksel lainnya.

```
mask_daun_total = cv2.bitwise_or(mask_daun_hijau, mask_daun_kuning)
mask_daun_final = cv2.subtract(mask_daun_total, mask_mangga)
```

Selanjutnya, cv2.bitwise_or untuk menggabungkan dua masker, yaitu mask_daun_hijau dan mask_daun_kuning, sehingga membentuk satu masker gabungan mask_daun_total yang mencakup seluruh area daun (baik yang hijau maupun kuning).

Kemudian, cv2.subtract digunakan untuk mengurangi area buah mangga (mask_mangga) dari masker daun total, sehingga diperoleh mask_daun_final yang hanya berisi area daun tanpa bagian buah mangga. Proses ini membantu memisahkan objek daun secara bersih dari objek buah dalam gambar.

```
segmentasi_daun = cv2.bitwise_and(image_rgb, image_rgb, mask=mask_daun_final)
```

Ini berfungsi untuk mengekstrak area daun dari gambar berwarna (image_rgb) berdasarkan mask_daun_final. Fungsi cv2.bitwise_and akan menampilkan hanya bagian gambar yang ditandai putih (255) pada masker, yaitu area daun yang sudah dipisahkan dari buah mangga. Hasilnya adalah gambar yang hanya berisi daun, sedangkan bagian lain akan menjadi hitam atau transparan.

```
fig, axs = plt.subplots(2, 2, figsize=(12, 12))

axs[0, 0].imshow(image_rgb)
axs[0, 0].set_title('1. Citra Asli')
axs[0, 0].axis('off')

axs[0, 1].imshow(mask_mangga, cmap='gray')
axs[0, 1].set_title('2. Mask Mangga')
axs[0, 1].axis('off')

axs[1, 0].imshow(segmentasi_mangga)
axs[1, 0].set_title('3. Segmentasi Mangga')
axs[1, 0].axis('off')

axs[1, 1].imshow(segmentasi_daun)
axs[1, 1].set_title('4. Segmentasi Daun')
axs[1, 1].axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Ini membuat tampilan subplot berukuran 2x2 dengan ukuran gambar 12x12 inci. Masing-masing panel menampilkan hasil proses segmentasi: panel pertama menampilkan gambar asli (image_rgb), panel kedua menampilkan mask buah mangga (mask_mangga), panel ketiga hasil segmentasi mangga (segmentasi_mangga), dan panel keempat hasil segmentasi daun (segmentasi_daun). Setiap subplot diberi judul yang sesuai dan sumbu koordinatnya disembunyikan (axis off) agar fokus pada gambar. plt.tight_layout() digunakan untuk mengatur tata letak agar tidak saling tumpang tindih, dan plt.show() menampilkan keseluruhan hasil visualisasi tersebut.

3.2 FILTERING CITRA

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

Yang pertama sekali pastinya saya import library dengan cara import cv2, mengimpor pustaka OpenCV (Open Source Computer Vision Library) yang banyak digunakan untuk tugas-tugas penglihatan komputer seperti pemrosesan gambar dan video, import numpy as np, mengimpor pustaka NumPy, yang merupakan fondasi untuk komputasi numerik di Python dan sangat penting untuk bekerja dengan array dan matriks data, yang seringkali merepresentasikan gambar. Terakhir, import matplotlib.pyplot as plt mengimpor modul pyplot dari pustaka Matplotlib.

```python
def manual_mean_filter(image, kernel_size):
    """
    Menerapkan filter mean secara manual ke citra.

    Args:
        image (np.array): Citra input (grayscale).
        kernel_size (int): Ukuran kernel (harus ganjil).

    Returns:
        np.array: Citra hasil setelah penerapan filter mean.
    """
    if kernel_size % 2 == 0:
        raise ValueError("Ukuran kernel harus ganjil.")
```

Fungsi manual_mean_filter(image, kernel_size) bertujuan untuk menghaluskan gambar (grayscale) dengan menggunakan filter mean berbasis ukuran kernel yang ditentukan pengguna. Pada awal fungsi, terdapat pengecekan apakah kernel_size adalah bilangan ganjil, karena filter mean memerlukan ukuran ganjil agar memiliki titik pusat. Jika yang diberikan genap, maka akan muncul error.

```python
rows, cols = image.shape
padded_image = np.pad(image, kernel_size // 2, mode='reflect')
output_image = np.zeros_like(image, dtype=np.float32)
```

Pada bagian ini mengambil ukuran baris dan kolom dari citra input. Selanjutnya, np.pad digunakan untuk menambahkan padding ke citra dengan ukuran setengah dari kernel_size, menggunakan mode 'reflect' agar pinggiran gambar dipantulkan, sehingga hasil filter lebih halus di tepi.

Terakhir, output_image dibuat sebagai array kosong dengan ukuran dan bentuk yang sama seperti gambar input, namun bertipe float32, untuk menyimpan hasil akhir dari proses mean filtering.

```python
    for i in range(rows):
        for j in range(cols):
            kernel_area = padded_image[i:i+kernel_size, j:j+kernel_size]
            output_image[i, j] = np.mean(kernel_area)

    return output_image.astype(np.uint8)
```

Ini berfungsi untuk menerapkan filter rata-rata (averaging filter) pada sebuah gambar. Melalui dua loop bersarang, program ini mengunjungi setiap piksel dalam gambar masukan yang sudah diberi "padding". Untuk setiap piksel, ia mengekstrak area seukuran kernel_size di sekelilingnya (disebut kernel_area), lalu menghitung rata-rata nilai piksel di area tersebut. Hasil rata-rata ini kemudian ditetapkan sebagai nilai piksel yang sesuai di gambar keluaran. Setelah semua piksel diproses, gambar keluaran yang telah dihaluskan ini dikembalikan dengan memastikan tipenya adalah uint8 (nilai piksel 0-255). Singkatnya, kode ini melakukan penghalusan gambar secara manual untuk mengurangi detail dan noise dengan mengganti setiap piksel dengan rata-rata tetangganya.

```
image_path = 'natnat2.jpg'
original_image_bgr = cv2.imread(image_path)
```

Selanjutnya, image_path = 'natnat2.jpg', menetapkan nama file gambar yang akan dibaca. Kemudian menggunakan fungsi imread dari pustaka OpenCV (cv2) untuk membaca gambar yang ditentukan oleh image_path dan menyimpannya ke dalam variabel original_image_bgr, di mana gambar tersebut akan direpresentasikan dalam format warna BGR (Blue-Green-Red) standar OpenCV.

```
original_image_gray = cv2.cvtColor(original_image_bgr, cv2.COLOR_BGR2GRAY)
```

Ini berfungsi untuk mengambil gambar original_image_bgr yang kemungkinan besar dalam format BGR (Blue-Green-Red), lalu menggunakan fungsi cvtColor dari pustaka OpenCV untuk mengkonversinya menjadi representasi skala abu-abu, yang kemudian disimpan dalam variabel original_image_gray.

```
median_filtered_image_bgr = cv2.medianBlur(original_image_bgr, 5)
mean_filtered_image_gray = manual_mean_filter(original_image_gray, 5)
```

Berikutnya, ini menerapkan filter median pada gambar berwarna (original_image_bgr) dengan ukuran kernel 5x5. Filter median ini efektif dalam menghilangkan noise jenis "salt-and-pepper" sambil menjaga tepi objek.

Kemudian menerapkan filter rata-rata (mean filter) yang diimplementasikan secara manual (melalui fungsi manual_mean_filter) pada gambar skala abu-abu (original_image_gray) juga dengan ukuran kernel 5x5. Filter rata-rata ini berfungsi menghaluskan gambar dengan mengganti setiap piksel dengan rata-rata nilai piksel di sekitarnya, yang mengurangi noise dan membuat gambar sedikit kabur.

```
plt.figure(figsize=(12, 10))

plt.subplot(2, 2, 1)
plt.imshow(original_image_gray, cmap='gray')
plt.title('Citra Asli')

plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(median_filtered_image_bgr, cv2.COLOR_BGR2RGB))
plt.title('After filter median')

plt.subplot(2, 2, 3)
plt.imshow(original_image_gray, cmap='gray')
plt.title('Original Image')

plt.subplot(2, 2, 4)
plt.imshow(mean_filtered_image_gray, cmap='gray')
plt.title('Mean Filtered Image')

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

Ini berfungsi untuk menampilkan beberapa gambar secara bersamaan dalam satu jendela plot untuk perbandingan, menggunakan pustaka Matplotlib. Pertama, sebuah figur plot besar dibuat. Kemudian, kode membagi figur menjadi grid 2x2 dan menempatkan empat gambar berbeda: gambar asli dalam skala abu-abu, gambar yang telah dihaluskan menggunakan filter median (setelah konversi warna ke RGB), gambar asli skala abu-abu lagi (sepertinya duplikasi), dan gambar yang telah dihaluskan menggunakan filter rata-rata. Terakhir, plt.tight_layout() menyesuaikan tata letak agar tidak ada tumpang tindih, dan plt.show() menampilkan figur tersebut.

3.3 KOMPRESI

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

Yang pertama sekali pastinya saya import library dengan cara import cv2, mengimpor pustaka OpenCV (Open Source Computer Vision Library) yang banyak digunakan untuk tugas-tugas penglihatan komputer seperti pemrosesan gambar dan video, import numpy as np, mengimpor pustaka NumPy, yang merupakan fondasi untuk komputasi numerik di Python dan sangat penting untuk bekerja dengan array dan matriks data, yang seringkali merepresentasikan gambar. Terakhir, import matplotlib.pyplot as plt mengimpor modul pyplot dari pustaka Matplotlib.

```
image_path = 'natnat1.jpg'
image_bgr = cv2.imread(image_path)
```

Selanjutnya, image_path = 'natnat1.jpg', menetapkan nama file gambar yang akan dibaca. Kemudian menggunakan fungsi imread dari pustaka OpenCV (cv2) untuk membaca

gambar yang ditentukan oleh image_path dan menyimpannya ke dalam variabel original_image_bgr, di mana gambar tersebut akan direpresentasikan dalam format warna BGR (Blue-Green-Red) standar OpenCV.

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

Ini berfungsi untuk mengambil gambar image_bgr (yang diasumsikan dalam format BGR standar OpenCV) dan menggunakan fungsi cvtColor dari pustaka OpenCV (cv2) untuk mengubah urutan salurannya menjadi RGB, lalu menyimpannya ke dalam variabel image_rgb.

```
jpeg_quality = [cv2.IMWRITE_JPEG_QUALITY, 10]
result, encoded_image_buffer = cv2.imencode('.jpg', image_bgr, jpeg_quality)
```

Pada bagian menentukan parameter kualitas JPEG yang diinginkan, yaitu 10 (dari skala 0-100, di mana 10 berarti kualitas sangat rendah atau kompresi sangat tinggi). Selanjutnya menggunakan fungsi cv2.imencode untuk mengambil gambar image_bgr dan mengubahnya menjadi buffer byte yang terkompresi sebagai gambar JPEG, dengan menggunakan pengaturan kualitas yang sudah didefinisikan. Variabel result akan menunjukkan apakah proses pengkodean berhasil, dan encoded_image_buffer akan berisi data gambar yang sudah terkompresi.

```
lossy_compressed_bgr = cv2.imdecode(encoded_image_buffer, 1)
lossy_compressed_rgb = cv2.cvtColor(lossy_compressed_bgr, cv2.COLOR_BGR2RGB)
```

Berikutnya ini mengambil data gambar yang telah dikompresi sebelumnya (encoded_image_buffer) dan mengubahnya kembali menjadi representasi gambar yang bisa digunakan (format BGR), di mana angka 1 menunjukkan bahwa gambar akan dibaca dalam mode warna. Selanjutnya mengkonversi gambar yang baru saja didekode dari format BGR ke format RGB, yang seringkali lebih sesuai untuk ditampilkan atau diproses oleh pustaka lain.

```
divisor = 256 // levels
quantized_image_rgb = (image_rgb // divisor) * (255 // (levels - 1))
```

Ini berfungsi untuk menghitung nilai pembagi berdasarkan jumlah level warna yang diinginkan. Pembagi ini digunakan untuk mengelompokkan rentang nilai piksel.

Setiap nilai piksel dalam image_rgb (gambar asli dalam format RGB) dibagi dengan divisor, kemudian hasilnya dikalikan dengan faktor penskalaan untuk memetakan kembali nilai-nilai ke rentang 0-255 namun dengan jumlah level warna yang lebih sedikit.

```
original_size_kb = image_rgb.nbytes / 1024
lossy_size_kb = encoded_image_buffer.nbytes / 1024
quantized_size_kb = quantized_image_rgb.nbytes / 1024
```

Ini untuk menghitung ukuran gambar asli (image_rgb) dalam byte (.nbytes) lalu membaginya dengan 1024 untuk mendapatkan ukuran dalam KB. Kemudian melakukan hal yang sama untuk gambar yang telah dikompresi dengan metode lossy, di mana encoded_image_buffer adalah data gambar terkompresi dalam bentuk buffer. Dan yang terakhir menghitung ukuran gambar yang telah dikuantisasi warnanya. Intinya, kode ini digunakan untuk membandingkan seberapa efektif metode kompresi lossy dan kuantisasi warna dalam mengurangi ukuran data gambar dibandingkan dengan gambar aslinya.

```
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

axs[0].imshow(image_rgb)
axs[0].set_title(f'Asli\nUkuran: {original_size_kb:.2f} KB (In-Memory)')
axs[0].axis('off')

axs[1].imshow(lossy_compressed_rgb)
axs[1].set_title(f'Lossy JPEG Q=10\nUkuran: {lossy_size_kb:.2f} KB')
axs[1].axis('off')

axs[2].imshow(quantized_image_rgb)
axs[2].set_title(f'Kuantisasi RGB (4 Level)\nUkuran: {quantized_size_kb:.2f} KB (In-Memory)')
axs[2].axis('off')

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

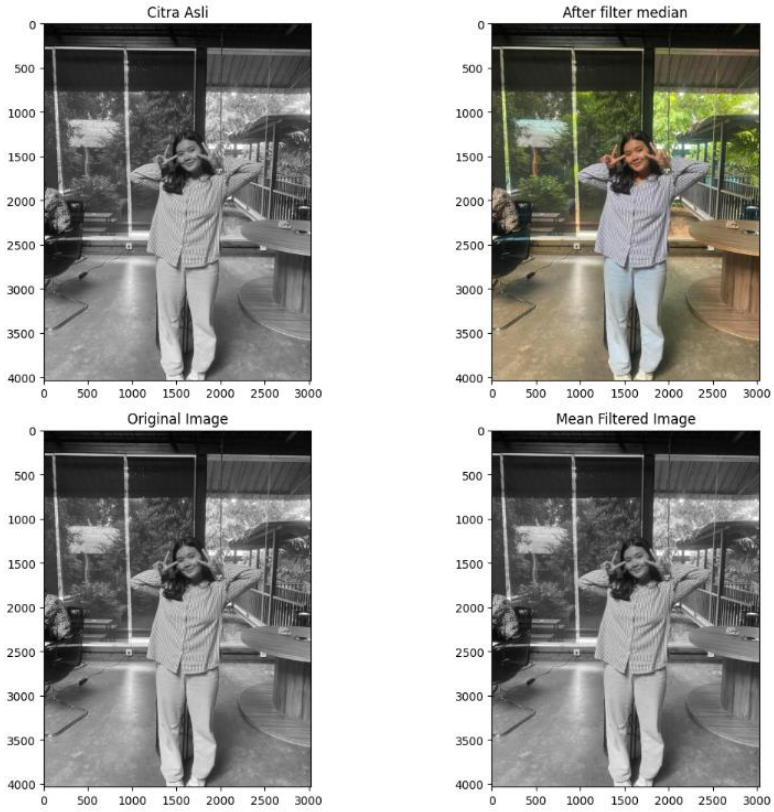
Terakhir ini untuk membuat sebuah figur plot yang dibagi menjadi tiga subplot sejajar. Setiap subplot kemudian menampilkan salah satu gambar (image_rgb, lossy_compressed_rgb, dan quantized_image_rgb) disertai judul yang menjelaskan jenis gambar dan ukuran memorinya. Fungsi axis('off') digunakan untuk menyembunyikan sumbu koordinat pada setiap gambar. Terakhir, plt.tight_layout() mengatur tata letak agar subplot tidak tumpang tindih, dan plt.show() menampilkan seluruh figur tersebut.

3.4 PENJELASAN OUTPUT

3.4.1 DETEKSI DAUN

- Citra asli
  Ini adalah gambar orisinal atau mentah, dalam kasus ini adalah foto pohon mangga dengan satu buah mangga yang menggantung. Citra ini mengandung semua informasi visual yang ada, termasuk warna, tekstur, dan detail dari seluruh adegan (pohon, daun, mangga, latar belakang).
- Mask Mangga
  Ini adalah representasi biner (hitam dan putih) yang digunakan untuk mengidentifikasi lokasi mangga dalam citra asli. Area putih pada mask menunjukkan piksel-piksel yang diidentifikasi sebagai bagian dari mangga, sementara area hitam menunjukkan semua piksel lainnya.
- Segmentasi Mangga
  Ini adalah hasil dari pengaplikasian "Mask Mangga" ke "Citra Asli". Dalam proses ini, hanya piksel-piksel dari citra asli yang sesuai dengan area putih pada "Mask Mangga" yang dipertahankan. Semua area lain (yang hitam pada mask) diubah menjadi hitam pada citra segmentasi.
- Segmentasi Daun
  Mirip dengan segmentasi mangga, ini adalah hasil dari pengaplikasian sebuah mask (yang tidak ditampilkan, tetapi diasumsikan ada dan menargetkan daun) ke "Citra Asli".

3.4.2 FILTERING CITRA

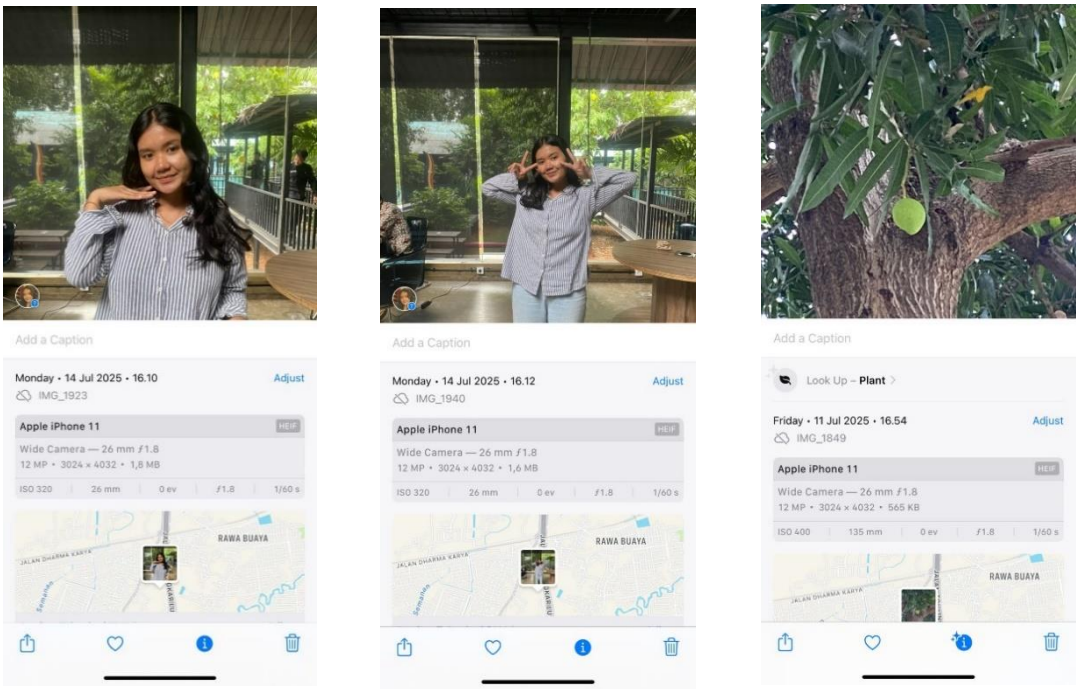


- Citra Asli / Original Image
  Keduanya adalah representasi gambar asli dalam skala abu-abu, berfungsi sebagai dasar perbandingan.
- After filter median
  Gambar yang telah dihaluskan menggunakan filter median, efektif mengurangi noise sambil menjaga ketajaman tepi, dan ditampilkan dalam warna.
- Mean Filtered Image
  Gambar yang telah dihaluskan menggunakan filter rata-rata, efektif mengurangi *noise* tetapi cenderung membuat gambar terlihat lebih buram dan tepi objek kurang tajam, ditampilkan dalam skala abu-abu.

3.4.3 KOMPRESI

- Asli
  Gambar lengkap dan utuh dengan kualitas terbaik, ukuran paling besar.
- Lossy
  Gambar terkompresi dengan kehilangan detail untuk mengecilkan ukuran file, kualitas visual menurun sebanding dengan tingkat kompresi.
- Kuantisasi RGB
  Mengurangi jumlah warna unik secara drastis, menciptakan efek blok warna atau posterisasi, tetapi ukuran data di memori bisa tetap sama jika tidak diikuti kompresi lebih lanjut.

3.5 BUKTI



BAB IV

PENUTUP

Dalam laporan ini, saya telah membahas berbagai aspek penting dalam pengolahan citra digital, termasuk segmentasi warna, ruang warna HSV, filtering warna, dan kompresi citra. Segmentasi warna merupakan teknik yang krusial untuk memisahkan objek dalam citra berdasarkan warna, yang dapat meningkatkan akurasi dalam pengenalan objek dan analisis citra. Penggunaan ruang warna HSV memberikan keuntungan signifikan dibandingkan ruang warna RGB, terutama dalam hal intuitivitas dan ketahanan terhadap variasi pencahayaan.

Filtering warna juga memainkan peran penting dalam meningkatkan kualitas citra, memungkinkan penekanan fitur-fitur tertentu yang relevan dalam berbagai aplikasi. Selain itu, pemahaman tentang kompresi citra, baik lossy maupun lossless, sangat penting untuk mengelola ukuran file dan kualitas citra, terutama dalam konteks penyimpanan dan transmisi data.

Melalui pemahaman yang lebih baik tentang teknik-teknik ini, diharapkan dapat meningkatkan kemampuan dalam menganalisis dan memproses citra digital secara efektif. Penelitian lebih lanjut dalam bidang ini dapat membuka peluang baru untuk inovasi dan aplikasi yang lebih luas dalam berbagai sektor, termasuk kesehatan, keamanan, dan industri kreatif. Dengan demikian, pengolahan citra digital tidak hanya menjadi alat yang kuat dalam analisis data visual, tetapi juga berkontribusi pada kemajuan teknologi dan peningkatan kualitas hidup.

DAFTAR PUSTAKA

[1] H. C. Kang, H. N. Han, H. C. Bae, M. G. Kim, J. Y. Son, and Y. K. Kim, “Hsv color-space-based automated object localization for robot grasping without prior knowledge,” *Applied Sciences (Switzerland)*, vol. 11, no. 16, Aug. 2021, doi: 10.3390/app11167593.

[2] C. Solomon and T. Breckon, “Fundamentals of Digital Image Processing A Practical Approach with Examples in Matlab.”

[3] Rasha Awad Abtan, “Image Enhancement using Adaptive Median Filter,” *Int J Sci Res Sci Eng Technol*, pp. 236–243, Aug. 2023, doi: 10.32628/ijsrset23102118.

[4] X. Wu, “Color Quantization Programming and by Dynamic Principal Analysis.”