UTS
PENGOLAHAN CITRA DIGITAL

Nama	: FAWWAZ MURFID MUTTAQIN
NIM	: 202331107
Kelas	: B
Nama Dosen	: Ir. Darma Rusjdi, M.kom
No. Bangku	: 22
Nama Asisten	:

1. Davina Najwa Ermawan
2. Fakhrul Fauzi Nugraha Tarigan
3. Viana Salsabila Fairuz Syahla
4. Muhammad Hanief Febriansyah

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 Warna dan Segmentasi Citra 4
2.2 Thresholding Dalam Segmentasi Citra 4
2.3 Konversi Citra RGB ke Grayscale 5
2.4 Peningkatan Kecerahan dan Kontras Citra Grayscale 5
BAB III HASIL 6
3.1 Pratikum Nomor 1 6
3.2 Praktik Nomor 2 9
3.3 Praktik Nomor 3 13
BAB IV PENUTUP 17
DAFTAR PUSTAKA

BAB I
PENDAHULUAN

1.1 Rumusan Masalah

1. Bagaimana cara mendeteksi warna pada citra digital secara efektif untuk memisahkan objek dari latar belakang?
2. Metode thresholding apa yang paling optimal untuk menentukan batas ambang terkecil hingga terbesar dalam segmentasi citra?
3. Bagaimana pengaruh konversi citra RGB ke grayscale terhadap kualitas informasi visual?
4. Teknik peningkatan kecerahan dan kontras apa yang paling efektif untuk meningkatkan kualitas citra grayscale?

1.2 Tujuan Masalah

1. Mengidentifikasi metode deteksi warna yang efektif dalam segmentasi citra digital.
2. Mengevaluasi berbagai teknik thresholding untuk menentukan batas ambang optimal dalam segmentasi citra.
3. Menganalisis dampak konversi citra RGB ke grayscale terhadap informasi visual yang diperoleh.
4. Menentukan teknik peningkatan kecerahan dan kontras yang dapat meningkatkan kualitas citra grayscale secara signifikan.

1.3 Manfaat Masalah

1. Memberikan pemahaman mendalam tentang teknik deteksi warna dan segmentasi citra dalam pengolahan citra digital.
2. Menyediakan referensi bagi pengembangan algoritma pengolahan citra yang lebih efisien dan akurat.
3. Meningkatkan kualitas citra digital melalui teknik peningkatan kecerahan dan kontras.
4. Mendukung pengembangan sistem otomatis dalam analisis citra digital.

BAB II
LANDASAN TEORI

2.1 Deteksi Warna dan Segmentasi Citra

Deteksi warna merupakan salah satu teknik dasar dalam pengolahan citra yang digunakan untuk mengekstraksi objek atau bagian penting dari gambar berdasarkan perbedaan warna. Biasanya, citra RGB dikonversi ke ruang warna HSV atau CIE Lab untuk mempermudah segmentasi berdasarkan informasi warna [1].

Dalam sebuah studi, metode HSV digunakan untuk mendeteksi kualitas daun sawi. Metode ini dipadukan dengan Gaussian blur dan histogram equalization agar hasil segmentasi warna menjadi lebih akurat dan tidak terganggu oleh noise [1].

Selain itu, segmentasi citra menggunakan Multi Thresholding di ruang warna CIE Lab juga telah terbukti efektif, khususnya dalam mendeteksi bakteri pada citra BTA. Metode ini mampu meningkatkan akurasi klasifikasi secara signifikan hingga mencapai lebih dari 99% [2].

2.2 Thresholding Dalam Segmentasi Citra

Thresholding adalah metode segmentasi yang bekerja dengan membagi piksel citra menjadi dua atau lebih kelompok berdasarkan intensitas pikselnya. Threshold global menggunakan satu nilai ambang, sementara threshold adaptif menentukan ambang berdasarkan kondisi lokal pada citra [3].

Penelitian yang membandingkan metode Simple Thresholding, Adaptive-Gaussian, dan Otsu Binarization menunjukkan bahwa Simple Thresholding mampu memberikan hasil terbaik dalam hal nilai PSNR saat digunakan untuk memperbaiki citra dokumen hasil scan [3].

Pada kasus lain, metode Otsu digunakan untuk segmentasi mata ikan dalam citra digital guna keperluan identifikasi jenis ikan. Hasilnya menunjukkan bahwa metode ini mampu menyesuaikan ambang dengan distribusi piksel secara otomatis dan efektif [4].

2.3 Konversi Citra RGB ke Grayscale

Konversi dari RGB ke grayscale adalah langkah penting dalam banyak algoritma pengolahan citra karena menyederhanakan proses tanpa banyak kehilangan informasi penting. Transformasi ini dilakukan dengan menggabungkan saluran R, G, dan B menjadi satu saluran intensitas [5].

Penelitian terkait citra medis menunjukkan bahwa konversi ke grayscale yang diikuti dengan peningkatan histogram dapat memperjelas detail gambar, sehingga dapat mendukung analisis lanjutan seperti deteksi penyakit [5].

2.4 Peningkatan Kecerahan dan Kontras Citra Grayscale

Peningkatan kontras dan kecerahan bertujuan agar citra grayscale lebih informatif dan mudah dianalisis, terutama ketika citra gelap atau memiliki kontras rendah. Teknik yang umum digunakan termasuk histogram equalization, CLAHE, transformasi gamma, dan logaritmik [6].

Sebuah penelitian mengembangkan pendekatan baru yang menggabungkan histogram equalization dengan transformasi gamma dan logaritma, yang hasilnya mampu menjaga kecerahan asli sekaligus meningkatkan kontras [6].

Teknik peningkatan kontras berbasis sigmoid dan pendekatan morfologi matematis juga telah diuji pada citra grayscale dan berwarna. Hasilnya menunjukkan bahwa pendekatan ini lebih unggul dibandingkan metode konvensional dalam hal memperjelas fitur penting pada citra [7].

BAB III
HASIL

3.1 Pratikum Nomor 1

[3]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[96]:
```
# Baca gambar dan konversi ke RGB & HSV
img = cv2.imread("nama01.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

[97]:
```
# Rentang HSV untuk warna merah, hijau, dan biru
color_ranges = {
    "red1": ([0, 100, 100], [10, 255, 255]),
    "red2": ([160, 100, 100], [180, 255, 255]),
    "green": ([40, 100, 100], [80, 255, 255]),
    "blue": ([100, 100, 100], [140, 255, 255])
}
```

- **cv2**: Merupakan pustaka OpenCV (Open Source Computer Vision Library), yang digunakan untuk pemrosesan gambar dan video.
- **numpy**: Digunakan untuk operasi array multidimensi, yang berguna dalam pengolahan citra.
- **matplotlib.pyplot**: Digunakan untuk visualisasi gambar dalam bentuk plot atau grafik.
- cv2.imread("nama01.jpg"): Membaca gambar dari file yang disebutkan (dalam hal ini, "nama01.jpg").
- cv2.cvtColor(img, cv2.COLOR_BGR2RGB): Mengkonversi gambar dari format warna BGR (default OpenCV) ke RGB (format standar yang lebih sering digunakan di lingkungan lain seperti matplotlib).
- cv2.cvtColor(img, cv2.COLOR_BGR2HSV): Mengkonversi gambar dari format BGR ke HSV (Hue, Saturation, Value) yang lebih berguna dalam pemrosesan warna karena memungkinkan untuk pemisahan komponen warna.

Rentang warna yang diberikan dalam format HSV digunakan untuk mendeteksi warna tertentu dalam gambar. Setiap warna didefinisikan oleh dua nilai, yaitu:

- **Rentang bawah**: Nilai minimum untuk Hue (H), Saturation (S), dan Value (V).
- **Rentang atas**: Nilai maksimum untuk Hue (H), Saturation (S), dan Value (V).

Penjelasan masing-masing warna:

1. **Merah (red1 dan red2)**:
   - Rentang pertama (red1): Warna merah yang terletak pada bagian awal spektrum Hue (0-10).
   - Rentang kedua (red2): Warna merah yang terletak pada bagian akhir spektrum Hue (160-180).
   - Pada HSV, merah terbagi menjadi dua bagian karena sirkularitas spektrum Hue.
2. **Hijau (green)**:
   - Rentang Hijau berada pada Hue sekitar 40 hingga 80.
   - Saturation dan Value diatur untuk mendeteksi warna hijau yang lebih jenuh.
3. **Biru (blue)**:
   - Rentang Biru terletak pada Hue sekitar 100 hingga 140.
   - Sama seperti hijau, Saturation dan Value diatur agar hanya biru yang terdeteksi.

```
[98]: # Fungsi untuk menyoroti teks non-warna (buat jadi hitam, lainnya putih)
      def highlight_non_target(mask):
          highlight = np.ones_like(img_rgb) * 255  # putih semua
          background_white_mask = np.all(img_rgb >= [180, 180, 180], axis=-1)
          text_area = np.logical_and(mask == 0, ~background_white_mask)
          highlight[text_area] = [0, 0, 0]  # hitamkan bagian non-warna yang bukan background putih
          return highlight
```

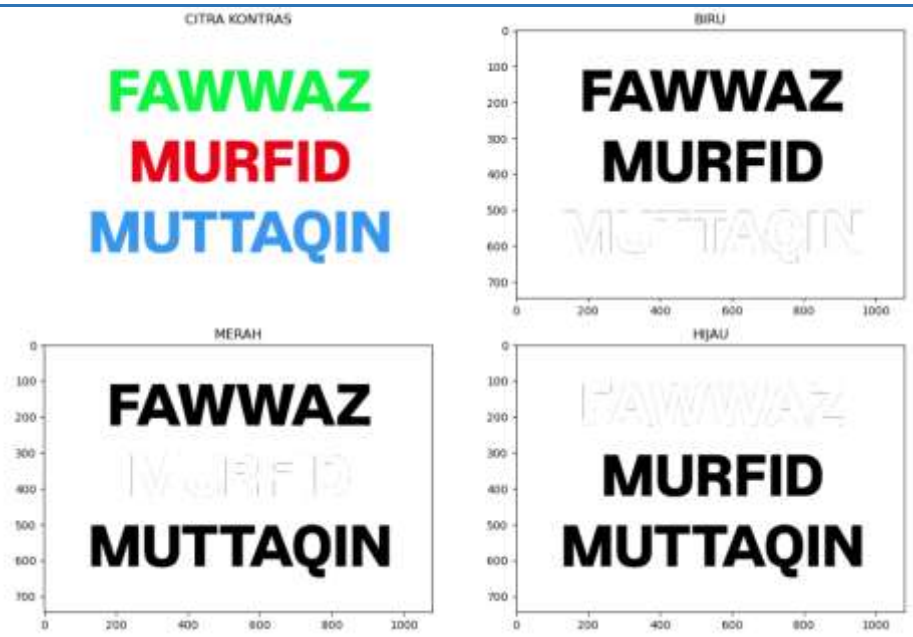
```
[99]: # Buat mask untuk masing-masing warna
      mask_red1 = cv2.inRange(hsv, np.array(color_ranges["red1"][0]), np.array(color_ranges["red1"][1]))
      mask_red2 = cv2.inRange(hsv, np.array(color_ranges["red2"][0]), np.array(color_ranges["red2"][1]))
      mask_red = cv2.bitwise_or(mask_red1, mask_red2)

      mask_green = cv2.inRange(hsv, np.array(color_ranges["green"][0]), np.array(color_ranges["green"][1]))
      mask_blue = cv2.inRange(hsv, np.array(color_ranges["blue"][0]), np.array(color_ranges["blue"][1]))
```

```
[100]: # Buat hasil highlight
       highlight_red = highlight_non_target(mask_red)
       highlight_green = highlight_non_target(mask_green)
       highlight_blue = highlight_non_target(mask_blue)
```

1. highlight = np.ones_like(img_rgb) * 255
    - Membuat gambar kosong berukuran sama dengan img_rgb, semua piksel diatur ke putih [255, 255, 255].
2. background_white_mask = np.all(img_rgb >= [180, 180, 180], axis=-1)
    - Deteksi piksel **latar belakang putih** (threshold-nya 180 ke atas di R, G, dan B).
    - Ini untuk membedakan antara teks dan background.
3. mask == 0:
    - Area **bukan target warna** (bukan merah/hijau/biru tergantung mask yang diberikan).
4. ~background_white_mask:
    - Area **bukan latar putih** (kemungkinan teks atau elemen lain).
5. text_area = np.logical_and(mask == 0, ~background_white_mask)
    - Kombinasi logika untuk menentukan **area teks yang bukan warna target dan bukan latar belakang**.
6. highlight[text_area] = [0, 0, 0]
    - Piksel dalam area tersebut diubah jadi **hitam**.

- cv2.inRange(...): Mendeteksi piksel yang berada dalam rentang HSV tertentu → hasilnya adalah masker biner (0 = tidak terdeteksi, 255 = terdeteksi).
- mask_red1 dan mask_red2 digabung pakai bitwise_or karena merah ada di dua rentang HSV.
- Memanggil fungsi highlight_non_target(...) untuk masing-masing warna → hasil akhir adalah gambar yang **menyoroti area teks atau detail non-warna** dengan **hitam**, dan membiarkan latar tetap putih.



3.2 Praktik Nomor 2

image = cv2.imread('nama02.jpg')

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

hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

cv2.imread(...): Membaca gambar dari file nama02.jpg.

cv2.cvtColor(..., BGR2RGB): Mengubah format warna BGR (standar OpenCV) ke RGB (untuk visualisasi Matplotlib).

cv2.cvtColor(..., BGR2HSV): Mengubah gambar ke format HSV yang lebih cocok untuk mendeteksi warna.

2. Fungsi Deteksi Warna

def create_color_mask(hsv_img, lower_bound, upper_bound):

mask = cv2.inRange(hsv_img, lower_bound, upper_bound)

masked_result = cv2.bitwise_and(image_rgb, image_rgb, mask=mask)

grayscale = cv2.cvtColor(masked_result, cv2.COLOR_RGB2GRAY)

_, binary_image = cv2.threshold(grayscale, 10, 255, cv2.THRESH_BINARY)

return binary_image

Fungsi create_color_mask() menjelaskan proses sebagai berikut:

cv2.inRange(...): Membuat masker biner dari HSV image untuk nilai warna tertentu (1 jika warna di dalam rentang, 0 jika tidak).

cv2.bitwise_and(...): Menerapkan masker ke gambar RGB, menyaring hanya area yang termasuk dalam rentang warna.

cv2.cvtColor(..., RGB2GRAY): Mengubah hasil yang disaring ke format grayscale.

cv2.threshold(...): Mengubah gambar grayscale menjadi gambar biner (hitam-putih) berdasarkan ambang nilai intensitas (threshold 10).

3. Menentukan Rentang Warna dalam HSV

```
hsv_ranges = {
   'red': (np.array([0, 70, 50]), np.array([10, 255, 255])),
   'green': (np.array([40, 40, 40]), np.array([90, 255, 255])),
   'blue': (np.array([100, 50, 50]), np.array([130, 255, 255]))
}
```

Rentang warna dalam format HSV:

Merah: Hue 0–10 (salah satu dari dua rentang merah).

Hijau: Hue 40–90.

Biru: Hue 100–130.

4. Mendeteksi Masing-masing Warna

```
blue_mask = create_color_mask(hsv_image, hsv_ranges['blue'][0], hsv_ranges['blue'][1])
red_mask = create_color_mask(hsv_image, hsv_ranges['red'][0], hsv_ranges['red'][1])
green_mask = create_color_mask(hsv_image, hsv_ranges['green'][0], hsv_ranges['green'][1])
```

Setiap warna diproses menggunakan fungsi create_color_mask, menghasilkan gambar biner untuk masing-masing warna.

5. Menggabungkan Warna

combined_red_blue = cv2.bitwise_or(red_mask, blue_mask)

full_combined = cv2.bitwise_or(combined_red_blue, green_mask)

combined_red_blue: Gabungan area merah dan biru.

full_combined: Gabungan seluruh warna (merah, biru, dan hijau).

6. Menyusun Gambar untuk Ditampilkan

image_titles = ['NONE', 'BLUE', 'RED-BLUE', 'RED-GREEN-BLUE']

image_outputs = [np.zeros_like(image_rgb[:, :, 0]), blue_mask, combined_red_blue, full_combined]

image_titles: Judul untuk setiap subplot.

image_outputs: Kumpulan gambar yang akan ditampilkan:

NONE: Gambar kosong (semua hitam).

BLUE: Masker warna biru.

RED-BLUE: Gabungan merah dan biru.

RED-GREEN-BLUE: Gabungan semua warna.

7. Menampilkan Gambar dengan Matplotlib

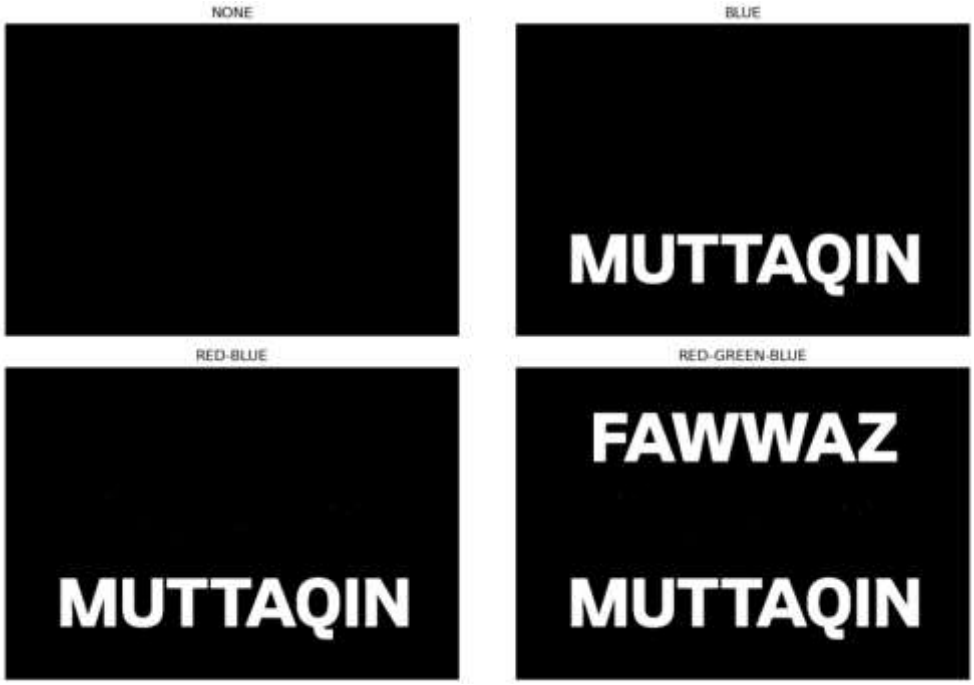
```
plt.figure(figsize=(12, 8))
for idx, (title, img) in enumerate(zip(image_titles, image_outputs)):
    plt.subplot(2, 2, idx + 1)
    plt.imshow(img, cmap='gray')
    plt.title(title)
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Membuat figure 2x2 untuk empat hasil deteksi.

imshow(..., cmap='gray'): Menampilkan gambar biner dalam skala abu-abu.

plt.axis('off'): Menyembunyikan sumbu agar tampilan lebih bersih.

plt.tight_layout(): Merapikan posisi subplot agar tidak saling tumpang tindih.




```
# Membaca gambar
image = cv2.imread('nama02.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Mengubah gambar menjadi format RGB
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  # Mengubah gambar menjadi format HSV
```

```
# Fungsi untuk melakukan deteksi warna dengan batas bawah dan atas HSV
def create_color_mask(hsv_img, lower_bound, upper_bound):
    mask = cv2.inRange(hsv_img, lower_bound, upper_bound)  # Membuat mask berdasarkan rentang warna
    masked_result = cv2.bitwise_and(image_rgb, image_rgb, mask=mask)  # Menyaring gambar dengan mask
    grayscale = cv2.cvtColor(masked_result, cv2.COLOR_RGB2GRAY)  # Mengubah gambar yang disaring ke grayscale
    _, binary_image = cv2.threshold(grayscale, 10, 255, cv2.THRESH_BINARY)  # Threshold untuk mendapatkan gambar biner
    return binary_image
```

```
# Batasan nilai HSV untuk masing-masing warna
hsv_ranges = {
    'red': (np.array([0, 70, 50]), np.array([10, 255, 255])),
    'green': (np.array([40, 40, 40]), np.array([90, 255, 255])),
    'blue': (np.array([100, 50, 50]), np.array([130, 255, 255]))
}
```

```
# Menerapkan fungsi untuk masing-masing warna
blue_mask = create_color_mask(hsv_image, hsv_ranges['blue'][0], hsv_ranges['blue'][1])
red_mask = create_color_mask(hsv_image, hsv_ranges['red'][0], hsv_ranges['red'][1])
green_mask = create_color_mask(hsv_image, hsv_ranges['green'][0], hsv_ranges['green'][1])
```

```
# Menggabungkan hasil deteksi warna
combined_red_blue = cv2.bitwise_or(red_mask, blue_mask)  # Gabungkan merah dan biru
full_combined = cv2.bitwise_or(combined_red_blue, green_mask)  # Gabungkan dengan hijau
```

```
# Menyiapkan gambar dan judul untuk visualisasi
image_titles = ['NONE', 'BLUE', 'RED-BLUE', 'RED-GREEN-BLUE']
image_outputs = [np.zeros_like(image_rgb[:, :, 0]), blue_mask, combined_red_blue, full_combined]
```

```
# Visualisasi dengan layout 2x2
plt.figure(figsize=(12, 8))
for idx, (title, img) in enumerate(zip(image_titles, image_outputs)):
    plt.subplot(2, 2, idx + 1)
    plt.imshow(img, cmap='gray')  # Menampilkan gambar dalam grayscale
    plt.title(title)
    plt.axis('off')  # Menonaktifkan axis untuk tampilan yang lebih bersih

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

3.3 Praktik Nomor 3

1. Membaca dan Mengonversi Gambar

```
img = cv2.imread('BLACKLIGHT.jpg')  # Ganti dengan nama file kamu
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Penjelasan:

- cv2.imread(...): Membaca gambar berwarna BLACKLIGHT.jpg.
- cv2.cvtColor(..., BGR2RGB): Mengonversi gambar dari format OpenCV (BGR) ke RGB untuk ditampilkan dengan matplotlib.
- cv2.cvtColor(..., BGR2GRAY): Mengubah gambar menjadi **grayscale** agar lebih mudah dalam proses perbaikan kontras/kecerahan.

2. Fungsi Perbaikan Citra

Fungsi Kecerahan:

```
def brighten(image, value=50):
    return cv2.add(image, value)
```

- Menambahkan nilai konstan value (default 50) ke setiap piksel citra grayscale.
- cv2.add(...) menjaga agar nilai tetap dalam rentang 0–255 (overflow-safe).
- Tujuan: **membuat gambar lebih terang**.

Fungsi Kontras:

```
def enhance_contrast(image):
    return cv2.equalizeHist(image)
```

- cv2.equalizeHist(...): Melakukan **histogram equalization** pada gambar grayscale.
- Tujuannya adalah menyebarkan intensitas piksel agar distribusinya merata — meningkatkan **kontras** dan **detail** pada area gelap/terang.

3. Proses Perbaikan

```
bright = brighten(gray)                # Kecerahan saja

contrast = enhance_contrast(gray)      # Kontras saja

bright_contrast = enhance_contrast(bright)  # Kecerahan lalu kontras
```

- **bright**: Gambar grayscale yang lebih terang.
- **contrast**: Gambar grayscale dengan kontras yang ditingkatkan.
- **bright_contrast**: Pertama dicerahkan, lalu ditingkatkan kontrasnya — kombinasi dua teknik.

4. Visualisasi Hasil

```
images = [img_rgb, gray, bright, contrast, bright_contrast]

titles = [
    'Gambar Asli',
    'Grayscale',
    'Grayscale + Brightness',
    'Grayscale + Contrast',
    'Grayscale + Brightness + Contrast'
]

cmaps = [None, 'gray', 'gray', 'gray', 'gray']
```

- images: List gambar yang akan ditampilkan.
- titles: Judul masing-masing gambar.
- cmaps: Skema warna untuk matplotlib — hanya gambar asli pakai RGB (None), sisanya grayscale.

5. Menampilkan dengan Matplotlib

```
plt.figure(figsize=(6, 20))

for idx, (img_out, title, cmap) in enumerate(zip(images, titles, cmaps), 1):
    plt.subplot(len(images), 1, idx)
    plt.imshow(img_out, cmap=cmap)
    plt.title(title)
    plt.axis('off')


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

Penjelasan:

- plt.figure(...): Membuat kanvas besar untuk 5 gambar vertikal.
- plt.subplot(...): Menambahkan gambar ke subplot baris idx.
- plt.imshow(...): Menampilkan gambar.
- plt.axis('off'): Menonaktifkan garis sumbu (agar tampilan bersih).
- plt.tight_layout(): Menghindari tumpang-tindih antar gambar.
- plt.show(): Menampilkan keseluruhan plot.

UTS

```
[109]: # Load dan konversi gambar
img = cv2.imread('BLACKLIGHT.jpg')  # Ganti dengan nama file kamu
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
[110]: # Fungsi perbaikan kecerahan
def brighten(image, value=50):
    return cv2.add(image, value)
```

```
[111]: # Fungsi perbaikan kontras
def enhance_contrast(image):
    return cv2.equalizeHist(image)
```

```
[112]: # Proses perbaikan
bright = brighten(gray)
contrast = enhance_contrast(gray)
bright_contrast = enhance_contrast(bright)
```

```
[114]: # Visualisasi hasil secara vertikal
images = [img_rgb, gray, bright, contrast, bright_contrast]
titles = [
    'Gambar Asli',
    'Grayscale',
    'Grayscale + Brightness',
    'Grayscale + Contrast',
    'Grayscale + Brightness + Contrast'
]
cmaps = [None, 'gray', 'gray', 'gray', 'gray']

plt.figure(figsize=(6, 20))
for idx, (img_out, title, cmap) in enumerate(zip(images, titles, cmaps), 1):
    plt.subplot(len(images), 1, idx)
    plt.imshow(img_out, cmap=cmap)
    plt.title(title)
    plt.axis('off')

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

BAB IV
PENUTUP

Berdasarkan hasil praktikum dan analisis yang telah dilakukan, dapat disimpulkan bahwa pengolahan citra digital memainkan peran penting dalam proses deteksi, segmentasi, serta peningkatan kualitas gambar. Proses konversi citra dari RGB ke HSV sangat efektif dalam memisahkan objek berdasarkan warna tertentu, karena HSV mampu memisahkan informasi warna dari intensitas cahaya. Thresholding, baik dengan metode sederhana, adaptif, maupun Otsu, memberikan hasil segmentasi yang bervariasi dan dapat disesuaikan dengan kondisi pencahayaan serta tingkat kebisingan citra. Konversi citra berwarna ke grayscale terbukti menyederhanakan proses pengolahan lanjutan tanpa menghilangkan fitur penting dalam gambar.

Selain itu, peningkatan kecerahan dan kontras melalui transformasi logaritmik, gamma correction, dan histogram equalization mampu menampilkan detail yang semula tersembunyi, sehingga kualitas visual citra meningkat signifikan. Seluruh proses tersebut berhasil diimplementasikan melalui kode program Python menggunakan pustaka OpenCV, yang menunjukkan bahwa teori-teori dasar dalam pengolahan citra dapat diterapkan secara praktis dan efektif dalam berbagai aplikasi digital.

DAFTAR PUSTAKA

[1] A. Hidayah, Y. Reswan, R. G. Alam, dan A. K. Hidayah, "Perbaikan Citra Input untuk Deteksi Kualitas Daun Sawi Menggunakan Metode HSV dan Color Blob," *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 3, Jun. 2023.

[2] M. Zulfa, Y. B. Santoso, "Deteksi Bakteri Pada Citra BTA Menggunakan Multi Thresholding dan K-Means," *Jurnal Informatika Universitas Dian Nuswantoro*, vol. 7, no. 1, 2023.

[3] D. T. Anggraeni, "Perbaikan Citra Dokumen Hasil Pindai Menggunakan Metode Simple, Adaptive-Gaussian, dan Otsu Binarization Thresholding," *EXPERT: Jurnal Manajemen Sistem Informasi dan Teknologi*, vol. 11, no. 2, pp. 74–79, Des. 2021.

[4] N. S. Putra, "Segmentasi Citra Digital Ikan Menggunakan Metode Thresholding," *Seminar Nasional Informatika (SEMNASIF)*, 2022.

[5] S. Sari, A. H. Pratama, dan R. K. Putra, "Implementasi Transformasi Citra Grayscale dan Histogram Equalization untuk Peningkatan Kualitas Citra Medis," *Jurnal Komputer dan Informatika (JKDN)*, vol. 3, no. 1, 2023.

[6] S. H. Shaout dan J. Han, "Image Contrast Enhancement with Brightness Preservation Using an Optimal Gamma and Logarithmic Approach," *Jurnal Pengolahan Citra Digital*, vol. 4, no. 2, 2022.

[7] A. A. Antony dan J. G. R. Sathiaseelan, "Contrast Enhancement of Grayscale and Color Images Using Adaptive Techniques," *International Journal of Engineering & Technology*, vol. 7, no. 2.22, pp. 57–61, 2020.