UTS

PENGOLAHAN CITRA

NAMA : Selma Zetapriana

NIM : 202331301

KELAS : E

DOSEN : Darma Rusjdi, Ir., M.Kom.

NO.PC :

ASISTEN : 1. Fauzan Arroyyan

2. Abdur Rasyid Ridho

INSTITUT TEKNOLOGI PLN

TEKNIK INFORMATIKA

2024/2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Rumusan Masalah

1. Bagaimana cara mendeteksi warna merah, hijau, dan biru dalam sebuah citra menggunakan Jupyter Notebook?
2. Bagaimana menentukan nilai ambang batas (threshold) terkecil hingga terbesar untuk mengelompokkan warna pada citra?
3. Bagaimana memperbaiki kualitas gambar backlight agar objek utama menjadi lebih terlihat dan menjadi fokus utama dalam citra?
4. Bagaimana cara menganalisis histogram dan hasil deteksi warna untuk mendapatkan informasi visual yang lebih jelas?

1.2 Tujuan Masalah

1. Menerapkan teknik deteksi warna pada gambar pribadi dengan menggunakan Jupyter Notebook.
2. Menentukan nilai ambang batas yang sesuai untuk segmentasi warna pada citra digital.
3. Mengolah dan memperbaiki gambar yang mengalami backlight agar bagian objek utama lebih terang dan kontras.
4. Menganalisis hasil deteksi warna dan histogram untuk menilai kualitas pengolahan citra yang telah dilakukan.

1.3 Manfaat Masalah

1. Mahasiswa mampu memahami dan mengaplikasikan teknik dasar dalam pengolahan citra digital secara praktis.
2. Menambah keterampilan mahasiswa dalam menggunakan Jupyter Notebook untuk pemrosesan gambar.
3. Meningkatkan kemampuan analisis visual dalam menilai kualitas dan efektivitas pengolahan gambar.
4. Membantu dalam mengenali pentingnya segmentasi warna dan perbaikan kualitas citra dalam berbagai kebutuhan visual atau teknologi.

BAB II

LANDASAN TEORI

2.1 Deteksi Warna Pada Citra

Deteksi warna adalah teknik dalam pengolahan citra digital yang digunakan untuk mengenali atau mengekstraksi area gambar berdasarkan warna tertentu. Proses ini umumnya dilakukan dengan mengubah ruang warna gambar agar warna lebih mudah dibedakan terlepas dari pencahayaan. Deteksi warna berguna dalam berbagai aplikasi seperti pelacakan objek, segmentasi citra, pengenalan benda, serta sistem navigasi robotik, karena memungkinkan sistem untuk memahami dan merespons elemen visual berdasarkan karakteristik warnanya.

Langkah pertama adalah memperoleh citra yang ingin dianalisis. Citra bisa diperoleh menggunakan kamera atau perangkat lain seperti scanner. Pastikan kondisi pencahayaan di sekitar objek yang akan dipotret dapat mendukung deteksi warna yang akurat.

Ini adalah contoh dari pengambilan citra menggunakan kamera handphone. Citra yang diperoleh biasanya dalam format RGB (Red, Green, Blue), namun untuk mendeteksi warna secara lebih efektif, citra tersebut sering kali dikonversi ke model warna lain, pada laporan ini citra yang telah diambil dikonversi ke BGR (Blue, Green, Red). Setelah citra dikonversi ke model warna yang sesuai, langkah berikutnya adalah memisahkan (mengekstrak) warna yang diinginkan dari citra tersebut. Ini bisa dilakukan dengan cara mendefinisikan rentang warna yang ingin dideteksi. Kemudian untuk mendeteksi warna tertentu dalam citra, kita dapat menggunakan teknik filter atau masking. Di sini, citra yang telah dikonversi akan dimanipulasi agar hanya piksel yang berada dalam rentang warna yang diinginkan yang tetap terlihat, sementara yang lainnya akan dihilangkan (misalnya, diubah menjadi hitam atau transparan). Setelah proses penyaringan, analisis dapat dilakukan untuk menentukan jumlah area yang terdeteksi, proporsi warna dalam citra, atau bahkan melakukan pelacakan objek berwarna tersebut. Misalnya, jika kita ingin mengetahui seberapa banyak objek berwarna merah dalam citra, kita bisa menghitung jumlah piksel yang terdeteksi oleh mask yang telah dibuat. Dalam Jupyter Notebook, analisis dilakukan menggunakan histogram, yang menampilkan seberapa sering intensitas warna tertentu muncul di gambar.

2.2 Ambang Batas (Threshold)

Untuk menentukan nilai ambang batas (threshold) terkecil hingga terbesar dalam mengelompokkan warna pada citra, langkah pertama yang dapat dilakukan adalah dengan mengambil sampel nilai warna dari objek yang ingin dikenali. Ini bisa dilakukan secara manual menggunakan alat bantu seperti OpenCV, color picker, atau tools online untuk memperoleh nilai HSV atau RGB dari piksel tertentu. Setelah nilai-nilai warna dari objek dikumpulkan, threshold ditentukan dari nilai minimum dan maksimum masing-masing kanal warna, seperti Hue, Saturation, dan Value jika menggunakan model warna HSV. Selain itu, analisis histogram warna juga bisa digunakan untuk mengetahui sebaran nilai warna yang dominan pada objek, sehingga rentang warna dapat ditentukan dengan lebih akurat. Pendekatan lainnya adalah dengan membuat program interaktif menggunakan OpenCV yang memungkinkan pengguna mengklik pada bagian gambar untuk mengetahui nilai HSV piksel tersebut. Nilai-nilai ini kemudian digunakan untuk menentukan rentang ambang batas warna yang diinginkan. Namun, penting untuk mempertimbangkan faktor pencahayaan, karena warna dapat tampak berbeda di bawah kondisi terang dan redup. Oleh karena itu, perlu dilakukan sampling dari berbagai kondisi cahaya dan memperluas rentang threshold untuk menjaga akurasi dalam segmentasi warna.

2.3 Memperbaiki Gambar Backlight dengan Jupyter Notebook

Secara umum, untuk memperbaiki kualitas gambar yang mengalami backlight agar objek utama terlihat lebih jelas dan menjadi fokus utama dalam citra, langkah pertama yang dilakukan adalah dengan meningkatkan kecerahan gambar. Peningkatan kecerahan bertujuan untuk menonjolkan detail pada area gelap yang biasanya merupakan bagian dari objek utama. Setelah itu, peningkatan kontras dilakukan untuk memperjelas perbedaan antara objek dan latar belakang sehingga objek terlihat lebih menonjol. Proses ini dilakukan dengan menyesuaikan nilai-nilai piksel pada gambar, di mana nilai piksel yang lebih rendah dinaikkan tanpa membuat bagian terang menjadi terlalu putih.

Dalam Jupyter Notebook, langkah pertama adalah membaca gambar dalam format RGB, kemudian dilakukan penyesuaian kecerahan dengan menambahkan nilai tertentu (misalnya beta = 40) pada setiap piksel citra agar bagian gelap menjadi lebih terang. Selanjutnya, untuk meningkatkan ketajaman perbedaan antara objek dan latar belakang, kontras ditingkatkan dengan mengalikan setiap nilai piksel dengan faktor alpha > 1 (misalnya alpha = 1.5). Proses ini dapat ditulis dalam Jupyter Notebook menggunakan operasi output = alpha * image + beta, dan hasilnya dipastikan tetap dalam rentang 0–255. Dengan kombinasi penyesuaian ini, objek utama yang sebelumnya gelap karena backlight akan tampak lebih jelas dan menjadi fokus utama dalam citra.

2.4 Analisis Hasil Deteksi Warna dengan Histogram

Secara umum, histogram dalam pengolahan citra adalah alat yang digunakan untuk menggambarkan distribusi intensitas warna atau kecerahan dalam sebuah gambar. Dengan histogram, kita bisa melihat seberapa merata distribusi warna atau kecerahan di seluruh gambar, yang dapat membantu kita memahami kontras, pencahayaan, dan dominasi warna tertentu dalam citra. Jika histogram terpusat di sisi kiri, ini menunjukkan bahwa gambar cenderung gelap, sementara jika terpusat di sisi kanan, gambar cenderung terang. Analisis histogram ini penting untuk melakukan koreksi atau penyesuaian pada gambar, seperti mengubah tingkat kecerahan, kontras, atau mengoptimalkan segmentasi warna.

Dalam Jupyter Notebook, histogram sangat berguna untuk menganalisis sebaran intensitas warna atau kecerahan pada citra. Dengan melihat grafik histogram, kita dapat menilai bagaimana intensitas piksel tersebar pada gambar dan mengidentifikasi area yang perlu diperbaiki, seperti daerah gelap atau terang yang berlebihan. Histogram juga sering digunakan untuk mengevaluasi hasil deteksi warna, apakah objek yang terdeteksi sesuai dengan ambang batas yang telah ditentukan, dan seberapa baik gambar tersebut menunjukkan warna yang dominan atau keseimbangan warna dalam citra. Dengan kata lain, histogram memberikan informasi visual yang jelas tentang kualitas dan karakteristik gambar, serta memberikan petunjuk untuk perbaikan lebih lanjut.

BAB III

HASIL

IMPORT LIBRARY

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

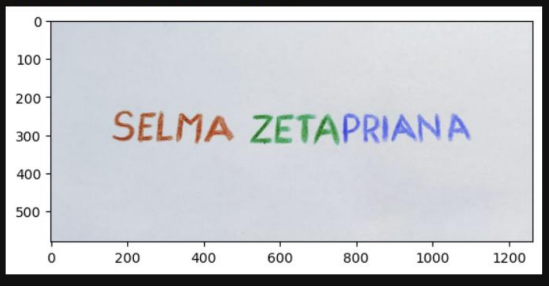
Sebelum melakukan praktikum lainnya, kita harus mengimport 3 library diatas:

1. cv2 (OpenCV): Pustaka cv2 adalah modul dari OpenCV (Open Source Computer Vision Library), yang digunakan untuk pengolahan citra dan komputer vision. OpenCV menyediakan berbagai fungsi untuk membaca, memproses, dan menganalisis gambar, termasuk operasi seperti pengubahan ukuran, pemrosesan warna, deteksi objek, dan lain-lain.
2. numpy: numpy adalah pustaka untuk perhitungan numerik di Python, yang sangat berguna untuk manipulasi array dan operasi matematika pada data citra. Dalam konteks pengolahan citra, gambar biasanya disimpan dalam format array (matriks) dan numpy digunakan untuk mengelola dan memanipulasi data gambar tersebut.
3. matplotlib.pyplot: matplotlib.pyplot adalah pustaka untuk membuat grafik dan visualisasi data, termasuk gambar. Fungsi plt digunakan untuk menampilkan gambar dalam format grafis yang mudah dimengerti, seperti histogram, citra grayscale, dan citra berwarna.

3.1 Deteksi Warna Pada Citra

MEMBACA DAN MENAMPILKAN CITRA ASLI

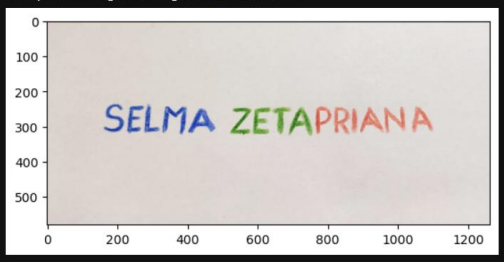
```
[4]: img = cv.imread("Citra Nama.jpg")

     plt.imshow(img)

[4]: <matplotlib.image.AxesImage at 0x16af121e6c0>
```



Pada line 4 merupakan perintah untuk membaca gambar menggunakan cv.imread dan menampilkan gambar menggunakan imshow milik plt.

KONVERSI RGB KE BGR

```
[6]: rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)

     plt.imshow(rgb)

[6]: <matplotlib.image.AxesImage at 0x16af16102c0>
```

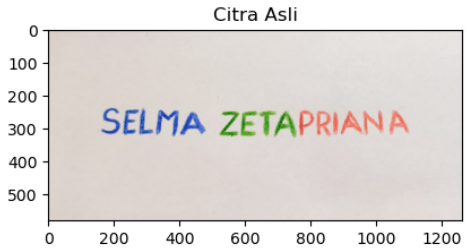


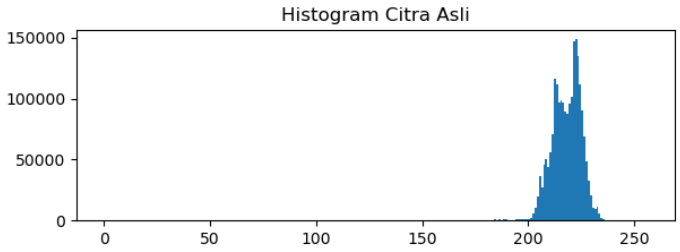
Kemudian konversi RGB ke BGR agar saat citra diolah tidak akan bertukar warna, karena OpenCV membaca citra dalam format BGR.

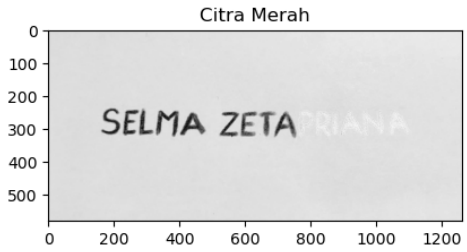
1. DETEKSI WARNA PADA CITRA SERTA HISTOGRAM

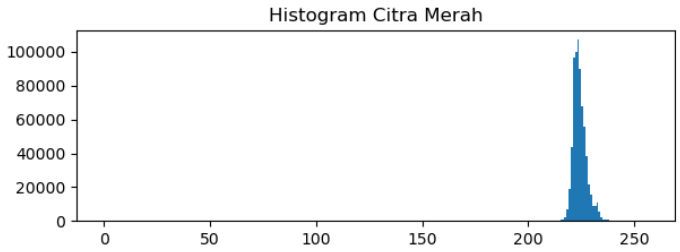
[8]:
```
fig, axs = plt.subplots(4,2, figsize=(15,12))

#citra asli
citra1 = cv.cvtColor(img, cv.COLOR_BGR2RGB)
axs[0,0].imshow(citra1)
axs[0,0].set_title('Citra Asli')
axs[0,1].hist(citra1.ravel(),256,[0,256])
axs[0,1].set_title('Histogram Citra Asli')

#channel merah
citraMerah = citra1[:, :, 0]
axs[1,0].imshow(citraMerah, cmap="gray")
axs[1,0].set_title('Citra Merah')
axs[1,1].hist(citraMerah.ravel(),256,[0,256])
axs[1,1].set_title('Histogram Citra Merah')

#channel hijau
citraHijau = citra1[:, :, 1]
axs[2,0].imshow(citraHijau, cmap="gray")
axs[2,0].set_title('Citra Hijau')
axs[2,1].hist(citraHijau.ravel(),256,[0,256])
axs[2,1].set_title('Histogram Citra Hijau')

#channel biru
citraBiru = citra1[:, :, 2]
axs[3,0].imshow(citraBiru, cmap="gray")
axs[3,0].set_title('Citra Biru')
axs[3,1].hist(citraBiru.ravel(),256,[0,256])
axs[3,1].set_title('Histogram Citra Biru')

plt.subplots_adjust(hspace=0.5)
plt.show()
```

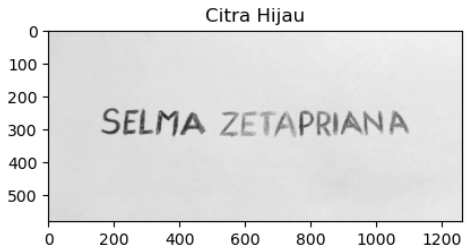
Line di atas membuat sebuah tampilan subplot menggunakan matplotlib untuk menampilkan citra asli dan histogram dari tiga saluran warna pada citra (merah, hijau, dan biru) secara terpisah. Dimulai dengan konversi citra dari format BGR ke RGB menggunakan cv.cvtColor(), citra asli ditampilkan pada subplot pertama, diikuti dengan histogram intensitas nilai piksel pada subplot kedua. Kemudian, citra dipisahkan berdasarkan saluran warna: merah, hijau, dan biru, yang masing-masing ditampilkan pada subplot dengan gambar grayscale dan histogram intensitas pikselnya pada subplot berikutnya. Terakhir, plt.subplots_adjust(hspace=0.5) digunakan untuk memberi jarak antar subplot agar tampilannya lebih rapi. Seluruh tampilan dihasilkan menggunakan plt.show().

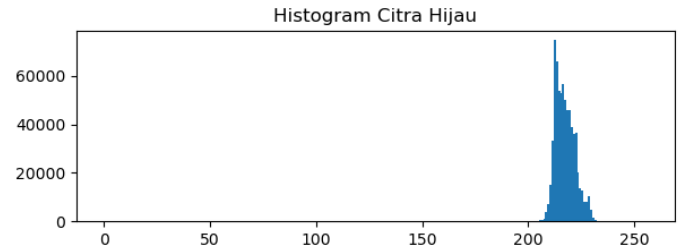


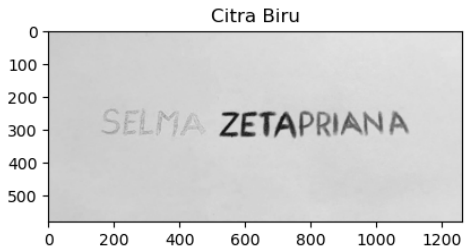


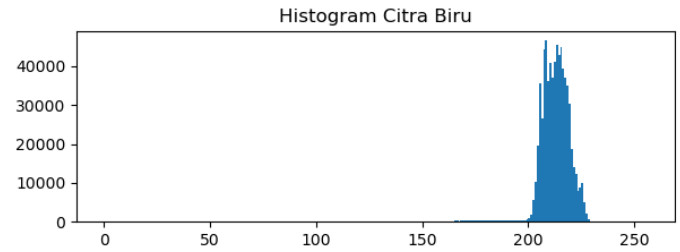












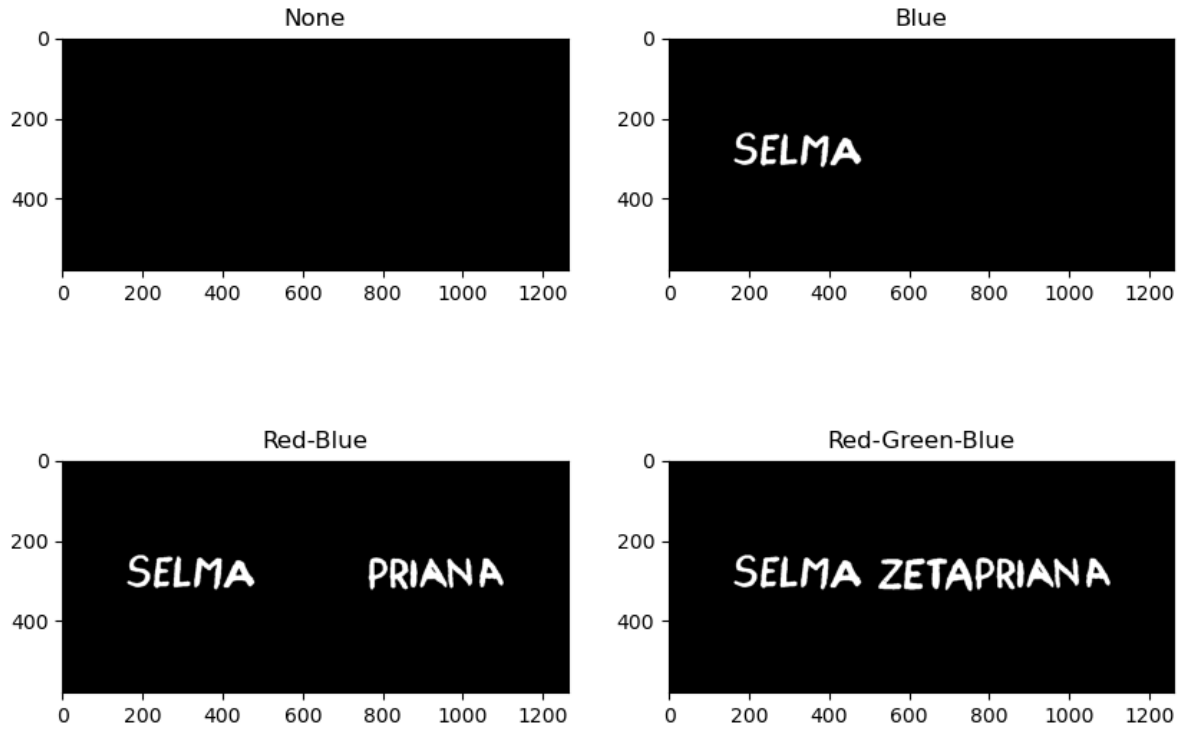
Output dalam histogram menunjukkan intensitas masing-masing channel warna dari RGB. Dengan citra asli merupakan gabungan dari ketiganya, maka mempunyai rentang yang lebih luas.

3.2 Ambang Batas (Threshold) pada Citra

2. AMBANG BATAS (THRESHOLD)

[10]:
```
image_hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)

gray = cv.cvtColor(img, cv.COLOR_RGB2GRAY)

fig, axs = plt.subplots(2, 2, figsize=(10,7))

red_lower1 = np.array([0, 50, 50])
red_upper1 = np.array([10, 255, 255])
red_lower2 = np.array([170, 50, 50])
red_upper2 = np.array([180, 255, 255])

green_lower = np.array([36, 50, 50])
green_upper = np.array([86, 255, 255])

blue_lower = np.array([100, 50, 50])
blue_upper = np.array([140, 255, 255])

mask_red1 = cv.inRange(image_hsv, red_lower1, red_upper1)
mask_red2 = cv.inRange(image_hsv, red_lower2, red_upper2)
mask_red = cv.bitwise_or(mask_red1, mask_red2) #untuk kombinasi mask
mask_green = cv.inRange(image_hsv, green_lower, green_upper)
mask_blue = cv.inRange(image_hsv, blue_lower, blue_upper)

combined_mask1 = np.bitwise_or(mask_red, mask_blue)
combined_mask2 = np.bitwise_or(combined_mask1, mask_green)

(thresh, binary1) = cv.threshold(gray, 0, 255, cv.THRESH_BINARY)
axs[0,0].imshow(binary1, cmap = 'gray')
axs[0,0].set_title('None')

plt.subplot(1, 3, 1)
axs[0,1].imshow(mask_blue, cmap='gray')
axs[0,1].set_title('Blue')
plt.axis('off')

plt.subplot(1, 3, 2)
axs[1,0].imshow(combined_mask1, cmap='gray')
axs[1,0].set_title('Red-Blue')
plt.axis('off')

plt.subplot(1, 3, 3)
axs[1,1].imshow(combined_mask2, cmap='gray')
axs[1,1].set_title('Red-Green-Blue')
plt.axis('off')

plt.show()
```

Line di atas melakukan pemrosesan citra menggunakan ruang warna HSV dan menerapkan teknik masking untuk menyorot saluran warna tertentu dalam citra. Dimulai dengan mengonversi citra dari format BGR ke HSV menggunakan cv.cvtColor(), kemudian citra juga diubah menjadi grayscale menggunakan cv.COLOR_RGB2GRAY. Selanjutnya, kode mendefinisikan rentang nilai HSV untuk warna merah, hijau, dan biru, dan membuat mask untuk masing-masing warna menggunakan cv.inRange(). Untuk warna merah, rentang dua bagian (antara 0-10 derajat dan 170-180 derajat) digabungkan dengan operasi bitwise_or() karena warna merah terletak di kedua ujung spektrum HSV. Mask merah, biru, dan hijau digunakan untuk menghasilkan kombinasi mask yang berbeda. Citra hasil thresholding grayscale kemudian ditampilkan bersama mask untuk warna biru, kombinasi merah-biru, dan kombinasi merah-hijau-biru di subplot. Setiap subplot menampilkan hasil tersebut menggunakan imshow() dengan colormap grayscale.



Output dari line diatas menunjukkan apabila mask yang digunakan warna biru maka teks yang berwarna biru akan tampil putih dan sisanya adalah hitam. Sama dengan red-blue yang menampilkan teks yang mempunyai warna antara atau keduanya, dan red-green-blue menampilkan semua teks yang mempunyai ketiga warna tersebut.

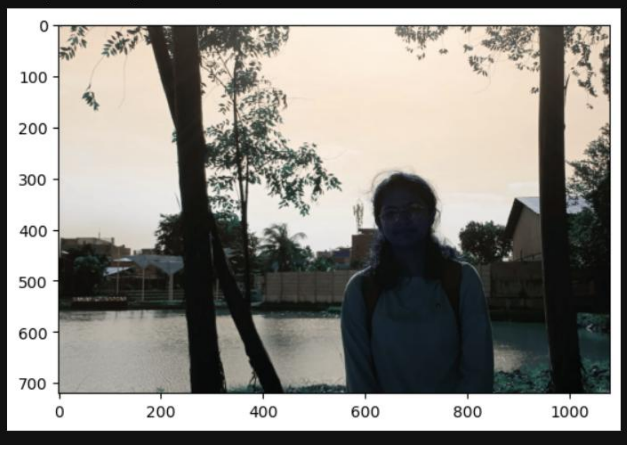
3.3 Memperbaiki Gambar Backlight Menggunakan Pencerahan dan Kontras

Foto yang digunakan adalah foto backlight saya, yang dimana kecerahan pada muka sangat minim atau gelap, maka diperlukan untuk memperbaiki foto diatas agar muka dapat terlihat.

3. MEMPERBAIKI GAMBAR BACKLIGHT

```
[12]: img2 = cv.imread("Backlight.png")

      plt.imshow(img2)
```

```
[12]: <matplotlib.image.AxesImage at 0x16af3e74170>
```

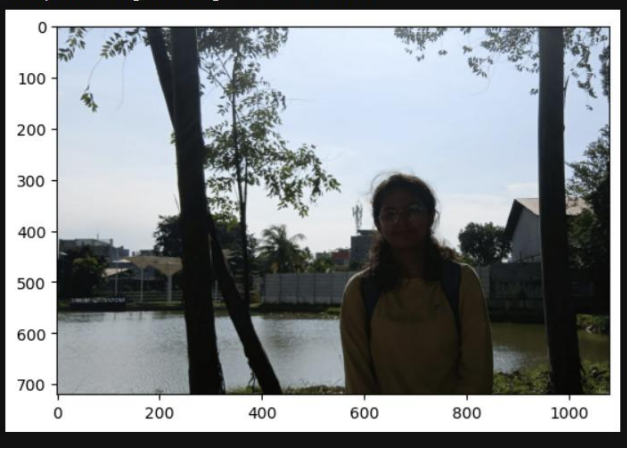


Sama seperti sebelumnya, kita perlu membaca gambar terlebih dahulu sehingga apabila ditampilkan gambarnya akan keluar seperti diatas. Namun, warna yang dihasilkan tidak sesuai citra aslinya dikarenakan warna belum di konversi.

RGB TO BGR

```
[14]: img2rgb = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

      plt.imshow(img2rgb)
```

```
[14]: <matplotlib.image.AxesImage at 0x16af4c457f0>
```

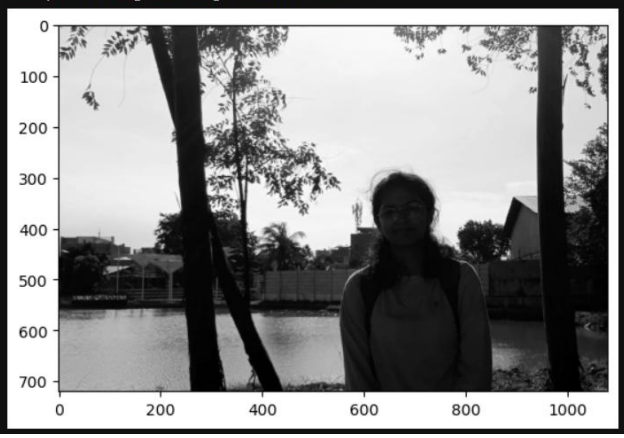


Line diatas ini adalah cara bagaimana mengonversi warna RGB ke BGR, sehingga apabila ditampilkan kembali gambar yang telah dikonversi warnanya, akan menunjukkan warna yang sesuai citra aslinya.

BGR TO GRAYSCALE

```
[16]: img2gray = cv.cvtColor(img2rgb, cv.COLOR_RGB2GRAY)

      plt.imshow(img2gray, cmap='gray')
```

```
[16]: <matplotlib.image.AxesImage at 0x16af4ca2660>
```



Kemudian line selanjutnya merupakan konversi BGR ke Grayscale (abu). Dengan cmap=’gray’ digunakan agar warna yang ditampilkan tidak mempunyai warna lain selain abu (hitam hingga putih).

```
[17]: (baris, kolom) = img2.shape[:2]
```

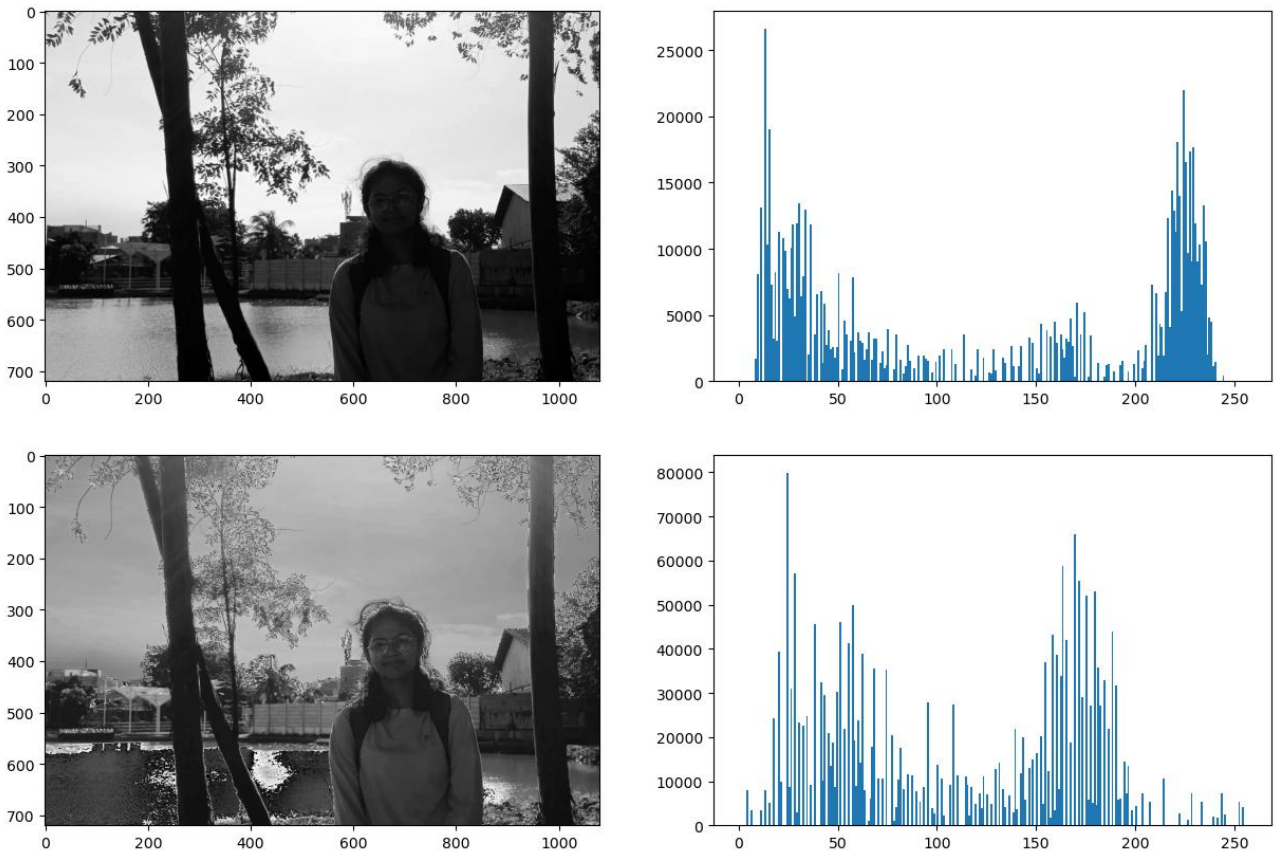
(baris, kolom) = img2.shape[:2] digunakan untuk mendapatkan dimensi dari citra img2. Dalam hal ini, img2.shape akan mengembalikan sebuah tuple yang berisi ukuran citra dalam format (tinggi, lebar, jumlah saluran warna). Dengan menggunakan [:2], kita hanya mengambil dua nilai pertama dari tuple tersebut, yang mewakili dimensi citra yaitu jumlah baris (tinggi) dan kolom (lebar), tanpa memperhitungkan jumlah saluran warna (misalnya RGB atau BGR).

GAMBAR GRAY YANG DIPERKONTRAS

```
[19]: alpha = 1.9
      citra_kontras = np.zeros((baris,kolom,3))

      for x in range(baris):
          for y in range(kolom):
              gmx = img2gray[x,y] * alpha
              citra_kontras[x,y] = gmx

      citra_kontras = citra_kontras.astype(np.uint8)

      fig, axs = plt.subplots(2,2, figsize=(15,10))
      axs[0,0].imshow(img2gray, cmap='gray')
      axs[0,1].hist(img2gray.ravel(),256,[0,256])
      axs[1,0].imshow(citra_kontras, cmap='gray')
      axs[1,1].hist(citra_kontras.ravel(),256,[0,256])
      plt.show()
```

Pada kode line di atas, dilakukan peningkatan kontras citra grayscale img2gray dengan mengalikan nilai intensitas piksel setiap titik dengan faktor kontras alpha, yang diatur menjadi 1.9. Citra baru yang dihasilkan disimpan dalam citra_kontras. Sebelumnya, sebuah array kosong citra_kontras dengan ukuran yang sesuai dengan citra grayscale (baris, kolom, 3 saluran warna) dibuat. Dalam loop ganda (for), setiap piksel pada citra grayscale diambil, kemudian intensitasnya dikalikan dengan faktor alpha, dan hasilnya disalin ke citra kontras.

Setelah itu, kedua citra (citra asli grayscale dan citra kontras) beserta histogramnya ditampilkan dalam bentuk subplot. Pada subplot pertama di baris pertama, ditampilkan citra grayscale img2gray, sedangkan pada subplot kedua, histogram dari citra tersebut diperlihatkan. Di baris kedua, subplot pertama menampilkan citra dengan kontras yang ditingkatkan (citra_kontras), dan subplot kedua menampilkan histogram dari citra kontras tersebut, lalu fungsi plt.show() digunakan untuk menampilkan seluruh plot tersebut.



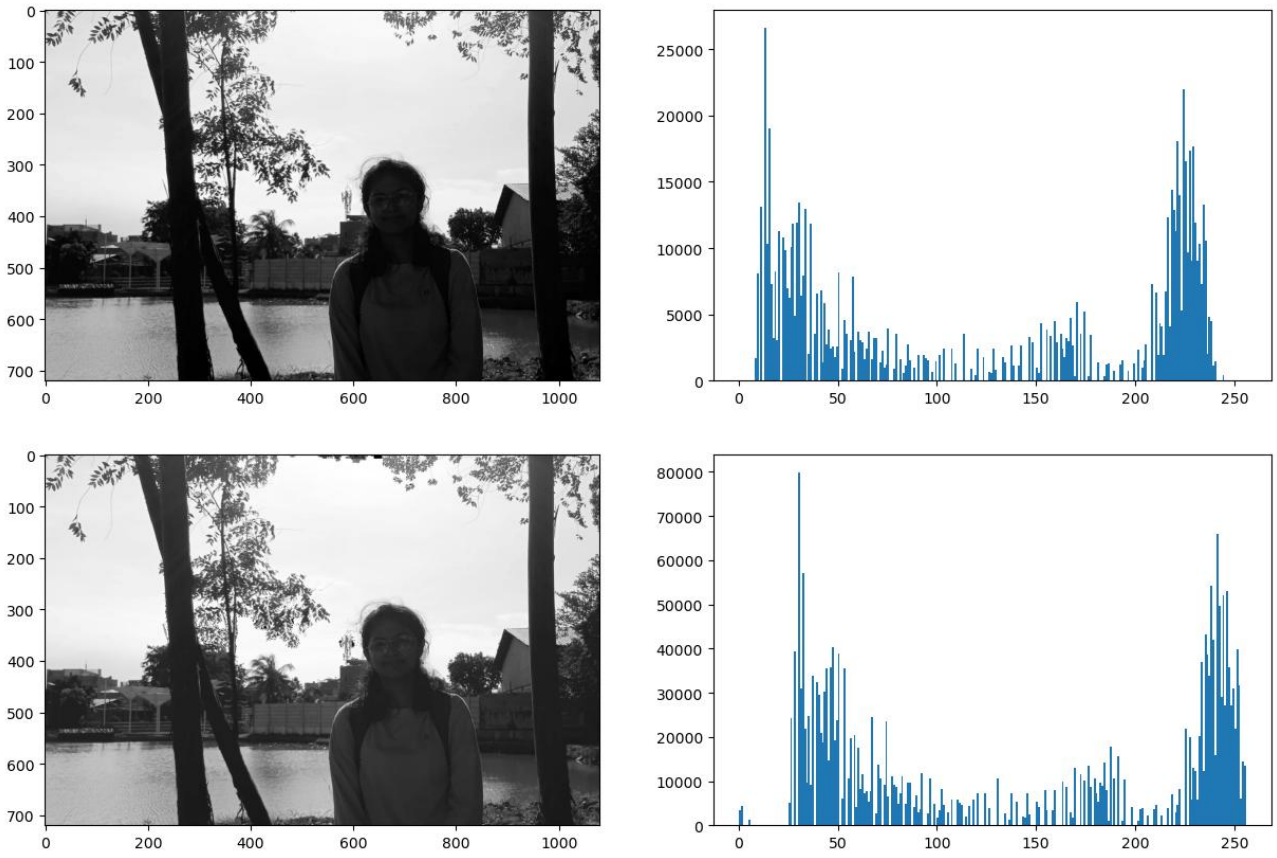
Output diatas menunjukkan gambar yang telah diperkontras dan histogram yang menunjukkan intensitas warna setelah diperkontras.

GAMBAR GRAY YANG DIPERCERAH

```
[21]: beta = 17
citra_cerah = np.zeros((baris,kolom,3))

for x in range(baris):
    for y in range(kolom):
        gyx = img2gray[x,y] + beta
        citra_cerah[x,y] = gyx

citra_cerah = citra_cerah.astype(np.uint8)

fig, axs = plt.subplots(2,2, figsize =(15,10))
axs[0,0].imshow(img2gray, cmap='gray')
axs[0,1].hist(img2gray.ravel(),256,[0,256])
axs[1,0].imshow(citra_cerah, cmap='gray')
axs[1,1].hist(citra_cerah.ravel(),256,[0,256])
plt.show()
```

Sama seperti cara mengkontraskan citra, hanya saja perbedaannya di beta, yang merupakan nilai kecerahan. Citra grayscale img2gray diberi peningkatan kecerahan dengan menambahkan nilai beta (yang diatur menjadi 17) ke setiap piksel dalam citra. Kemudian nilai beta ditambahkan ke intensitas piksel tersebut. Hasilnya disalin ke array citra_cerah, yang

berfungsi untuk menyimpan citra dengan kecerahan yang telah ditingkatkan. Setelah itu, array citra_cerah diubah menjadi tipe data np.uint8 agar dapat digunakan dalam visualisasi citra.



Output diatas adalah gambar sebelum dan sesudah dipercerah. Dengan beta = 17, maka hasil pencerahan yang terjadi juga tidak besar. Dalam praktikum ini, saya mendapatkan color burn atau perubahan warna signifikan, dari yang putih ke hitam. Maka dari itu, saya menggunakan beta kecil agar hasil tidak rusak.

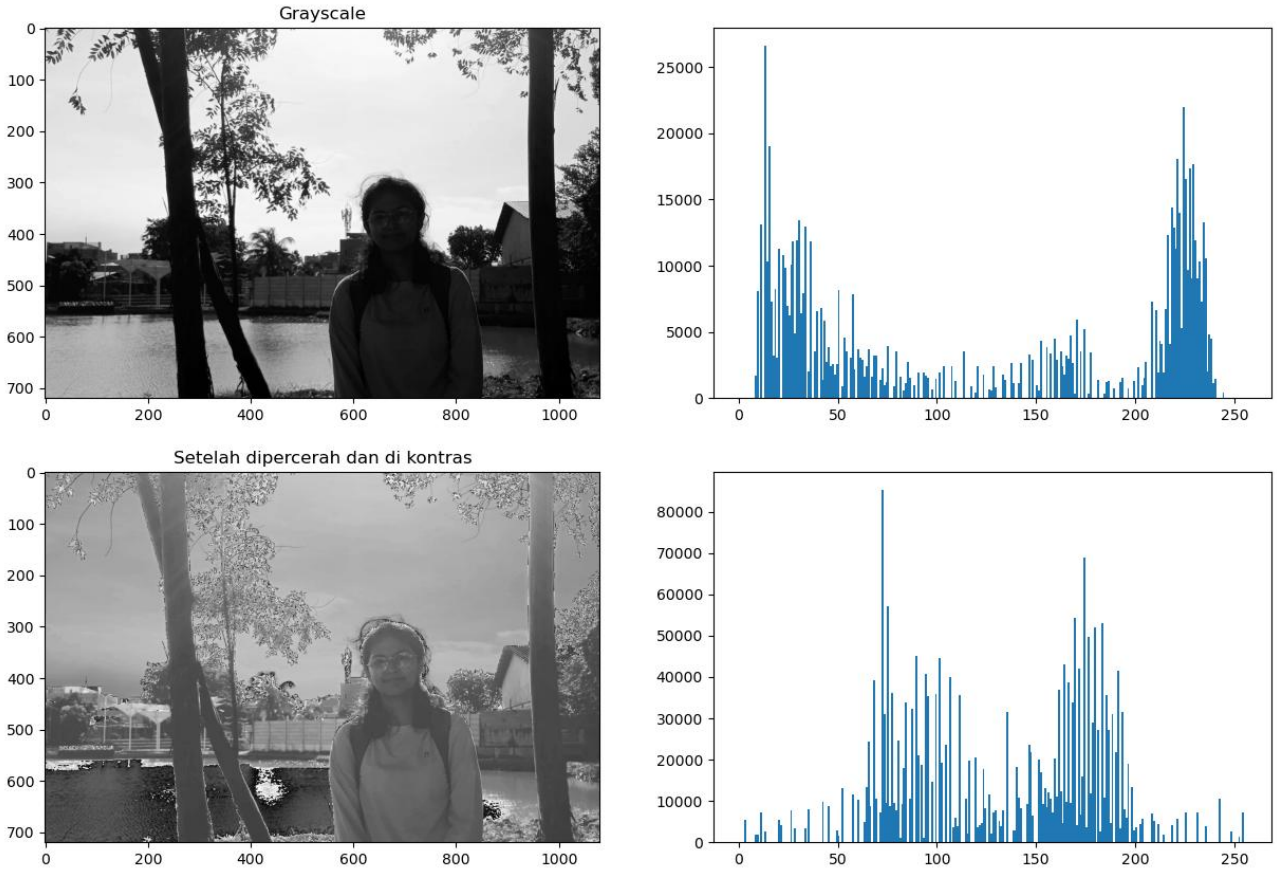
PERBANDINGAN CITRA SEBELUM DAN SESUDAH DIPERBAIKI

[23]:
```
beta = 50
alpha = 1.7
img_hasil = np.zeros((baris,kolom,3))

for x in range(baris):
    for y in range(kolom):
        gcx = alpha * img2gray[x,y] + beta
        img_hasil[x,y] = gcx

img_hasil = img_hasil.astype(np.uint8)

fig, axs = plt.subplots(2,2, figsize=(15,10))
axs[0,0].imshow(img2gray,cmap='gray')
axs[0,0].set_title('Grayscale')
axs[0,1].hist(img2gray.ravel(),256,[0,256])
axs[1,0].imshow(img_hasil,cmap='gray')
axs[1,0].set_title('Setelah dipercerah dan di kontras')
axs[1,1].hist(img_hasil.ravel(),256,[0,256])
plt.show()
```

Kemudian line berikutnya adalah gabungan dari cara mencerahkan dan menambah kontras kedalam citra. Proses pertama melibatkan pengalian nilai intensitas piksel dengan faktor alpha (yang diatur menjadi 1.7) untuk meningkatkan kontras. Proses kedua menambahkan nilai beta (yang diatur menjadi 50) ke setiap piksel untuk meningkatkan kecerahan citra. Hasil kombinasi kedua proses ini disalin ke dalam array img_hasil.

Setelah proses tersebut, array img_hasil diubah menjadi tipe data np.uint8 untuk memastikan nilai intensitas berada dalam rentang yang tepat untuk citra (0-255).

Kemudian, dua citra dan histogramnya ditampilkan menggunakan matplotlib. Subplot pertama menampilkan citra asli grayscale (img2gray), sementara subplot kedua menunjukkan histogram dari citra tersebut. Subplot ketiga menampilkan citra hasil setelah peningkatan kecerahan dan kontras (img_hasil), dengan subplot terakhir yang menampilkan histogram dari citra hasil tersebut.



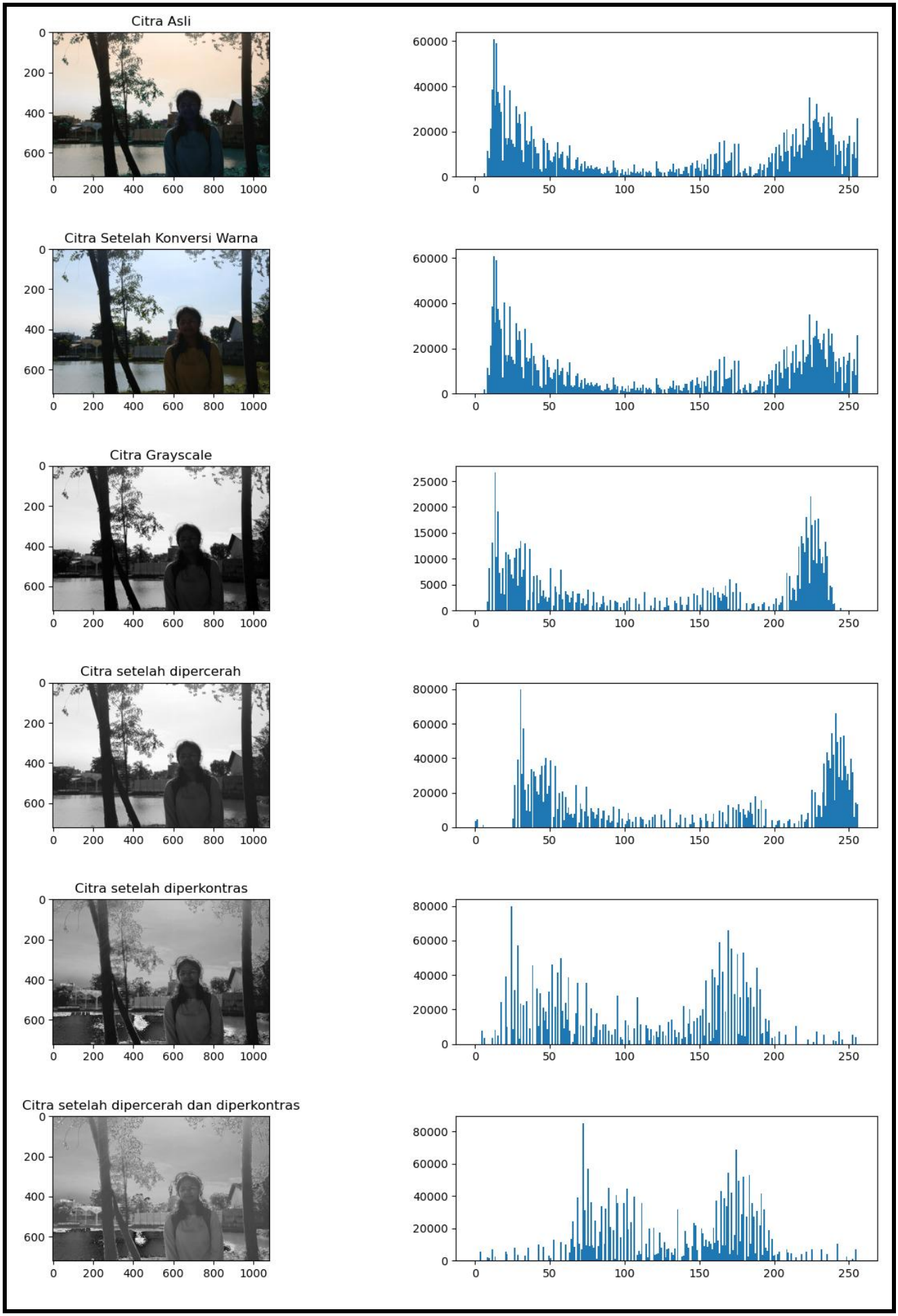
Output diatas adalah kombinasi pencerahan dan penambahan kontras cahaya sehingga menghasilkan citra yang terang. Dari yang sebelumnya muka tidak dapat terlihat dengan jelas, setelah diperbaiki maka bentuk muka dapat terlihat dengan jelas.

ALL IN ONE DENGAN HISTOGRAM

[25]:
```
fig, axs = plt.subplots(6,2, figsize=(15,20))

axs[0,0].imshow(img2)
axs[0,0].set_title('Citra Asli')
axs[0,1].hist(img2.ravel(),256,[0,256])

axs[1,0].imshow(img2rgb)
axs[1,0].set_title('Citra Setelah Konversi Warna')
axs[1,1].hist(img2rgb.ravel(),256,[0,256])

axs[2,0].imshow(img2gray, cmap='gray')
axs[2,0].set_title('Citra Grayscale')
axs[2,1].hist(img2gray.ravel(),256,[0,256])

axs[3,0].imshow(citra_cerah, cmap='gray')
axs[3,0].set_title('Citra setelah dipercerah')
axs[3,1].hist(citra_cerah.ravel(),256,[0,256])

axs[4,0].imshow(citra_kontras, cmap='gray')
axs[4,0].set_title('Citra setelah diperkontras')
axs[4,1].hist(citra_kontras.ravel(),256,[0,256])

axs[5,0].imshow(img_hasil, cmap='gray')
axs[5,0].set_title('Citra setelah dipercerah dan diperkontras')
axs[5,1].hist(img_hasil.ravel(),256,[0,256])

plt.subplots_adjust(hspace=0.5)
plt.show()
```

Dalam laporan ini, saya menambahkan output dalam 1 gambar utuh, dengan code diatas adalah cara menampilkan gambar menggunakan subplot serta histogram.

UTS


Citra Asli
Citra Setelah Konversi Warna
Citra Grayscale
Citra setelah dipercerah
Citra setelah diperkontras
Citra setelah dipercerah dan diperkontras

BAB IV

PENUTUP

Dalam pengolahan citra digital, deteksi warna, penentuan ambang batas (threshold), dan perbaikan gambar backlight adalah teknik-teknik dasar yang sangat penting dalam memahami dan mengolah gambar. Deteksi warna memungkinkan kita untuk mengenali dan mengekstraksi objek berdasarkan karakteristik warna, yang sangat berguna dalam berbagai aplikasi seperti pelacakan objek, segmentasi citra, dan pengenalan benda. Teknik ini melibatkan konversi citra ke model warna tertentu, seperti HSV, untuk mempermudah pemisahan warna dan menghindari gangguan dari pencahayaan.

Ambang batas pada citra digunakan untuk mengelompokkan warna berdasarkan nilai minimum dan maksimum dari saluran warna tertentu, seperti Hue, Saturation, dan Value (HSV). Penggunaan threshold ini memungkinkan kita untuk menyorot warna tertentu dalam citra dengan lebih akurat, menghilangkan noise, dan mempermudah analisis.

Memperbaiki gambar dengan backlight memerlukan peningkatan kecerahan dan kontras agar objek utama dalam citra dapat lebih terlihat jelas. Dengan menyesuaikan kecerahan dan kontras menggunakan faktor alpha dan beta, kita dapat meningkatkan kualitas gambar yang sebelumnya terhalang oleh pencahayaan belakang (backlight).

Pada praktikum yang telah dilakukan menggunakan Jupyter Notebook, beberapa teknik pengolahan citra diterapkan dengan menggunakan pustaka seperti OpenCV, numpy, dan matplotlib. Proses dimulai dengan deteksi warna pada citra menggunakan teknik masking berdasarkan ruang warna HSV. Setelah konversi citra ke format HSV, mask untuk warna tertentu (seperti merah, hijau, dan biru) dibuat dan digabungkan untuk mendeteksi objek berwarna dalam citra.

Selanjutnya, pada tahap ambang batas, citra diolah untuk memisahkan warna berdasarkan rentang nilai tertentu pada saluran warna HSV. Teknik ini memungkinkan identifikasi objek berdasarkan warna yang diinginkan, dengan hasil yang dapat dianalisis menggunakan histogram untuk memvisualisasikan distribusi intensitas warna dalam citra.

Pada tahap perbaikan gambar backlight, citra yang awalnya gelap karena pencahayaan belakang diperbaiki dengan meningkatkan kecerahan (dengan menambahkan nilai beta) dan kontras (dengan mengalikan intensitas piksel dengan alpha). Gabungan kedua teknik ini menghasilkan citra yang lebih terang dan jelas, terutama pada area yang sebelumnya terlalu gelap. Hasil citra yang telah diperbaiki menunjukkan peningkatan kualitas visual, di mana objek utama dalam citra, seperti wajah, menjadi lebih jelas terlihat.

menggunakan subplot dan histogram, hasil dari setiap proses pengolahan citra ditampilkan secara jelas, memperlihatkan perbedaan antara citra asli, citra yang diproses, dan distribusi intensitas warna atau kecerahan pada setiap citra.

DAFTAR PUSTAKA

[1] Gunawan, Irwan Prasetya et al. (2023) Pemrosesan citra. PT MAFY MEDIA LITERASI INDONESIA.
[2] Ratna, S. (2020). Pengolahan citra digital dan histogram dengan Python dan text editor PyCharm. Technologia, 11(3).
[3] Zonyfar, C. (2020). Pengolahan citra digital: Sebuah pengantar. Desanta Publisher.
[4] Arifin, T. N., Pratiwi, G. F., & Ilman, S. (2024). Pendeteksian warna dengan image processing menggunakan model warna HSB. JTEIN: Jurnal Teknik Elektro Indonesia, 5(1)
[5] Fitriyah, H., & Wihandika, R. C. (2021). Dasar-dasar pengolahan citra digital. Universitas Brawijaya Press.
[6] Arnita, Marpaung, et al. (2022) Computer Vision Dan Pengolahan Citra Digital. Pustaka Aksara, Surabaya.
[7] Dijaya, R., & Setiawan, H. (2023). Buku Ajar Pengolahan Citra Digital. Umsida Press