CASE METHOD 1 WEEK7

Ghoffar Abdul Ja'far(16)

Muhammad Irsyad Dimas Abdillah (20) 2341720088

Petunjuk Pengerjaan

Gunakan 3 citra wajah (misalnya face1.jpg, face2.jpg, face3.jpg) dengan variasi kondisi cahaya.

1. Tampilkan histogram tiap citra dan analisis distribusi intensitasnya.
2. Terapkan transformasi brightness dan contrast (linear/log brightness).
    - Tentukan nilai b (brightness) dan a (contrast) yang sesuai agar wajah tampak natural.
3. Lakukan histogram equalization untuk memperbaiki sebaran kontras.
    - Bandingkan hasil visual dan histogram sebelum–sesudah.
4. Terapkan filter spasial:
    - Low-pass filter untuk menghaluskan noise kulit wajah.
    - High-pass atau Laplacian filter untuk menajamkan tepi mata dan bibir.
5. Implementasikan Floyd–Steinberg Dithering untuk menurunkan kedalaman warna wajah (bit-depth 4–6 bit), lalu analisis bagaimana efeknya terhadap detail dan ekspresi wajah.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
from google.colab.patches import cv2_imshow
```

```
from google.colab import drive
drive.mount('/content/drive')
```

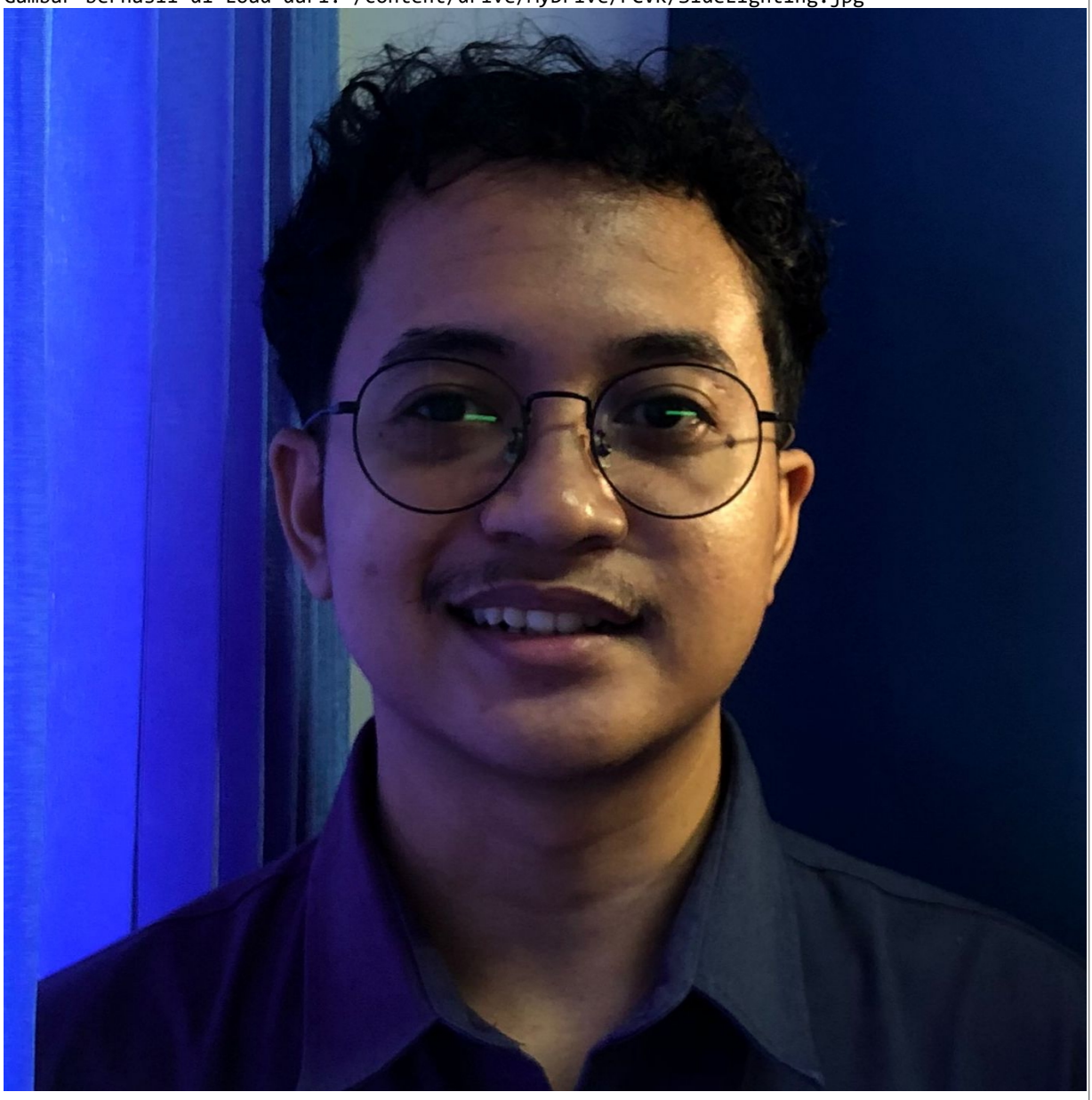
```
Mounted at /content/drive
```

```
# Load Images from drive with a loop
img_paths = [
    '/content/drive/MyDrive/PCVK/UnderExposed.jpg',
    '/content/drive/MyDrive/PCVK/OverExposed.jpg',
    '/content/drive/MyDrive/PCVK/SideLighting.jpg'
]

loaded_images = {}
for img_path in img_paths:
    img = cv.imread(img_path)
    if img is not None:
        # Ambil nama foto dari path
        short_name = img_path.split('/')[-1].split('.')[0]
        loaded_images[short_name] = img
        print(f"Gambar berhasil di Load dari: {img_path}")
        # Tampilkan Gambar
        cv2_imshow(img)
    else:
        print(f"Error: Gambar berhasil di load dari: {img_path}. Cek path file.")
```

```
Gambar berhasil di Load dari: /content/drive/MyDrive/PCVK/UnderExposed.jpg
```

```
Gambar berhasil di Load dari: /content/drive/MyDrive/PCVK/OverExposed.jpg
```


```
Gambar berhasil di Load dari: /content/drive/MyDrive/PCVK/SideLighting.jpg
```

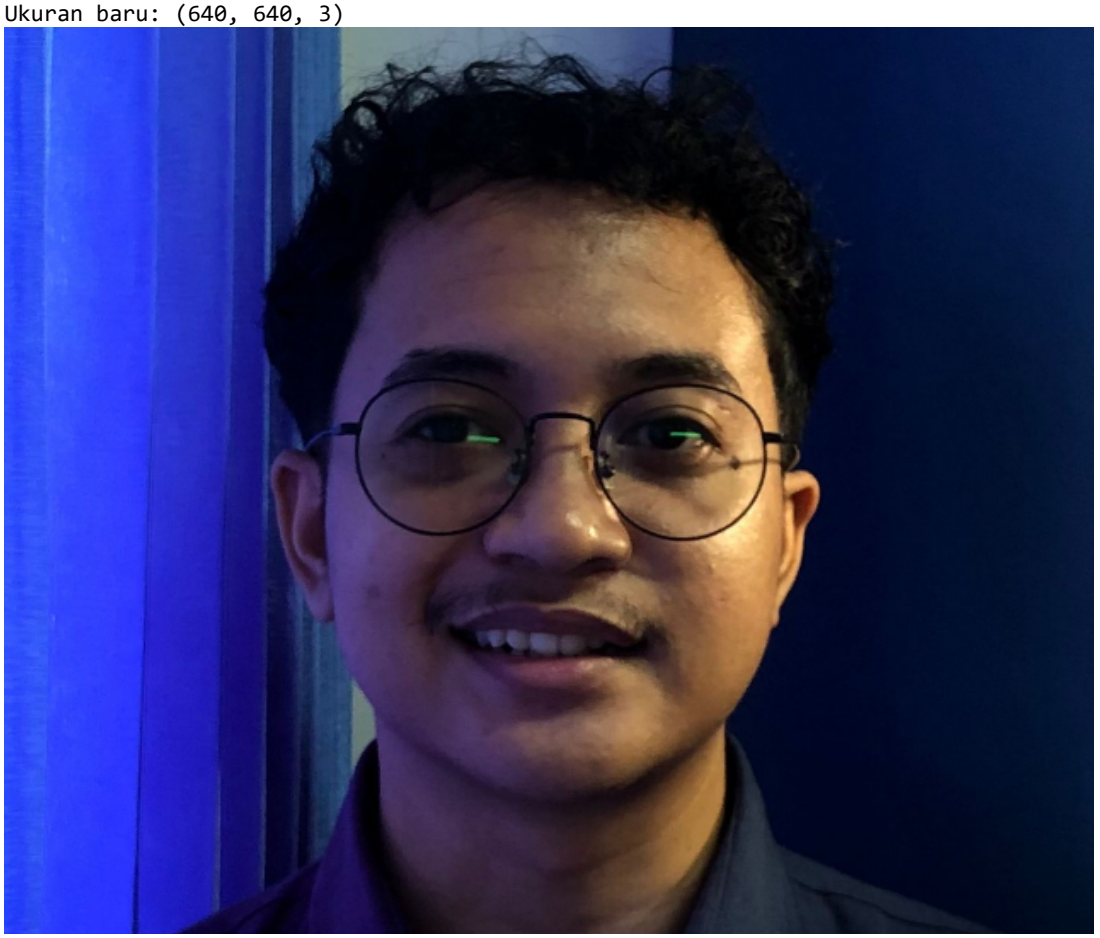


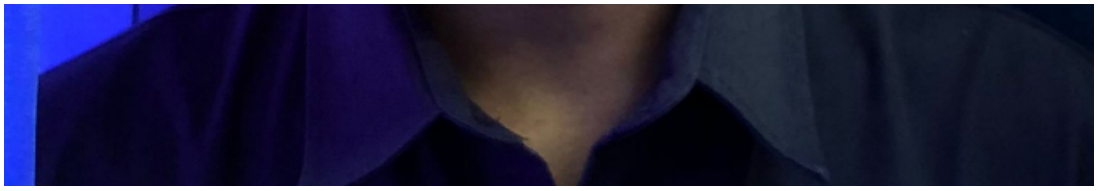
Preprocessing gambar

```
# 1. ubah ukuran gambar menjadi 640*640
# a. Under Exposed
underResized = cv.resize(loaded_images['UnderExposed'], (640, 640))
cv2_imshow(underResized)
print("Ukuran baru:", underResized.shape)

# b. Over Exposed
overResized = cv.resize(loaded_images['OverExposed'], (640, 640))
cv2_imshow(overResized)
print("Ukuran baru:", overResized.shape)

# c. Side Lighting
sideResized = cv.resize(loaded_images['SideLighting'], (640, 640))
cv2_imshow(sideResized)
print("Ukuran baru:", sideResized.shape)
```

```
Ukuran baru: (640, 640, 3)
```


```
Ukuran baru: (640, 640, 3)
```

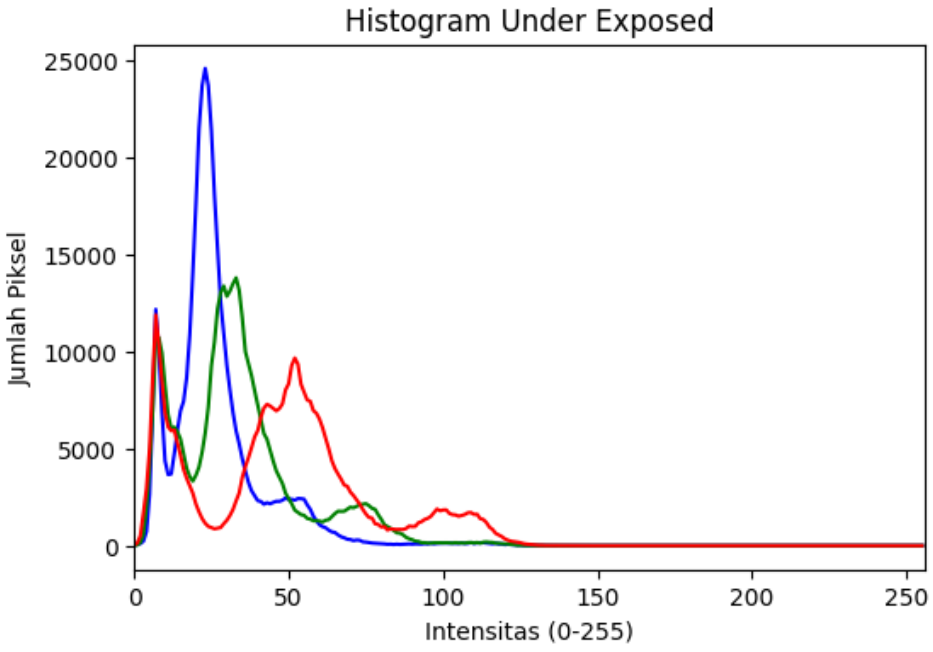


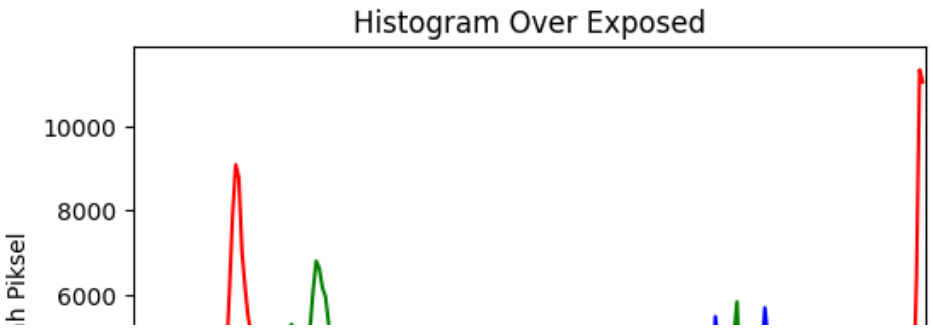


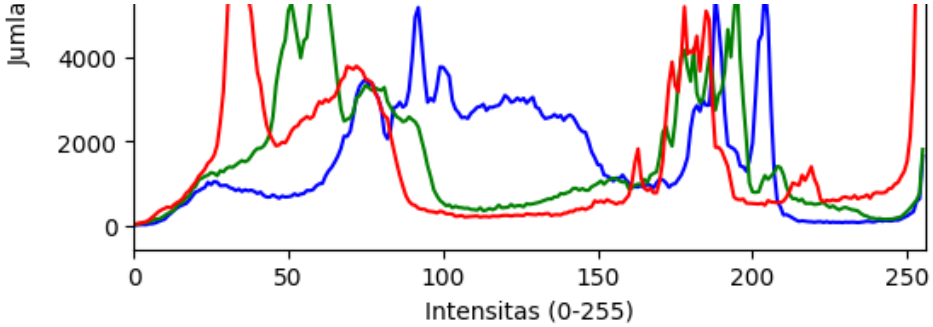
```
Ukuran baru: (640, 640, 3)
```

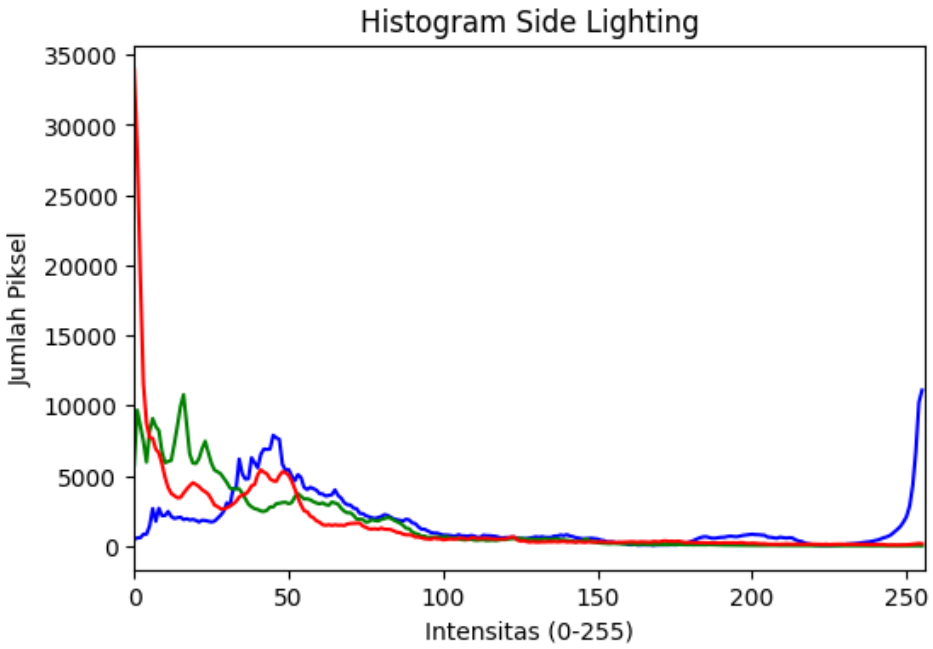
1. Tampilkan Histogram Tiap Citra dan Analisis distribusi intensitasnya

```
# Fungsi bantu untuk menampilkan histogram
def show_histogram(image, title):
    color = ('b','g','r')
    plt.figure(figsize=(6,4))
    for i,col in enumerate(color):
        hist = cv.calcHist([image],[i],None,[256],[0,256])
        plt.plot(hist,color=col)
        plt.xlim([0,256])
    plt.title(f"Histogram {title}")
    plt.xlabel("Intensitas (0-255)")
    plt.ylabel("Jumlah Piksel")
    plt.show()

# Tampilkan histogram untuk tiap citra
show_histogram(underResized, "Under Exposed")
show_histogram(overResized, "Over Exposed")
show_histogram(sideResized, "Side Lighting")
```









ANALISIS HISTOGRAM

Dalam histogram pertama, yang disebut "Under Exposed", sebagian besar warna menumpuk di area gelap dengan puncak tajam dan intensitas rendah, dengan bagian tengah yang hampir kosong dari terang ke gelap. Ini membuat foto terlihat suram, tidak ada detail, dan tidak cukup cahaya.

Distribusi piksel cenderung berada di area terang pada histogram kedua yang disebut "Over Exposed". Serangan besar di ujung kanan menunjukkan dominasi fokus. Akibatnya, banyak detail terbakar oleh cahaya, yang menghasilkan gambar yang terlalu pucat.

Di sisi lain, histogram ketiga menunjukkan "Penerangan sisi", yang menunjukkan distribusi ekstrem dengan puncak di sisi gelap dan lonjakan di sisi terang. Bagian tengahnya relatif kosong, yang menghasilkan kontras tinggi. Pencahayaan samping terlihat di gambar ini. Area gelap terlihat jelas di satu sisi, dan area yang lebih terang di sisi lain, menghasilkan efek yang sangat besar.

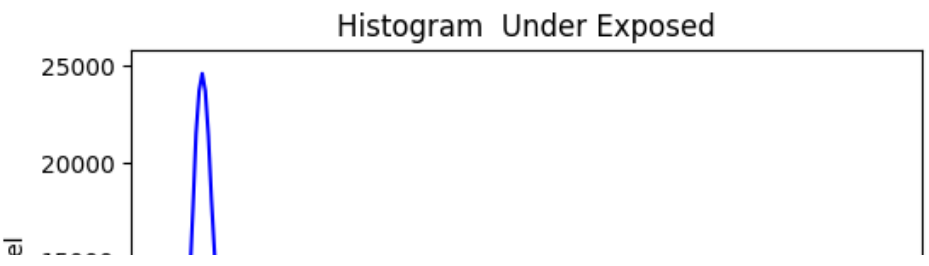
2. Terapkan transformasi brightness dan contrast (linear/log brightness).

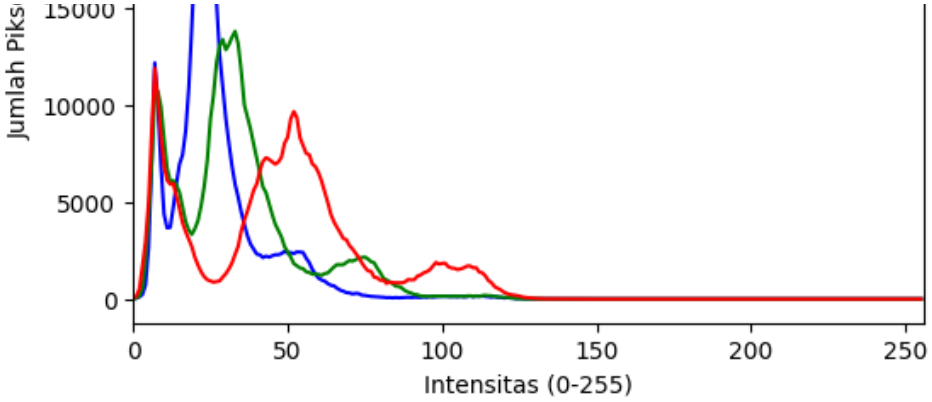
- Tentukan nilai b (brightness) dan a (contrast) yang sesuai agar wajah tampak natural.

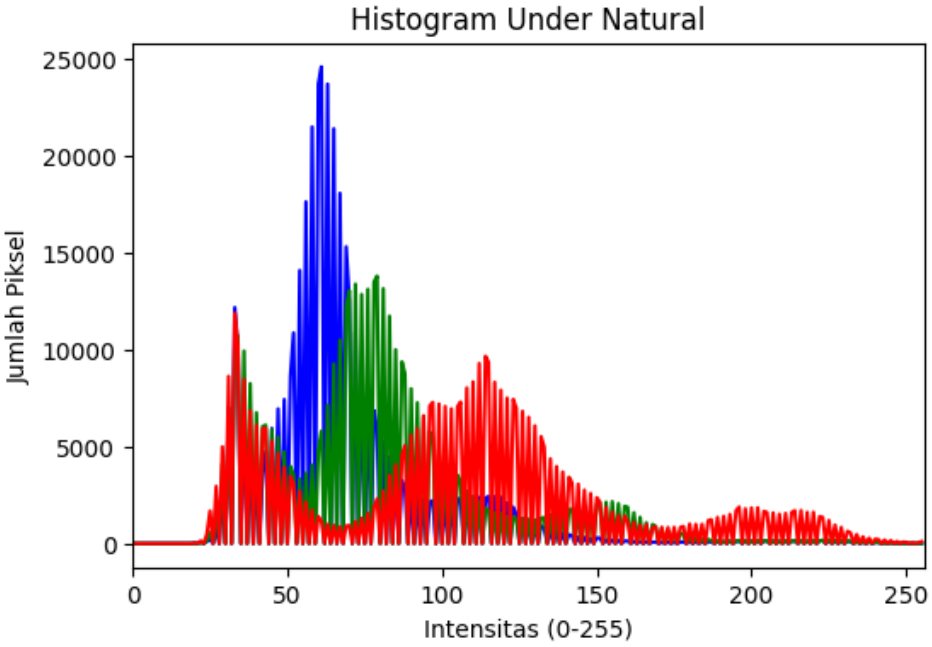
```
# Fungsi untuk menerapkan brightness dan contrast linier
def adjust_brightness_contrast(image, alpha, beta):
    # new_image = alpha * image + beta
    adjusted_image = cv.convertScaleAbs(image, alpha=alpha, beta=beta)
    return adjusted_image
```

```
    return adjusted_image

# Fungsi untuk menampilkan gambar side by side
def show_before_after(original, transformed, title):
    plt.figure(figsize=(10,4))

    # Sebelum
    plt.subplot(1,2,1)
    plt.imshow(cv.cvtColor(original, cv.COLOR_BGR2RGB))
    plt.title(f"{title} - Before")
    plt.axis("off")

    # Sesudah
    plt.subplot(1,2,2)
    plt.imshow(cv.cvtColor(transformed, cv.COLOR_BGR2RGB))
    plt.title(f"{title} After")
    plt.axis("off")

    plt.show()
```

```
# a. Under Exposed
underNatural = adjust_brightness_contrast(underResized, 1.8, 20)
show_before_after(underResized, underNatural, "Under Exposed")
show_histogram(underResized, " Under Exposed")
show_histogram(underNatural, "Under Natural")

# b. Over Exposed
overNatural = adjust_brightness_contrast(overResized, 1.15, -20)
show_before_after(overResized, overNatural, "Over Exposed")
show_histogram(overResized, "Over Exposed")
show_histogram(overNatural, "Over Natural")

# c. Side Lighting
sideNatural = adjust_brightness_contrast(sideResized, 1.15, 20)
show_before_after(sideResized, sideNatural, "Side Lighting")
show_histogram(sideResized, "Side Lighting")
show_histogram(sideNatural, "Side Natural")
```

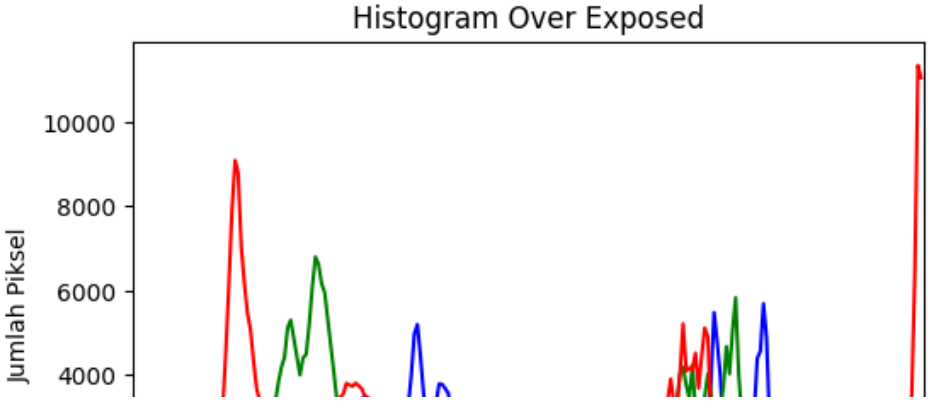


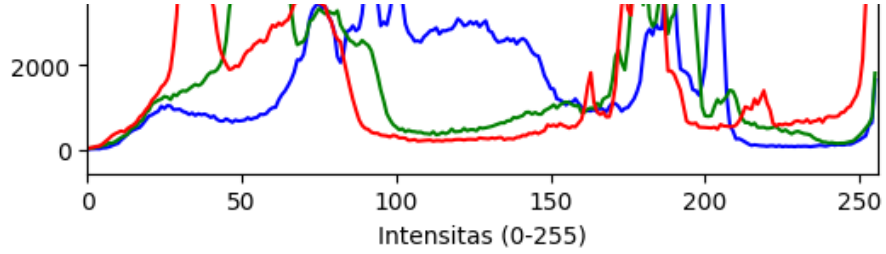


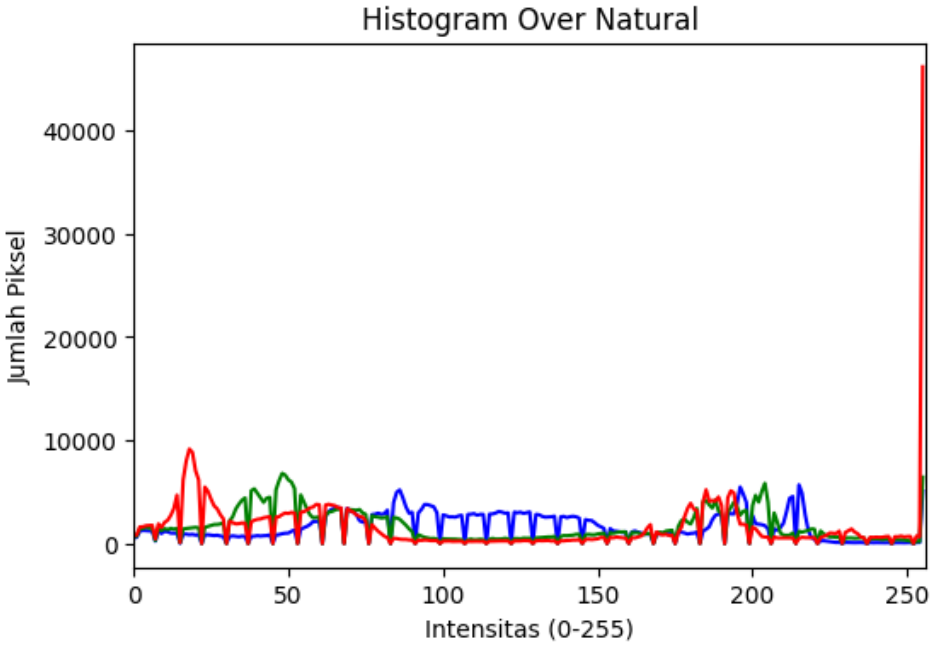


Over Exposed - Before

Over Exposed After

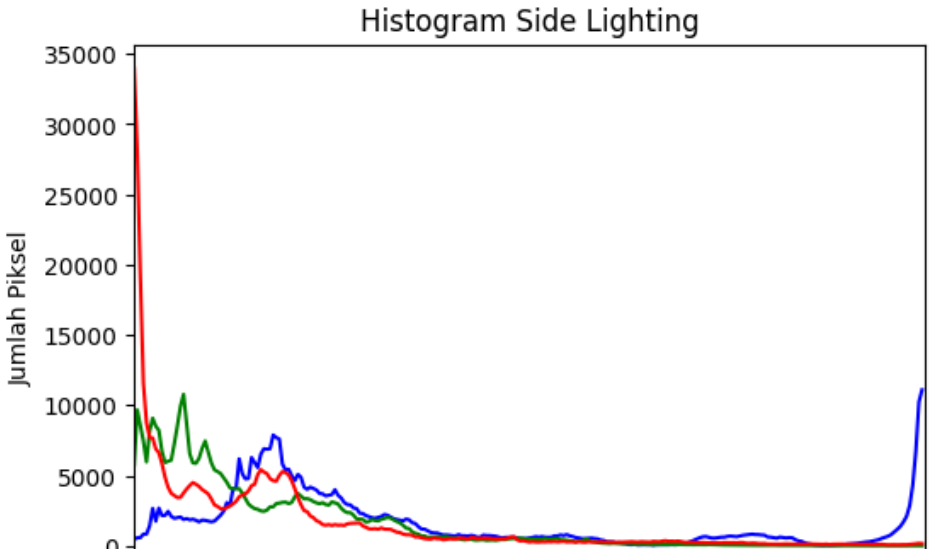


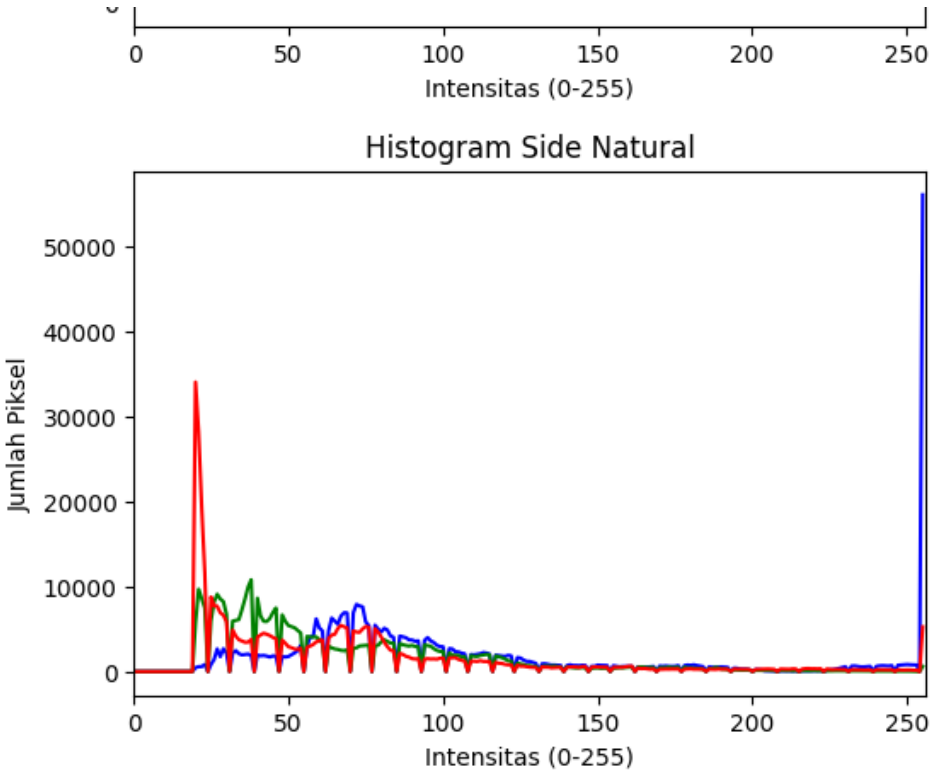




Side Lighting - Before

Side Lighting After



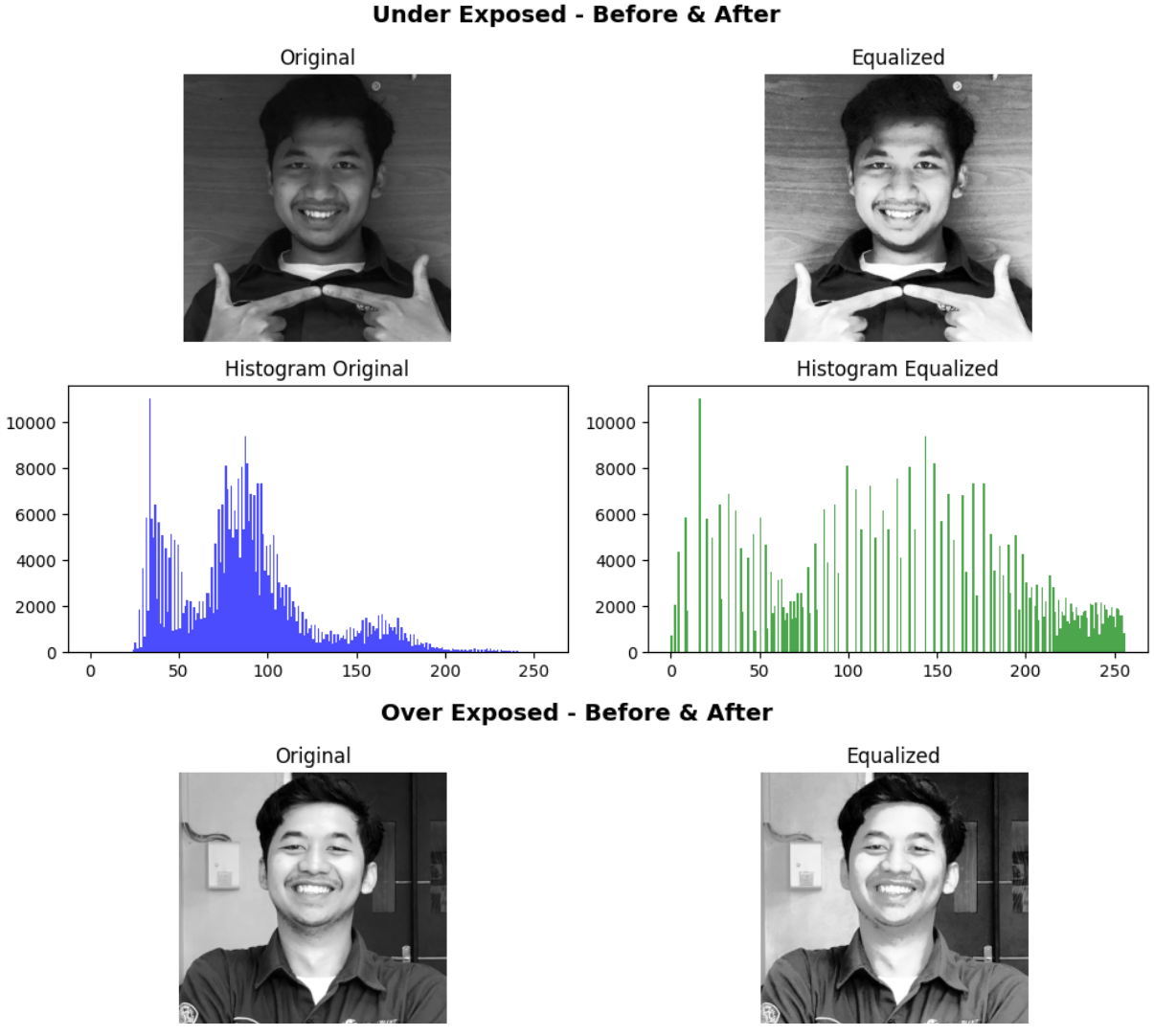


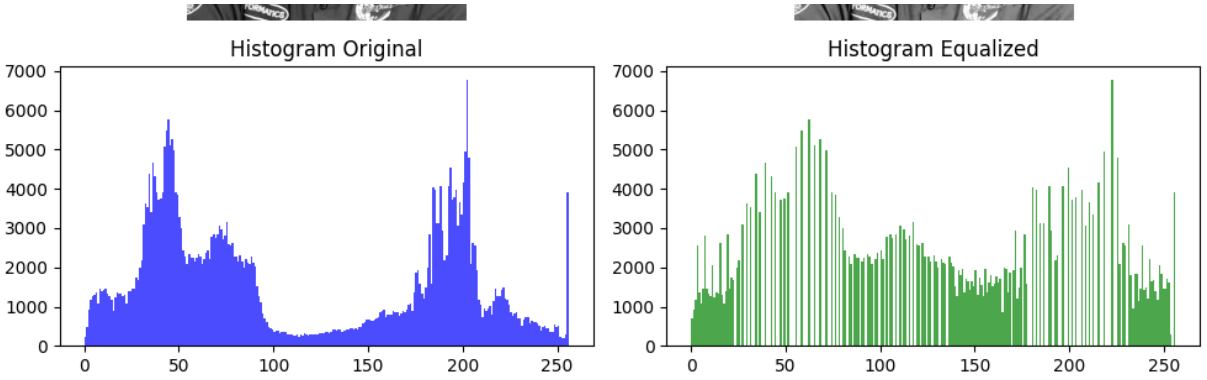
3. Lakukan histogram equalization untuk memperbaiki sebaran kontras.

- Bandingkan hasil visual dan histogram sebelum–sesudah.

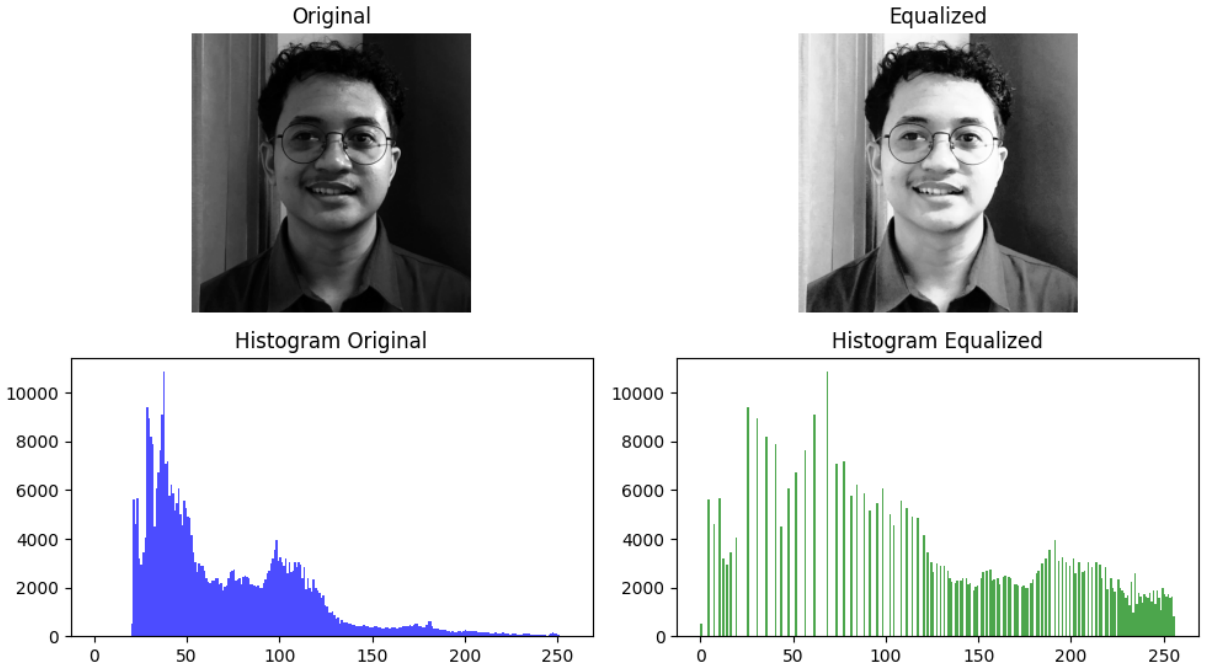
```
def manual_histogram_equalization(image):
    """Manual Histogram Equalization: s_k = (L-1) * Σ(p_i)"""
    if len(image.shape) == 3:
        gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    else:
        gray = image.copy()

    M, N = gray.shape
    total = M * N
    L = 256

    # Hitung histogram dan probabilitas
    hist, _ = np.histogram(gray.flatten(), bins=L, range=[0, L])
    prob = hist / total

    # Hitung CDF dan transformasi
    cdf = np.cumsum(prob)
    transform = np.round((L - 1) * cdf).astype(np.uint8)

    # Terapkan transformasi
    equalized = transform[gray]

    return gray, equalized

def compare_before_after(image, title="Histogram Equalization"):
    """Tampilkan gambar & histogram sebelum–sesudah dan kembalikan hasil equalized"""
    gray, eq = manual_histogram_equalization(image)
```

```
    gray, eq = manual_histogram_equalization(image)

    # === Visualisasi (opsional) ===
    fig, axes = plt.subplots(2, 2, figsize=(10, 6))
    fig.suptitle(f"{title} - Before & After", fontsize=14, fontweight='bold')

    axes[0, 0].imshow(gray, cmap='gray')
    axes[0, 0].set_title("Original")
    axes[0, 0].axis('off')

    axes[0, 1].imshow(eq, cmap='gray')
    axes[0, 1].set_title("Equalized")
    axes[0, 1].axis('off')

    axes[1, 0].hist(gray.flatten(), bins=256, range=[0, 256], color='blue', alpha=0.7)
    axes[1, 0].set_title("Histogram Original")

    axes[1, 1].hist(eq.flatten(), bins=256, range=[0, 256], color='green', alpha=0.7)
    axes[1, 1].set_title("Histogram Equalized")

    plt.tight_layout()
    plt.show()

    # Return hasil equalized
    return eq

# Panggil dan simpan hasil equalization
under_eq = compare_before_after(underNatural, "Under Exposed")
over_eq = compare_before_after(overNatural, "Over Exposed")
side_eq = compare_before_after(sideNatural, "Side Lighting")
```





Side Lighting - Before & After

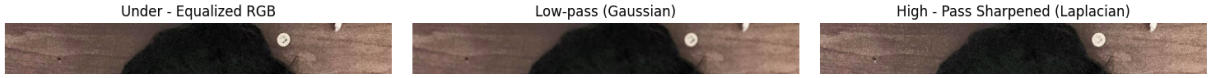


4. Terapkan filter spasial:

4. Terapkan filter spasial:

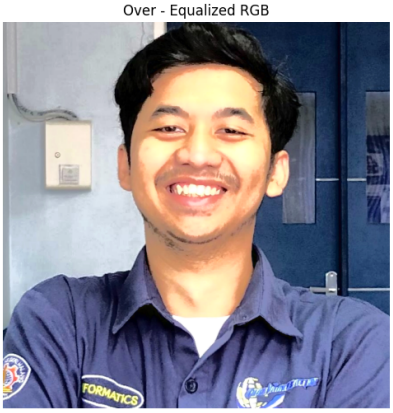
- Low-pass filter untuk menghaluskan noise kulit wajah.
- High-pass atau Laplacian filter untuk menajamkan tepi mata dan bibir.

```
# Fungsi untuk menyisipkan hasil equalization ke channel Y dari citra asli
def inject_equalized_to_rgb(original_rgb, equalized_gray):
    ycrcb = cv.cvtColor(original_rgb, cv.COLOR_BGR2YCrCb)
    ycrcb[:, :, 0] = equalized_gray
    return cv.cvtColor(ycrcb, cv.COLOR_YCrCb2BGR)

# Gabungkan hasil equalization ke RGB
under_eq_rgb = inject_equalized_to_rgb(underNatural, under_eq)
over_eq_rgb = inject_equalized_to_rgb(overNatural, over_eq)
side_eq_rgb = inject_equalized_to_rgb(sideNatural, side_eq)

# LOW-PASS FILTER (Gaussian Blur)
blur_under = cv.GaussianBlur(under_eq_rgb, (5, 5), sigmaX=1)
blur_over = cv.GaussianBlur(over_eq_rgb, (5, 5), sigmaX=1)
blur_side = cv.GaussianBlur(side_eq_rgb, (5, 5), sigmaX=1)

# HIGH-PASS FILTER (Laplacian)
lap_under = cv.Laplacian(under_eq_rgb, cv.CV_64F)
lap_under = cv.convertScaleAbs(lap_under)

lap_over = cv.Laplacian(over_eq_rgb, cv.CV_64F)
lap_over = cv.convertScaleAbs(lap_over)

lap_side = cv.Laplacian(side_eq_rgb, cv.CV_64F)
lap_side = cv.convertScaleAbs(lap_side)

# SHARPENED IMAGE (combine Laplacian with original)
sharpened_under = cv.addWeighted(under_eq_rgb, 1.0, lap_under, 0.3, 0)
sharpened_over = cv.addWeighted(over_eq_rgb, 1.0, lap_over, 0.3, 0)
sharpened_side = cv.addWeighted(side_eq_rgb, 1.0, lap_side, 0.3, 0)

# Fungsi visualisasi berdampingan
def show_side_by_side(images, titles, figsize=(15,5)):
    plt.figure(figsize=figsize)
    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(1, len(images), i+1)
        if len(img.shape) == 2:
            plt.imshow(img, cmap='gray')
        else:
            plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis("off")
    plt.tight_layout()
    plt.show()

# Tampilkan hasil
show_side_by_side([under_eq_rgb, blur_under, sharpened_under],
                  ["Under - Equalized RGB", "Low-pass (Gaussian)", "High - Pass Sharpene

show_side_by_side([over_eq_rgb, blur_over, sharpened_over],
                  ["Over - Equalized RGB", "Low-pass (Gaussian)", "High - Pass Sharpened

show_side_by_side([side_eq_rgb, blur_side, sharpened_side],
                  ["Side - Equalized RGB", "Low-pass (Gaussian)", "High - Pass Sharpened
```

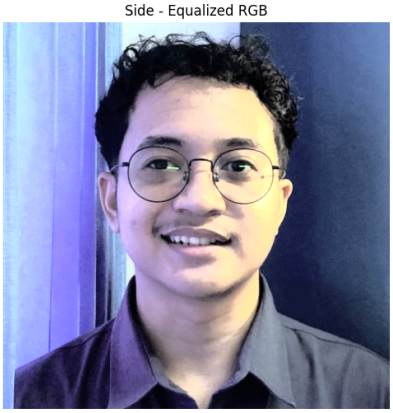
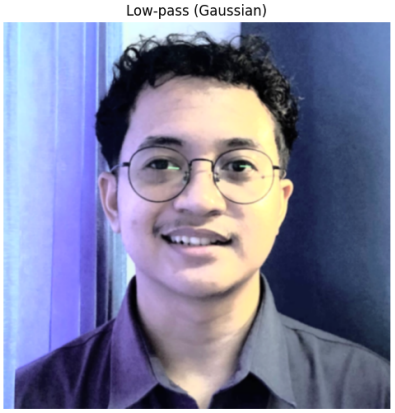




Over - Equalized RGB | Low-pass (Gaussian) | High - Pass Sharpened (Laplacian)



Side - Equalized RGB | Low-pass (Gaussian) | High - Pass Sharpened (Laplacian)

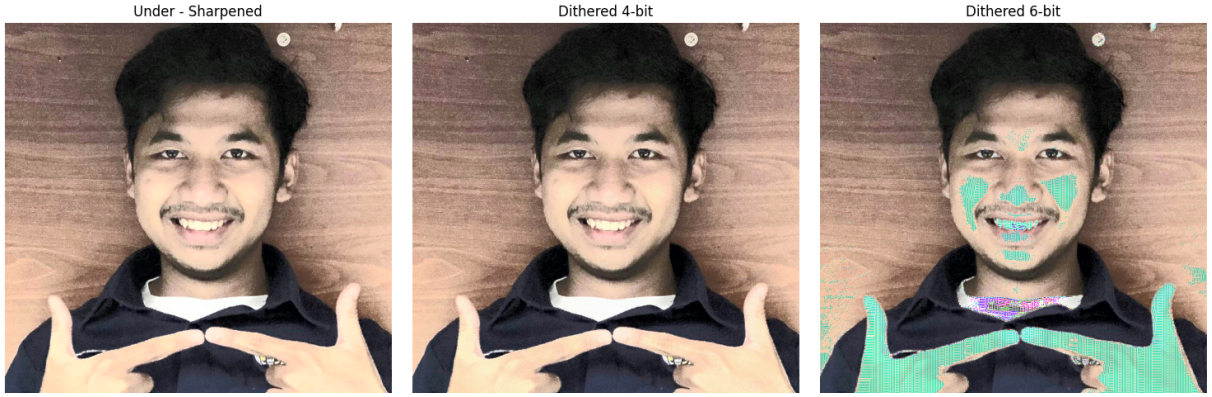



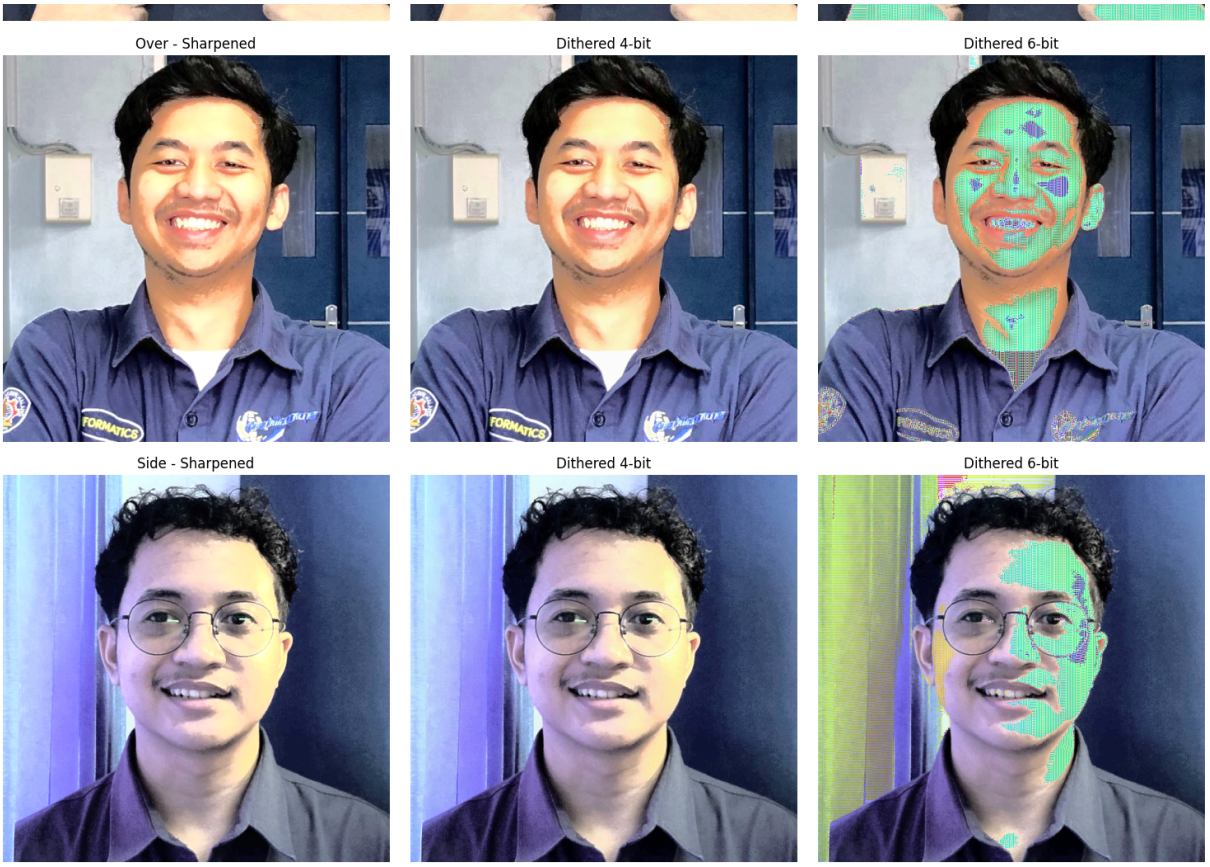

5. Implementasikan Floyd–Steinberg Dithering untuk menurunkan kedalaman warna

wajah (bit-depth 4–6 bit), lalu analisis bagaimana efeknya terhadap detail dan ekspresi wajah.

```
# Fungsi threshold agar nilai tetap dalam rentang 0–255
def threshold(value):
    return max(0, min(255, int(value)))

# Floyd–Steinberg Dithering untuk RGB image
def floyd_steinberg_dithering_rgb(image, bit_depth):
    levels = 2 ** bit_depth
    step = 255 // (levels - 1)

    img = image.astype(np.float32)
    h, w, c = img.shape

    for y in range(h - 1):
        for x in range(1, w - 1):
            for ch in range(3):  # R, G, B
                old = img[y, x, ch]
                new = round(old / step) * step
                img[y, x, ch] = new
                error = old - new

                # Sebarkan error ke tetangga
                img[y, x + 1, ch]     = threshold(img[y, x + 1, ch]     + error * 7 / 16
                img[y + 1, x - 1, ch] = threshold(img[y + 1, x - 1, ch] + error * 3 / 16
                img[y + 1, x, ch]     = threshold(img[y + 1, x, ch]     + error * 5 / 16
                img[y + 1, x + 1, ch] = threshold(img[y + 1, x + 1, ch] + error * 1 / 16

    return img.astype(np.uint8)

# Inject equalized grayscale ke RGB
def inject_equalized_to_rgb(original_rgb, equalized_gray):
    ycrcb = cv.cvtColor(original_rgb, cv.COLOR_BGR2YCrCb)
    ycrcb[:, :, 0] = equalized_gray
    return cv.cvtColor(ycrcb, cv.COLOR_YCrCb2BGR)

# Histogram Equalization
gray_under, eq_under = manual_histogram_equalization(underNatural)
gray_over, eq_over = manual_histogram_equalization(overNatural)
gray_side, eq_side = manual_histogram_equalization(sideNatural)

```

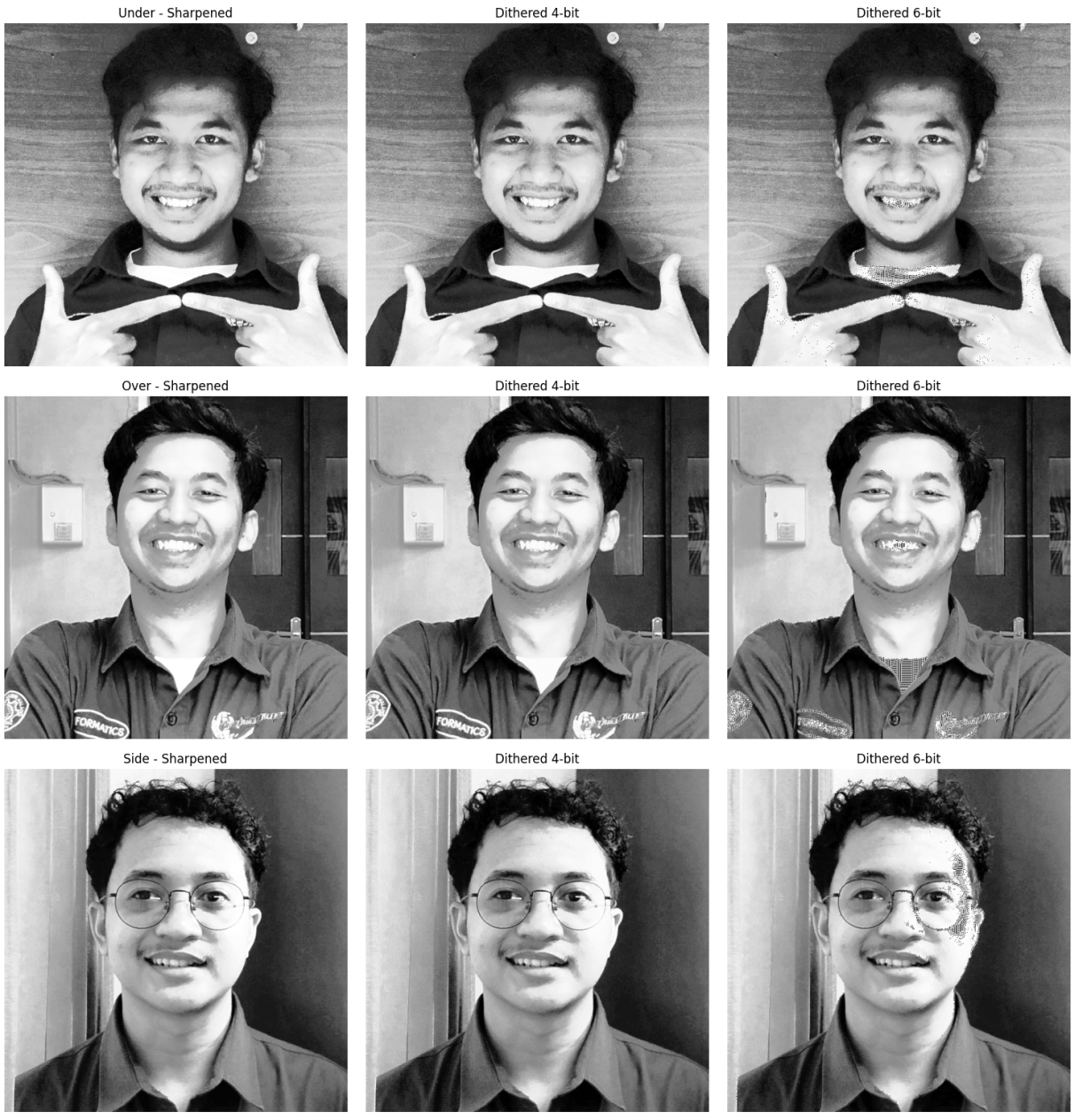
```
eq_under_rgb = inject_equalized_to_rgb(underNatural, eq_under)
eq_over_rgb = inject_equalized_to_rgb(overNatural, eq_over)
eq_side_rgb = inject_equalized_to_rgb(sideNatural, eq_side)

# Laplacian (High-pass)
lap_under = cv.Laplacian(eq_under_rgb, cv.CV_64F)
lap_under = cv.convertScaleAbs(lap_under)
lap_over = cv.Laplacian(eq_over_rgb, cv.CV_64F)
lap_over = cv.convertScaleAbs(lap_over)
lap_side = cv.Laplacian(eq_side_rgb, cv.CV_64F)
lap_side = cv.convertScaleAbs(lap_side)

# Sharpened image
sharpened_under = cv.addWeighted(eq_under_rgb, 1.0, lap_under, 0.3, 0)
sharpened_over = cv.addWeighted(eq_over_rgb, 1.0, lap_over, 0.3, 0)
sharpened_side = cv.addWeighted(eq_side_rgb, 1.0, lap_side, 0.3, 0)

# Dithering dari hasil sharpened
under_4bit = floyd_steinberg_dithering_rgb(sharpened_under, 4)
under_6bit = floyd_steinberg_dithering_rgb(sharpened_under, 6)

over_4bit = floyd_steinberg_dithering_rgb(sharpened_over, 4)
over_6bit = floyd_steinberg_dithering_rgb(sharpened_over, 6)

side_4bit = floyd_steinberg_dithering_rgb(sharpened_side, 4)
side_6bit = floyd_steinberg_dithering_rgb(sharpened_side, 6)

# Visualisasi
def show_side_by_side(images, titles, figsize=(15,5)):
    plt.figure(figsize=figsize)
    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(1, len(images), i+1)
        if len(img.shape) == 2:
            plt.imshow(img, cmap='gray')
        else:
            plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis("off")
    plt.tight_layout()
    plt.show()

# Tampilkan hasil
show_side_by_side([sharpened_under, under_4bit, under_6bit],
                  ["Under - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])

show_side_by_side([sharpened_over, over_4bit, over_6bit],
                  ["Over - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])

show_side_by_side([sharpened_side, side_4bit, side_6bit],
                  ["Side - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])
```




Over - Sharpened
Dithered 4-bit
Dithered 6-bit
Side - Sharpened
Dithered 4-bit
Dithered 6-bit

```python
# Fungsi visualisasi berdampingan
def show_side_by_side(images, titles, figsize=(15,5)):
    plt.figure(figsize=figsize)
    for i, (img, title) in enumerate(zip(images, titles)):
        plt.subplot(1, len(images), i+1)
        plt.imshow(img, cmap='gray')
        plt.title(title)
        plt.axis("off")
    plt.tight_layout()
    plt.show()

# Fungsi threshold agar nilai tetap dalam rentang 0–255
def threshold(value):
    return max(0, min(255, int(value)))

# Floyd–Steinberg Dithering untuk grayscale image
def floyd_steinberg_dithering_gray(image, bit_depth):
    levels = 2 ** bit_depth
    step = 255 // (levels - 1)

    img = image.astype(np.float32)
    h, w = img.shape

    for y in range(h - 1):
        for x in range(1, w - 1):
            old = img[y, x]
            new = round(old / step) * step
            img[y, x] = new
            error = old - new

            # Sebarkan error ke tetangga
            img[y, x + 1]     = threshold(img[y, x + 1]     + error * 7 / 16)
            img[y + 1, x - 1] = threshold(img[y + 1, x - 1] + error * 3 / 16)
            img[y + 1, x]     = threshold(img[y + 1, x]     + error * 5 / 16)
            img[y + 1, x + 1] = threshold(img[y + 1, x + 1] + error * 1 / 16)

    return img.astype(np.uint8)

# Histogram equalization (grayscale)
gray_under, eq_under = manual_histogram_equalization(underNatural)
gray_over, eq_over = manual_histogram_equalization(overNatural)
gray_side, eq_side = manual_histogram_equalization(sideNatural)

# Buat citra sharpened (grayscale)
lap_under = cv.Laplacian(eq_under, cv.CV_64F)
lap_under = cv.convertScaleAbs(lap_under)
sharpened_under = cv.addWeighted(eq_under, 1.0, lap_under, 0.3, 0)

lap_over = cv.Laplacian(eq_over, cv.CV_64F)
lap_over = cv.convertScaleAbs(lap_over)
sharpened_over = cv.addWeighted(eq_over, 1.0, lap_over, 0.3, 0)

lap_side = cv.Laplacian(eq_side, cv.CV_64F)
lap_side = cv.convertScaleAbs(lap_side)
sharpened_side = cv.addWeighted(eq_side, 1.0, lap_side, 0.3, 0)

# Dithering 4-bit dan 6-bit dari hasil sharpened
```

```
under_4bit = floyd_steinberg_dithering_gray(sharpened_under, 4)
under_6bit = floyd_steinberg_dithering_gray(sharpened_under, 6)

over_4bit = floyd_steinberg_dithering_gray(sharpened_over, 4)
over_6bit = floyd_steinberg_dithering_gray(sharpened_over, 6)

side_4bit = floyd_steinberg_dithering_gray(sharpened_side, 4)
side_6bit = floyd_steinberg_dithering_gray(sharpened_side, 6)

# Tampilkan hasil
show_side_by_side([sharpened_under, under_4bit, under_6bit],
                  ["Under - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])

show_side_by_side([sharpened_over, over_4bit, over_6bit],
                  ["Over - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])

show_side_by_side([sharpened_side, side_4bit, side_6bit],
                  ["Side - Sharpened", "Dithered 4-bit", "Dithered 6-bit"])
```



Bagian B - Analisis

1. Bagaimana perubahan nilai brightness dan contrast memengaruhi hasil histogram wajah gelap?

Kondisi	(α, β)	Citra Sebelum	Citra Sesudah
Under Exposed	(1.8, 20)	Histogram terpusat di kiri (rentang 0–70). Mayoritas piksel gelap, detail wajah hilang.	Histogram bergeser ke kanan (rentang 50–200). Puncak warna RGB menyebar merata.
Over Exposed	(1.15, -20)	Histogram dominan di kanan (150–255). Banyak piksel jenuh di area terang.	Histogram bergeser ke kiri (50–220). Distribusi piksel lebih seimbang.
Side Lighting	(1.15, 20)	Histogram tidak seimbang. Puncak di kiri (gelap) dan kanan (terang).	Histogram bergeser ke kanan dan lebih merata di tenga Nilai RGB lebih seragam.

2. Apakah histogram equalization selalu memperbaiki detail wajah? Jelaskan alasannya dengan contoh hasil Anda

Aspek Analisis	
Pertanyaan	Apakah histogram equalization selalu memperbaiki detail wajah?
Konsep Utama	Histogram equalization berfungsi meratakan distribusi intensitas piksel agar kontras lebih
Kondisi Saat Meningkatkan Detail	Efektif ketika gambar wajah terlalu gelap atau terlalu terang, sehingga detail seperti kerut
Kondisi Saat Tidak Efektif / Merusak Detail	Tidak efektif jika gambar sudah memiliki kontras yang baik; justru bisa menimbulkan noise,
Contoh Hasil / Observasi	Pada wajah yang underexposed (gelap), histogram equalization berhasil menonjolkan mat
Kesimpulan	Histogram equalization tidak selalu memperbaiki detail wajah; efektivitasnya tergantung k

3. Bandingkan hasil low-pass dan high-pass filter pada area wajah — bagian mana yang paling terpengaruh?

Area Wajah	Low-Pass Filter (Gaussian)	High-Pass Filter (Laplacian Sharpened)
Tekstur Kulit	Halus dan smooth, pori-pori dan tekstur kulit menghilang	Detail tekstur kulit lebih terlihat, pori-pori dan c
Rambut	Detail helaian rambut kabur dan menyatu	Setiap helai rambut lebih terpisah dan tajam, te
Garis Tepi (Edges)	Garis tepi wajah, hidung, dan bibir menjadi soft dan blur	Garis tepi sangat tajam dan kontras, kontur waj
Mata & Alis	Detail bulu mata dan alis melunak	Bulu mata dan alis terlihat lebih jelas dan terpis
Warna Kulit	Tone warna lebih merata dan natural	Warna kulit terlihat lebih kontras dengan area h
Kacamata (Image 3)	Frame kacamata terlihat smooth	Frame kacamata sangat tajam, refleksi dan det

4. Mengapa Proses Dithering Bisa Mempertegas atau Justru Mengaburkan Ekspresi Wajah?

   Dithering 4-bit mengaburkan ekspresi wajah karena hanya memiliki 16 level abu-abu yang terlalu sedikit, sehingga quantization error yang besar disebarkan menjadi pola noise kasar yang menutupi detail penting seperti mata, hidung, mulut, dan tekstur kulit. Sebaliknya, dithering 6-bit dengan 64 level abu-abu mampu mempertegas ekspresi karena memiliki cukup banyak tingkat gradasi untuk mempertahankan detail wajah dengan error yang lebih kecil, menghasilkan noise yang lebih halus dan tidak mendominasi fitur wajah.

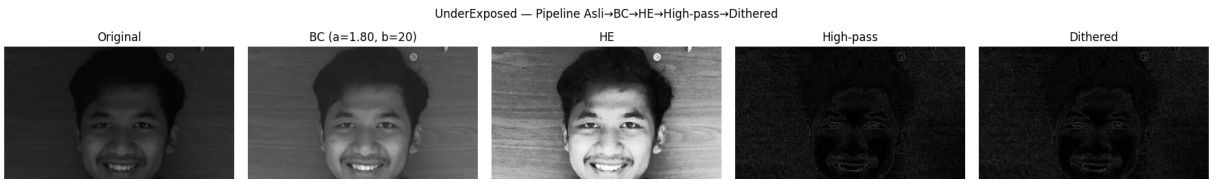
5. Berdasarkan hasil percobaan Anda, kombinasi teknik mana yang paling efektif untuk meningkatkan readability fitur wajah sebelum tahap deteksi?

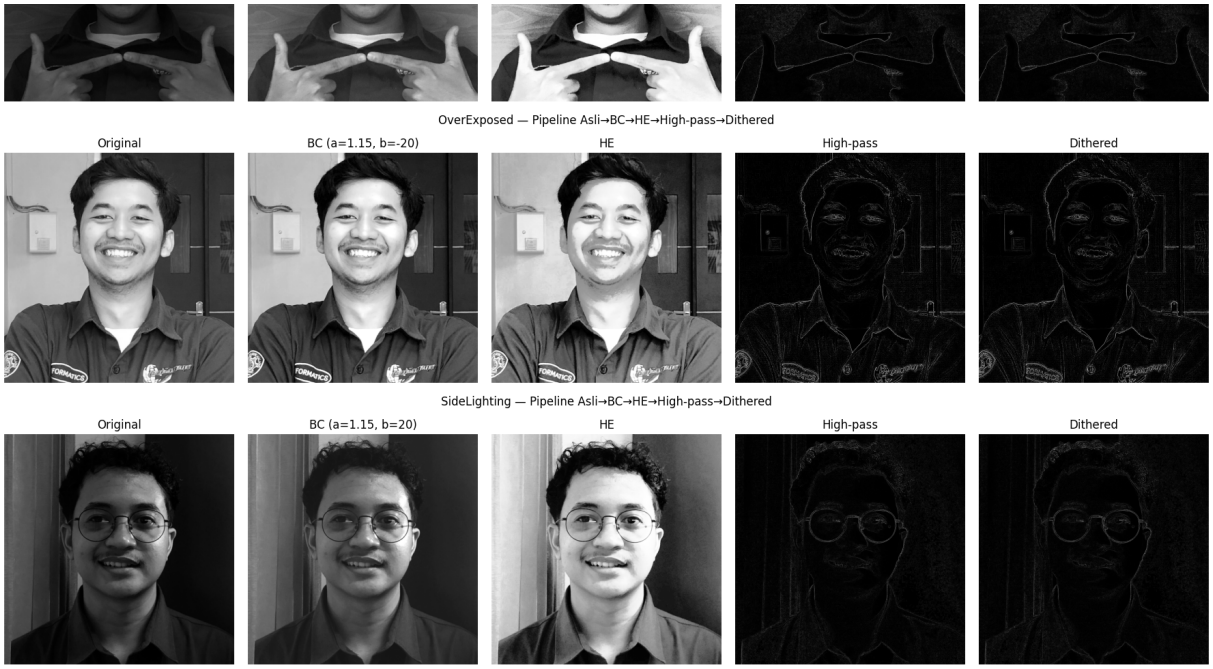
   Terbukti bahwa metode yang paling efektif untuk meningkatkan ketajaman dan kontras fitur wajah adalah kombinasi histogram equalization, diikuti dengan filtering high-pass (Laplacian) dan dithering Floyd–Steinberg. Teknik ini memudahkan deteksi dengan memperjelas kontur ekspresi seperti garis wajah, bibir, dan mata.

Bagian C - Implementasi (Python-OpenCV)

```
# FUNGSI FLOYD-STEINBERG DITHERING
def floyd_steinberg(img, n_bits=5):
    """Floyd-Steinberg dithering untuk grayscale"""
    levels = 2 ** n_bits
    step = 255 / (levels - 1)
    img_float = img.astype(np.float64)
    h, w = img_float.shape

    for y in range(h):
        for x in range(w):
            old_pixel = img_float[y, x]
            new_pixel = np.round(old_pixel / step) * step
            img_float[y, x] = new_pixel
            error = old_pixel - new_pixel

            if x + 1 < w:
                img_float[y, x + 1] += error * 7/16
            if y + 1 < h:
                if x - 1 >= 0:
                    img_float[y + 1, x - 1] += error * 3/16
                img_float[y + 1, x] += error * 5/16
                if x + 1 < w:
                    img_float[y + 1, x + 1] += error * 1/16

```

```
    return np.clip(img_float, 0, 255).astype(np.uint8)


# Memproses gambar
# List gambar yang akan diproses
images_data = [
    {
        'path': '/content/drive/MyDrive/PCVK/UnderExposed.jpg',
        'alpha': 1.8,
        'beta': 20,
        'title': 'UnderExposed'
    },
    {
        'path': '/content/drive/MyDrive/PCVK/OverExposed.jpg',
        'alpha': 1.15,
        'beta': -20,
        'title': 'OverExposed'
    },
    {
        'path': '/content/drive/MyDrive/PCVK/SideLighting.jpg',
        'alpha': 1.15,
        'beta': 20,
        'title': 'SideLighting'
    }
]

# Proses setiap gambar
for img_data in images_data:
    # Baca citra warna dan ubah ke grayscale
    img = cv2.imread(img_data['path'], cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Brightness & Contrast
    b, a = img_data['beta'], img_data['alpha']
    img_bc = cv2.convertScaleAbs(gray, alpha=a, beta=b)

    # Histogram Equalization
    img_he = cv2.equalizeHist(img_bc)

    # Spatial Filtering
    kernel_hp = np.array([[-1, -1, -1],
                          [-1,  8, -1],
                          [-1, -1, -1]], dtype=np.float32)
    img_hp = cv2.filter2D(img_he, -1, kernel_hp)

    # Dithering
    img_dith = floyd_steinberg(img_hp, n_bits=5)

    # Panel 5 tahap: Original → BC → HE → High-pass → Dithered
    fig, axes = plt.subplots(1, 5, figsize=(18, 4))
    axes[0].imshow(gray,    cmap='gray', vmin=0, vmax=255); axes[0].set_title('Original'
    axes[1].imshow(img_bc,  cmap='gray', vmin=0, vmax=255); axes[1].set_title(f'BC (a={a
    axes[2].imshow(img_he,  cmap='gray', vmin=0, vmax=255); axes[2].set_title('HE'); axe
    axes[3].imshow(img_hp,  cmap='gray', vmin=0, vmax=255); axes[3].set_title('High-pass
    axes[4].imshow(img_dith,cmap='gray', vmin=0, vmax=255); axes[4].set_title('Dithered'
    fig.suptitle(f'{img_data["title"]} — Pipeline Asli→BC→HE→High-pass→Dithered', y=1.02
    plt.tight_layout(); plt.show()
```





Bagian D - Analisis Akhir dan Kesimpulan

Tuliskan 1 halaman analisis:

- Urutan proses terbaik untuk menghasilkan wajah paling siap recognition
- Perbandingan nilai PSNR antara citra asli yang memiliki pencahanyaan baik dan hasil pra pemrosesan
- Rekomendasi metode peningkatan kualitas wajah untuk sistem pengenalan wajah real-time

1. Urutan proses terbaik untuk menghasilkan wajah paling siap recognition:

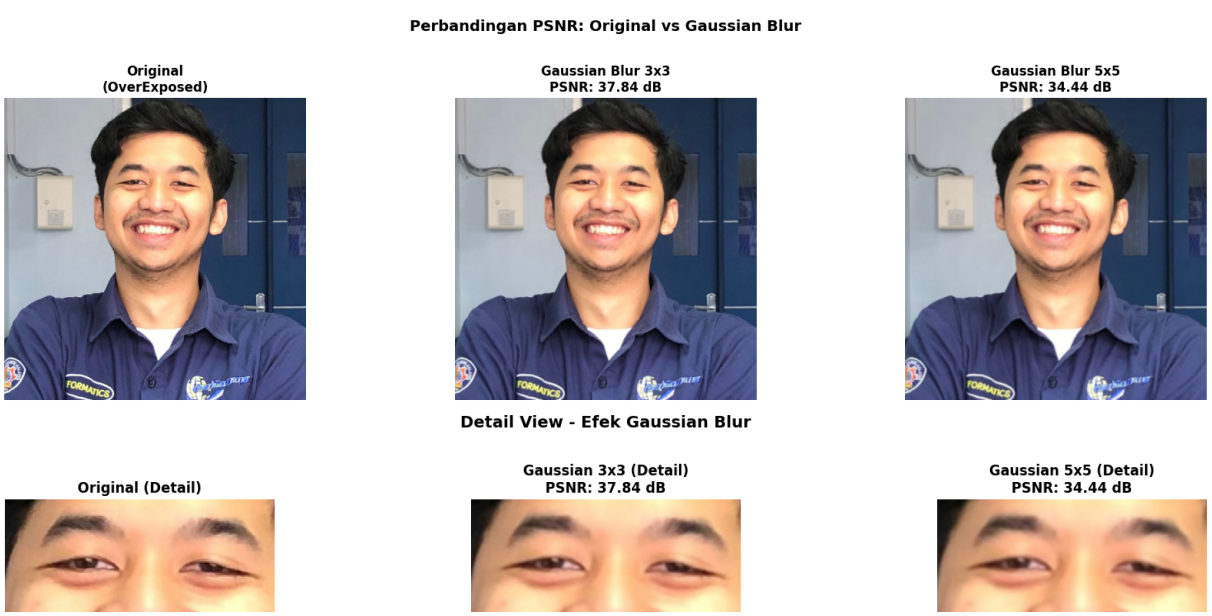
   a. Resizing & Cropping untuk menyeragamkan ukuran wajah agar konsisten di seluruh dataset.

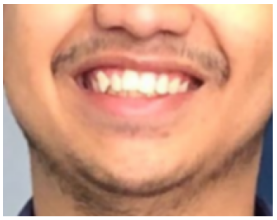
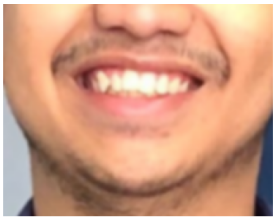
   b. Penyesuaian Brightness dan Contrast (Linear/Log Transform) untuk menormalkan intensitas pencahayaan antar gambar, sehingga area wajah tidak terlalu gelap atau terlalu terang.

   c. Low-pass Filtering untuk mengurangi noise pada kulit wajah tanpa mengaburkan fitur penting. Dengan filter Gaussian berukuran kecil, tekstur kasar akibat pencahayaan tidak merata dapat dihaluskan, sehingga sistem deteksi dapat lebih fokus pada bentuk fitur utama seperti mata dan mulut.

```
# 2. PSNR dengan menggunakan foto dari variable overResized
def calculate_psnr(original, processed):
    """
    Hitung PSNR antara gambar original dan processed
    PSNR = 10 * log10(MAX^2 / MSE)
    """
    mse = np.mean((original.astype(np.float64) - processed.astype(np.float64)) ** 2)

    if mse == 0:
        return float('inf')

    max_pixel = 255.0
    psnr = 10 * np.log10((max_pixel ** 2) / mse)

    return psnr

# Terapkan gaussian blur
blur_3x3 = cv.GaussianBlur(overResized, (3, 3), 0)
blur_5x5 = cv.GaussianBlur(overResized, (5, 5), 0)

# Hitung PSNR
psnr_3x3 = calculate_psnr(overResized, blur_3x3)
psnr_5x5 = calculate_psnr(overResized, blur_5x5)

# Visualisasi side by side
plt.figure(figsize=(18, 5))

# Original
plt.subplot(1, 3, 1)
plt.imshow(cv.cvtColor(overResized, cv.COLOR_BGR2RGB))
plt.title('Original\n(OverExposed)', fontsize=12, fontweight='bold')
plt.axis('off')

# Gaussian 3x3
plt.subplot(1, 3, 2)
plt.imshow(cv.cvtColor(blur_3x3, cv.COLOR_BGR2RGB))
plt.title(f'Gaussian Blur 3x3\nPSNR: {psnr_3x3:.2f} dB',
          fontsize=12, fontweight='bold')
plt.axis('off')

```

```
# Gaussian 5x5
plt.subplot(1, 3, 3)
plt.imshow(cv.cvtColor(blur_5x5, cv.COLOR_BGR2RGB))
plt.title(f'Gaussian Blur 5x5\nPSNR: {psnr_5x5:.2f} dB',
          fontsize=12, fontweight='bold')
plt.axis('off')

plt.suptitle('Perbandingan PSNR: Original vs Gaussian Blur',
             fontsize=14, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()

# Crop area tengah untuk melihat detail
h, w = overResized.shape[:2]
y1, y2 = h//5, h*3//5
x1, x2 = w//3, w*2//3

crop_orig = overResized[y1:y2, x1:x2]
crop_3x3 = blur_3x3[y1:y2, x1:x2]
crop_5x5 = blur_5x5[y1:y2, x1:x2]

plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
plt.imshow(cv.cvtColor(crop_orig, cv.COLOR_BGR2RGB))
plt.title('Original (Detail)', fontsize=12, fontweight='bold')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(cv.cvtColor(crop_3x3, cv.COLOR_BGR2RGB))
plt.title(f'Gaussian 3x3 (Detail)\nPSNR: {psnr_3x3:.2f} dB',
          fontsize=12, fontweight='bold')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cv.cvtColor(crop_5x5, cv.COLOR_BGR2RGB))
plt.title(f'Gaussian 5x5 (Detail)\nPSNR: {psnr_5x5:.2f} dB',
          fontsize=12, fontweight='bold')
plt.axis('off')

plt.suptitle('Detail View - Efek Gaussian Blur',
             fontsize=14, fontweight='bold', y=1.02)
plt.tight_layout()
plt.show()
```



3. Rekomendasi metode peningkatan kualitas wajah untuk sistem pengenalan wajah real-time:

   a. Gamma Correction

      - Menormalkan cahaya secara global

   b. CLAHE (Contrast Limited Adaptive Histogram Equalization)

      - Menyeimbangkan kontras lokal tanpa membuat noise berlebihan.
      - Lebih stabil dibanding histogram equalization biasa.

   c. Normalisasi Intensitas

      - Konversi ke grayscale atau normalisasi piksel ke [0,1].
      - Mengurangi dimensi data → mempercepat inference.