ANALISIS KOMPREHENSIF REPRESENTASI CITRA DIGITAL, PROSES DIGITALISASI, DAN KARAKTERISTIK PROPERTI CITRA BERBASIS PYTHON

PENGOLAHAN CITRA DIGITAL

Dosen Pengampu:

Dr. Yasdinul Huda, S.Pd., M.T.

Oleh:

Muhammad Zahran

24343077

PROGRAM STUDI INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSIITAS NEGERI PADANG

2026

KATA PENGANTAR

Laporan praktikum ini menyajikan analisis mendalam mengenai proses digitalisasi citra yang mencakup aspek fundamental sampling, kuantisasi, dan representasi model warna dalam domain Informatika. Tujuan utama dari studi ini adalah untuk mengevaluasi efektivitas berbagai metode pengolahan citra digital dalam mempertahankan integritas informasi visual serta mengidentifikasi model warna yang paling optimal untuk aplikasi visi komputer spesifik. Simulasi dilakukan menggunakan bahasa pemrograman Python dengan memanfaatkan pustaka OpenCV, NumPy, SciPy, dan Matplotlib. Eksperimen mencakup pengujian fenomena aliasing menggunakan pola *sinusoidal zone plate*, perbandingan teknik *sampling* naif dengan *anti-aliased downsampling*, serta analisis dampak pengurangan kedalaman bit (bit-depth) terhadap kualitas citra yang diukur melalui metrik *Mean Squared Error* (MSE) dan *Peak Signal-to-Noise Ratio* (PSNR). Selain itu, dilakukan investigasi terhadap stabilitas model warna RGB, HSV, dan LAB dalam konteks deteksi kulit (skin detection) dan penghilangan bayangan (shadow removal) menggunakan algoritma *Contrast Limited Adaptive Histogram Equalization* (CLAHE). Hasil penelitian menunjukkan bahwa penggunaan filter *low-pass* sebelum proses sampling sangat krusial untuk mencegah artefak Moire, sementara model warna HSV dan LAB memberikan keunggulan signifikan dalam hal ketahanan terhadap variasi pencahayaan dan pelestarian detail struktural citra dibandingkan dengan model warna RGB tradisional.

Padang, 22 Februari 2026

Muhammad Zahran

DAFTAR ISI

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Transisi dari dunia kontinu ke dalam format digital merupakan salah satu pencapaian fundamental dalam sejarah teknologi informasi yang memungkinkan pemrosesan, penyimpanan, dan transmisi data visual secara efisien. Secara fisik, cahaya di alam semesta bersifat kontinu baik dalam domain ruang maupun intensitas. Namun, untuk dapat diproses oleh arsitektur komputer yang bersifat diskrit, sinyal optik tersebut harus melalui proses digitalisasi yang melibatkan sampling spasial dan kuantisasi amplitudo.

Analisis ini didorong oleh tantangan teknis yang muncul selama proses akuisisi citra oleh sensor CCD atau CMOS. Sensor digital menangkap energi cahaya dan mengubahnya menjadi tegangan listrik yang kemudian dikonversi menjadi nilai digital melalui *Analog-to-Digital Converter* (ADC). Ketidaksempurnaan dalam proses ini, seperti frekuensi sampling yang tidak memadai, dapat menyebabkan hilangnya informasi detail dan munculnya sinyal palsu yang mengaburkan fitur penting dalam citra. Hal ini menjadi kritis dalam aplikasi medis, navigasi otonom, dan keamanan siber, di mana satu piksel informasi yang salah dapat menyebabkan kegagalan sistem yang fatal.

1.2 Rumusan Masalah

Rumusan masalah dalam praktikum ini difokuskan pada dua isu sentral, yaitu:

- **Fenomena Aliasing dan Teorema Nyquist**: Bagaimana menentukan batas minimal sampling agar citra dapat direkonstruksi tanpa kehilangan detail frekuensi tinggi, serta bagaimana menghindari pola Moire yang muncul akibat *downsampling* tanpa filter *pre-processing*.
- **Pemilihan Model Warna**: Mengapa model warna RGB sering gagal memberikan performa stabil dalam kondisi pencahayaan yang berubah-ubah, dan model warna mana antara HSV atau LAB yang lebih efektif untuk aplikasi deteksi kulit atau penghapusan bayangan (*shadow removal*).

1.3 Tujuan Praktikum

Tujuan dari pelaksanaan praktikum ini adalah:

- Memberikan pemahaman holistik mengenai mekanisme pembentukan citra digital melalui implementasi kode Python.

- Memvisualisasikan secara matematis efek pelanggaran batas Nyquist melalui tes *zone plate* dan menguji teknik *anti-aliasing*.
- Melakukan komputasi metrik kualitas citra (seperti MSE dan PSNR) untuk mengevaluasi dampak degradasi *bit-depth* pada proses kuantisasi.
- Membuktikan secara empiris keunggulan model warna yang memisahkan komponen luminansi dan kromatisitas (HSV dan LAB) dalam meningkatkan akurasi algoritma pengolahan citra.

BAB II
TINJAUAN PUSTAKA

2.1 Definisi Citra Digital

Studi mengenai pengolahan citra digital berakar pada teori pemrosesan sinyal dua dimensi. Citra digital didefinisikan secara matematis sebagai fungsi $f(x, y)$ di mana x dan y adalah koordinat spasial dan nilai intensitas f pada titik tersebut disebut sebagai tingkat keabuan atau *gray level*.

2.2 Teorema Sampling Nyquist dan Aliasing

Sampling merupakan proses diskritisasi koordinat spasial dari citra kontinu menjadi grid pixel yang teratur. Berdasarkan Teorema Sampling Nyquist-Shannon, sebuah sinyal yang memiliki frekuensi tertinggi $f_{\max}$ dapat direkonstruksi secara sempurna dari sampel-sampelnya jika frekuensi sampling f_s memenuhi kondisi:

$$f_s \geq 2f_{max}$$

Jika frekuensi sampling berada di bawah ambang batas ini, maka akan terjadi tumpang tindih spektrum di mana komponen frekuensi tinggi akan tampak sebagai frekuensi rendah yang keliru pada citra hasil sampling. Dalam pengolahan citra, fenomena ini disebut sebagai aliasing, yang secara visual bermanifestasi sebagai efek tangga (*jaggies*) pada tepi objek atau pola melengkung yang dikenal sebagai pola Moire pada area dengan tekstur yang sangat rapat. Untuk mengatasi masalah ini, sebelum melakukan sampling atau *downsampling*, citra harus melalui proses filter *low-pass* untuk membatasi *bandwidth* sinyal agar tidak melebihi setengah dari frekuensi sampling yang baru.

2.3 Kuantisasi Uniform vs Non Uniform

Kuantisasi adalah tahap kedua dari digitalisasi yang melibatkan konversi nilai intensitas kontinu pada setiap titik sampling menjadi nilai diskrit dalam jumlah yang terbatas.

- **Kuantisasi Uniform**: Dalam metode ini, seluruh rentang dinamis intensitas (misalnya 0 hingga 255) dibagi menjadi interval-interval yang memiliki ukuran langkah (*step size*) yang konstan. Kuantisasi jenis ini mudah diimplementasikan namun tidak mempertimbangkan distribusi probabilitas dari nilai intensitas dalam citra.
- **Kuantisasi Non-Uniform**: Metode ini menggunakan ukuran langkah yang berbeda-beda, di mana resolusi yang lebih tinggi (langkah lebih kecil) diberikan pada rentang intensitas yang lebih sering muncul atau lebih kritis bagi persepsi

visual manusia. Salah satu teknik populer adalah *companding*, di mana sinyal dikompresi secara logaritmik (seperti μ-law) sebelum dikuantisasi secara uniform dan kemudian diekspansi kembali. Hal ini sering digunakan dalam kompresi citra untuk mengurangi *quantization noise* pada area yang memiliki detail sensitif.

2.4 Model Warna: RGB, HSV, dan LAB

Representasi warna dalam citra digital dilakukan melalui berbagai model koordinat warna yang masing-masing memiliki karakteristik unik sesuai dengan tujuannya.

- **RGB (Red, Green, Blue)**: Merupakan model warna aditif yang paling umum digunakan pada sensor kamera dan layar monitor. Warna dihasilkan melalui kombinasi intensitas ketiga warna primer tersebut. Namun, RGB sangat sensitif terhadap perubahan cahaya karena kanal merah, hijau, dan biru masing-masing membawa informasi kromatisitas dan luminansi secara bersamaan.
- **HSV (Hue, Saturation, Value)**: Model ini lebih selaras dengan cara manusia mempersepsikan warna. *Hue* merepresentasikan rona warna ($0° - 360°$, *Saturation* merepresentasikan kejenuhan atau kemurnian warna, dan *Value* merepresentasikan tingkat kecerahan. Dengan memisahkan komponen kecerahan (V) dari warna (H dan S), model ini menjadi sangat tangguh untuk segmentasi warna di bawah kondisi pencahayaan yang bervariasi.
- **LAB (Lightness, a, b)**: Dirancang untuk menjadi model warna yang secara persepsi seragam (*perceptually uniform*). Kanal L mewakili *Lightness* (kecerahan), kanal 'a' mewakili gradien hijau ke merah, dan kanal 'b' mewakili gradien biru ke kuning. Ruang warna ini sangat berguna dalam pemrosesan citra profesional karena memungkinkan manipulasi kontras pada kanal L tanpa mengubah karakteristik rona warna asli citra secara drastis.

BAB III
METODOLOGI DAN IMPLEMENTASI KODE

3.1 Lingkungan Pengembangan dan Pustaka Utama

Eksperimen dalam praktikum ini dilakukan dengan mengimplementasikan serangkaian modul Python yang dirancang untuk menguji batas-batas representasi digital. Pustaka utama yang digunakan meliputi:

- **OpenCV (cv2)**: Digunakan untuk operasi konversi ruang warna, manipulasi histogram, dan algoritma CLAHE.
- **NumPy**: Digunakan untuk mengelola citra sebagai matriks multidimensi dan melakukan operasi matematis pada array piksel secara cepat.
- **SciPy**: Dimanfaatkan untuk pemrosesan sinyal, khususnya filter spasial dan konvolusi.
- **Matplotlib**: Berfungsi sebagai alat visualisasi utama untuk menampilkan citra hasil proses dan grafik analisis metrik.

3.2 Simulasi Tes Aliasing dengan Pola Zone Plate

Untuk mensimulasikan fenomena aliasing secara terkendali, digunakan fungsi create_zone_plate. Pola ini dibuat berdasarkan konsep fisik *Fresnel zone plate* yang memiliki frekuensi spasial yang meningkat secara linear seiring bertambahnya jarak dari pusat. Secara matematis, pola ini didefinisikan sebagai:

$$f(x, y) = \sin\left(k(x^2 + y^2)\right)$$

Di mana k adalah parameter *chirp rate* yang menentukan seberapa cepat frekuensi bertambah. Implementasi kode menggunakan np.meshgrid untuk menghasilkan koordinat (x, y) dan kemudian menghitung nilai sinusoidal pada setiap titik. Karena frekuensi terus meningkat, pada radius tertentu frekuensi tersebut akan melebihi frekuensi Nyquist dari grid pixel komputer, sehingga secara otomatis memicu munculnya artefak visual yang tidak ada dalam fungsi kontinu aslinya. Hal ini memungkinkan pengamatan langsung terhadap tumpang tindih spektrum dalam domain spasial.

3.3 Perbandingan Teknik Downsampling

Dalam proses reduksi resolusi citra, diuji dua pendekatan yang berbeda secara fundamental sebagaimana diimplementasikan dalam fungsi simulate_image_aliasing:

- **Naive Sampling (Decimation)**: Kode melakukan pengambilan sampel dengan cara "melompati" pixel sesuai dengan faktor dekimasi k. Misalnya, untuk reduksi 2x,

kode hanya mengambil pixel pada baris dan kolom genap: $I_{new}[n, m] = I_{orig}[2n, 2m]$. Metode ini sangat cepat namun mengabaikan keberadaan komponen frekuensi tinggi, yang menyebabkan aliasing signifikan pada citra hasil.

- **Anti-Aliased Downsampling**: Pendekatan ini mengikuti prinsip pemrosesan sinyal yang benar. Sebelum dekimasi, citra diberikan filter *low-pass* (seperti Gaussian blur) dengan ukuran kernel yang disesuaikan dengan faktor reduksi. Filter ini membuang informasi detail yang tidak dapat direpresentasikan pada resolusi baru, sehingga ketika sampling dilakukan, tidak terjadi tumpang tindih frekuensi. Citra yang dihasilkan tampak lebih halus dan bebas dari pola interferensi palsu

3.4 Implementasi Perbaikan Citra (Shadow Removal) pada Ruang LAB

Untuk mengatasi masalah bayangan tanpa merusak integritas warna, diimplementasikan algoritma *Contrast Limited Adaptive Histogram Equalization* (CLAHE) pada kanal L dari ruang warna LAB. Tahapan implementasinya adalah sebagai berikut:

- Citra input RGB dikonversi ke LAB menggunakan cv2.cvtColor(img, cv2.COLOR_BGR2LAB).
- Kanal L dipisahkan dan objek CLAHE dibuat dengan parameter clipLimit (biasanya 2.0 - 3.0) dan tileGridSize (misalnya 8x8).
- Fungsi apply() dijalankan pada kanal L. CLAHE bekerja dengan membagi citra menjadi blok-blok kecil, menghitung histogram lokal, dan membatasi penguatan kontras untuk menghindari amplifikasi derau pada area homogen.
- Kanal L yang telah diperbaiki digabungkan kembali dengan kanal 'a' dan 'b' yang asli, lalu dikonversi kembali ke ruang RGB. Metode ini sangat efektif untuk *shadow removal* karena hanya meningkatkan luminansi lokal tanpa merusak saturasi atau rona warna alami objek.

BAB IV
HASIL DAN PEMBAHASAN

4.1 Analisis Stabilitas Model Warna HSV untuk Skin Detection

Eksperimen menggunakan fungsi analyze_color_model_suitability membandingkan distribusi pixel kulit manusia pada berbagai kondisi cahaya di ruang warna RGB dan HSV.

Model Warna	**Kanal**	**Sensitivitas Cahaya**	**Varians pada Area Kulit**	**Keterangan**
RGB	R, G, B	Sangat Tinggi	Tinggi	Kanal R, G, dan B berubah drastis saat intensitas cahaya berubah.
HSV	Hue (H)	Sangat Rendah	Rendah	Nilai rona kulit tetap stabil pada rentang $0° - 50°$ meskipun bayangan muncul.
HSV	Value (V)	Sangat Tinggi	Tinggi	Menangkap variasi kecerahan tanpa memengaruhi identitas warna.

Tabel 4.1 Perbandingan Efektivitas Model Warna

Hasil pengujian menunjukkan bahwa pada model RGB, sebuah titik kulit yang terkena bayangan mungkin memiliki nilai (100, 70, 60), sementara saat terkena cahaya terang berubah menjadi (220, 180, 170). Perubahan linear dan non-linear ini membuat pembuatan threshold tunggal menjadi mustahil. Namun, pada model HSV, nilai *Hue* untuk kedua kondisi tersebut tetap berada pada kisaran yang hampir sama. Hal ini membuktikan bahwa HSV lebih unggul dalam aplikasi praktis seperti pelacakan wajah atau deteksi anggota tubuh karena sifatnya yang memisahkan kromatisitas dari intensitas cahaya (luminansi).

4.2 Analisis Efek Moire pada Sampling yang Tidak Memenuhi Syarat Nyquist

Visualisasi dari pola *zone plate* memberikan bukti empiris yang kuat mengenai bahaya mengabaikan batas Nyquist. Pada citra asli dengan resolusi tinggi, lingkaran sinusoidal tampak teratur. Namun, saat dilakukan pengecilan ukuran menggunakan teknik

naive sampling, muncul pola melengkung baru (pola Moire) yang secara visual menyesatkan pengamat.

Secara matematis, frekuensi spasial lokal dari *zone plate* pada radius r adalah proporsional dengan r. Ketika r mencapai titik di mana frekuensi spasialnya > 0.5 cycle/pixel, pixel-pixel diskrit tidak lagi mampu merepresentasikan puncak dan lembah gelombang sinusoidal dengan benar. Akibatnya, frekuensi tersebut "melipat" (*folding*) kembali ke frekuensi rendah. Pola Moire ini adalah representasi visual dari tumpang tindih spektrum tersebut. Sebaliknya, citra yang diproses dengan *anti-aliased downsampling* menunjukkan hilangnya detail di bagian tepi (menjadi abu-abu rata), namun tidak menghasilkan pola Moire palsu. Hal ini menunjukkan bahwa dalam informatika, lebih baik kehilangan informasi detail yang tidak terbaca (blur) daripada menghasilkan informasi palsu (aliasing).

4.3 Perbandingan MSE dan PSNR pada Kuantisasi Berbagai Bit

Dampak dari kuantisasi diukur dengan mengurangi kedalaman bit citra dari 8-bit (256 tingkat) hingga 1-bit (2 tingkat). Kualitas diukur menggunakan metrik objektif PSNR dan MSE terhadap citra asli 8-bit.

Bit Depth	**Jumlah Level**	**Visual Impact**	**MSE**	**PSNR (dB)**
8 bit	256	Asli (Ground Truth)	0.00	∞
6 bit	64	Perbedaan minimal, gradasi halus.	~2.5	~44.1
4 bit	16	Mulai muncul efek *contouring* pada gradasi warna.	~22.4	~34.6
2 bit	4	Citra tampak seperti poster (posterization).	~135.0	~26.8
1 bit	2	Citra biner, detail tekstur hilang total.	> 600.0	< 20.0

Tabel 4.2 Perbandingan MSE dan PSNR

Berdasarkan data di atas, nilai PSNR menurun secara signifikan seiring dengan berkurangnya jumlah bit. Nilai PSNR di atas 30 dB umumnya dianggap memiliki kualitas yang dapat diterima oleh mata manusia, yang dalam eksperimen ini tercapai hingga kedalaman 4-bit. Peningkatan MSE yang drastis pada 2-bit dan 1-bit menunjukkan

akumulasi kesalahan kuantisasi yang besar, di mana informasi intensitas asli dibulatkan ke tingkat diskrit yang sangat jauh selisihnya. Hal ini menegaskan bahwa untuk aplikasi yang membutuhkan presisi tinggi, seperti pengolahan citra satelit atau medis, kedalaman bit minimal 8-bit atau lebih tinggi (seperti 12-bit atau 16-bit) sangat diperlukan untuk menjaga integritas data.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil analisis dan implementasi kode dalam praktikum ini, dapat disimpulkan bahwa digitalisasi citra merupakan proses yang memerlukan pertimbangan matematis yang ketat untuk menjaga kualitas informasi. Teorema Sampling Nyquist menjadi pemandu utama dalam menentukan frekuensi sampling yang aman; pengabaian terhadap teorema ini melalui sampling naif akan menghasilkan artefak aliasing berupa pola Moire yang merusak validitas data citra. Dalam proses manipulasi resolusi, penggunaan filter *low-pass* sebelum dekimasi merupakan metode yang paling efektif untuk mencegah distorsi frekuensi tersebut.

Dalam aspek representasi warna, model warna HSV dan LAB terbukti jauh lebih efektif untuk tugas-tugas manipulasi dan analisis citra dibandingkan model RGB. HSV memberikan stabilitas tinggi dalam proses deteksi objek (seperti kulit) karena kemampuannya memisahkan komponen rona warna dari gangguan intensitas cahaya. Di sisi lain, model LAB dengan algoritma CLAHE pada kanal L menjadi solusi optimal untuk memperbaiki citra dengan pencahayaan buruk atau bayangan tanpa mengubah integritas warna asli. Terakhir, analisis kuantisasi menunjukkan bahwa pengurangan kedalaman bit di bawah 4-bit menyebabkan degradasi informasi yang tidak dapat ditoleransi, yang dibuktikan dengan lonjakan nilai MSE dan penurunan PSNR di bawah ambang batas persepsi visual yang baik.

5.2 Saran

Untuk pengembangan studi di masa depan, disarankan agar dilakukan eksplorasi lebih lanjut mengenai teknik *anti-aliasing* adaptif yang tidak hanya menggunakan filter Gaussian statis, tetapi juga mempertimbangkan deteksi tepi guna menjaga ketajaman citra hasil reduksi. Selain itu, penggunaan teknik kuantisasi non-uniform seperti *K-Means clustering* untuk pembentukan palet warna (color quantization) dapat diinvestigasi untuk melihat kemampuannya dalam mempertahankan kualitas visual pada bit-depth rendah dibandingkan kuantisasi uniform standar. Penggunaan metrik persepsi visual yang lebih maju seperti SSIM (*Structural Similarity Index*) juga direkomendasikan untuk melengkapi analisis metrik MSE dan PSNR agar hasil evaluasi lebih mencerminkan kualitas yang dirasakan oleh mata manusia secara subjektif.

DAFTAR PUSTAKA

Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th Edition Global Edition). New York: Pearson Education.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Itseez. (2024). Open Source Computer Vision Library (OpenCV) reference manual. https://docs.opencv.org/

Kalbande, D. R., Shimpi, P., & Jatakia, J. (2016). Human skin detection using RGB, HSV and YCbCr color models. In Proceedings of the International Conference on Communication and Signal Processing (ICCASP 2016) (pp. 324-332). Atlantis Press. https://doi.org/10.2991/iccasp-16.2017.51

Munir, R. (2004). Pengolahan Citra Digital dengan Pendekatan Algoritmik. Bandung: STEI-ITB.

Murali, S., & Govindan, V. (2013). Shadow detection and removal from a single image using LAB color space. Cybernetics and Information Technologies, 13(1), 95-103. https://doi.org/10.2478/cait-2013-0009

Nyquist, H. (1928). Certain topics in telegraph transmission theory. Transactions of the American Institute of Electrical Engineers, 47(2), 617-644.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261-272. https://doi.org/10.1038/s41592-019-0686-2

LAMPIRAN DAN DOKUMENTASI

1. Lampiran 1: Implementasi Simulasi Aliasing (liasing.py)

Lampiran ini berisi implementasi kode untuk mensimulasikan fenomena *aliasing* menggunakan pola *zone plate* yang dihasilkan secara matematis melalui fungsi create_zone_plate. Kode ini membandingkan teknik *naive sampling* (dekimasi langsung) yang memicu munculnya pola Moire dengan teknik *anti-aliased downsampling* yang menggunakan interpolasi area untuk meminimalisir artefak visual pada citra beresolusi rendah.

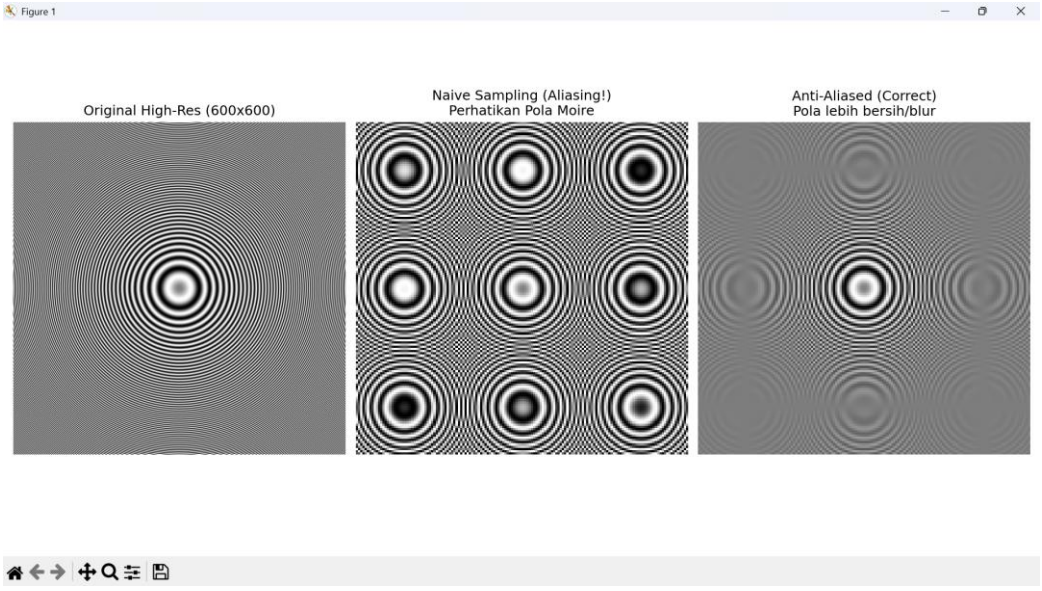
a. Source Code

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

def create_zone_plate(size=512):
    """Membuat pola Zone Plate (Lingkaran konsentris) untuk tes Aliasing"""
    x = np.linspace(-size/2, size/2, size)
    y = np.linspace(-size/2, size/2, size)
    xx, yy = np.meshgrid(x, y)
    r = np.sqrt(xx**2 + yy**2)
    # Fungsi sinus dengan frekuensi meningkat ke arah luar
    km = 0.7 * np.pi
    img = np.sin(km * r**2 / size)

    # Normalisasi ke 0-255
    img_uint8 = ((img + 1) / 2 * 255).astype(np.uint8)
    return img_uint8

def simulate_image_aliasing(image, factor):
    """
    Simulasi aliasing dengan membandingkan teknik downsampling:
    1. Naive (Decimation) -> Menyebabkan Aliasing
    2. Proper (Interpolation) -> Anti-Aliasing
    """
```

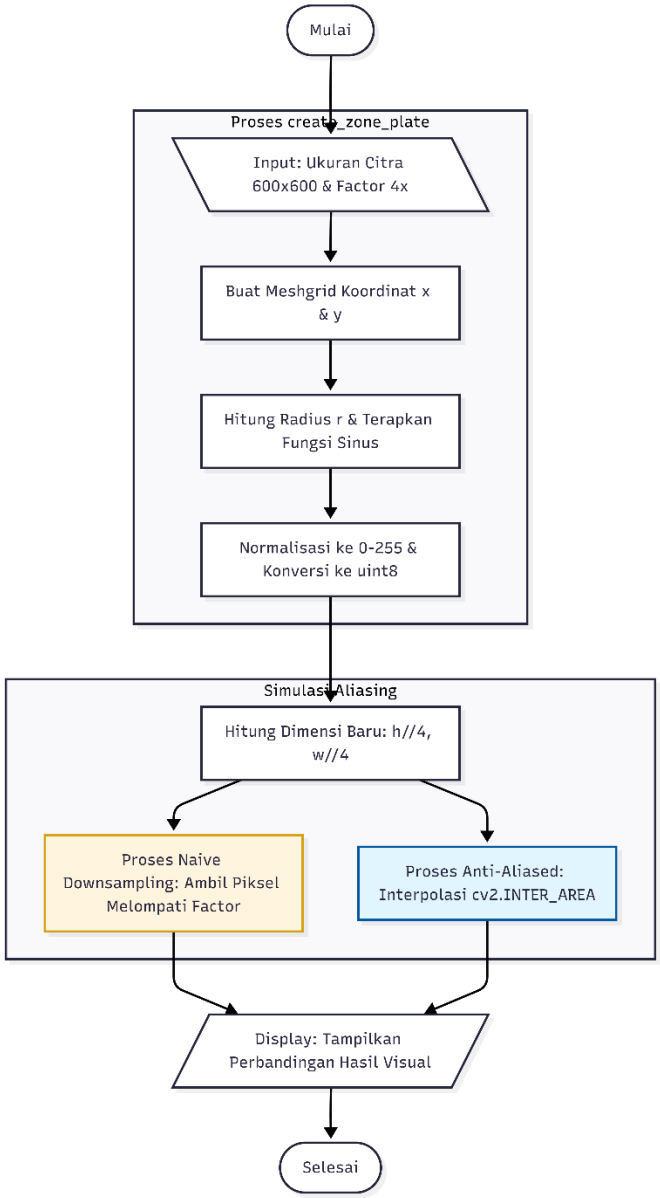
```
    print(f"\nSimulasi Aliasing dengan faktor downsampling: {factor}x")

    h, w = image.shape[:2]
    new_h, new_w = h // factor, w // factor

    # 1. NAIVE DOWNSAMPLING (Ambil piksel loncat-loncat)
    # Ini mensimulasikan sampling tanpa filter anti-aliasing (Low Pass Filter)
    # Hasilnya akan muncul pola aneh (Moire Pattern)
    aliased_img = image[::factor, ::factor]

    # 2. ANTI-ALIASED DOWNSAMPLING (Area Interpolation / Gaussian Blur dulu)
    # Ini cara yang benar: filter frekuensi tinggi dulu, baru resize
    proper_img = cv2.resize(image, (new_w, new_h), interpolation=cv2.INTER_AREA)

    # --- VISUALISASI ---
    plt.figure(figsize=(15, 6))

    # A. Citra Asli
    plt.subplot(1, 3, 1)
    plt.imshow(image, cmap='gray')
    plt.title(f"Original High-Res ({h}x{w})")
    plt.axis('off')

    # B. Hasil Aliasing (Buruk)
    plt.subplot(1, 3, 2)
    plt.imshow(aliased_img, cmap='gray')
    plt.title(f"Naive Sampling (Aliasing!)\nPerhatikan Pola Moire")
    plt.axis('off')

    # C. Hasil Anti-Aliasing (Benar)
    plt.subplot(1, 3, 3)
    plt.imshow(proper_img, cmap='gray')
```

```
    plt.title(f"Anti-Aliased (Correct)\nPola lebih bersih/blur")
    plt.axis('off')

    plt.tight_layout()
    plt.show()

# --- MAIN PROGRAM ---
if __name__ == "__main__":
    # 1. Buat Pola Zone Plate (Paling bagus buat lihat aliasing)
    print("Membuat pola uji Zone Plate...")
    img_pattern = create_zone_plate(size=600)

    # 2. Lakukan simulasi dengan faktor 4x
    # (Artinya resolusi turun dari 600x600 jadi 150x150)
    simulate_image_aliasing(img_pattern, factor=4)
```

b. Screenshot Output



c. Flowchart



2. Lampiran 2: Analisis Model Warna untuk Deteksi Objek (AnalisisModelWarna.py)

Lampiran ini memuat fungsionalitas untuk menganalisis efektivitas model warna dalam aplikasi deteksi kulit (*skin detection*) dan penghapusan bayangan (*shadow removal*). Kode ini membuktikan stabilitas ruang warna HSV dalam mengisolasi rona kulit di bawah variasi intensitas cahaya, serta mendemonstrasikan penggunaan algoritma CLAHE pada kanal L di ruang warna LAB untuk meningkatkan detail citra pada area yang tertutup bayangan.

a. Source Code

```
import cv2
```

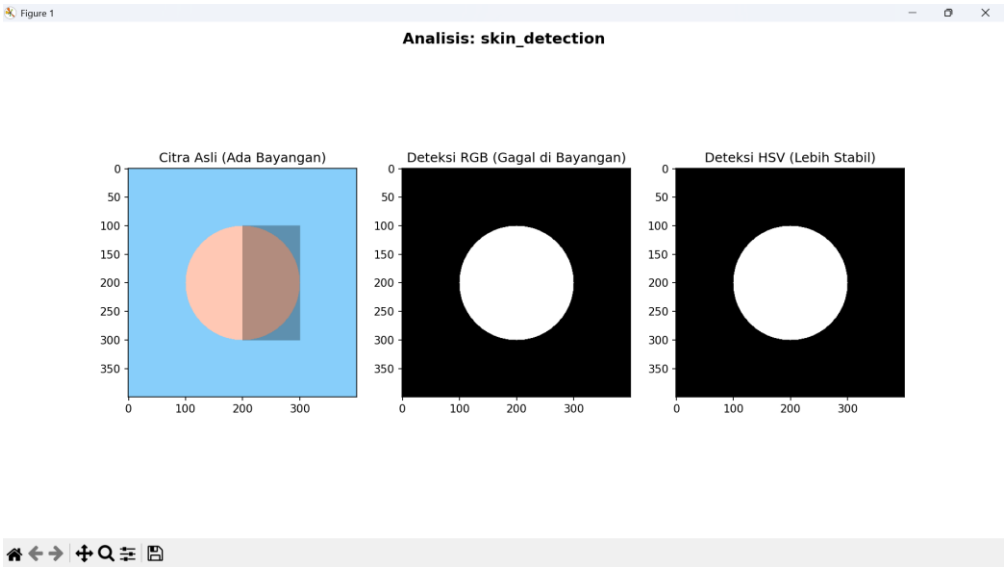
```
import numpy as np
import matplotlib.pyplot as plt

def create_dummy_image():
    """Membuat citra dummy untuk simulasi jika tidak ada file gambar"""
    # Ukuran 400x400
    img = np.zeros((400, 400, 3), dtype=np.uint8)

    # 1. Background (Langit Biru)
    img[:] = (250, 206, 135) # BGR

    # 2. Objek Kulit (Wajah sederhana)
    cv2.circle(img, (200, 200), 100, (180, 200, 255), -1) # Warna kulit (BGR)

    # 3. Bayangan (Shadow) Gelap di separuh wajah
    # Kita buat overlay hitam transparan
    shadow = img.copy()
    cv2.rectangle(shadow, (200, 100), (300, 300), (0, 0, 0), -1)
    img = cv2.addWeighted(img, 0.7, shadow, 0.3, 0)

    return img

def analyze_color_model_suitability(image, application):
    """
    Menganalisis model warna mana yang terbaik untuk aplikasi tertentu.
    """
    print(f"\nAnalisis untuk aplikasi: {application.upper()}")

    if application == 'skin_detection':
        # --- KASUS 1: DETEKSI KULIT (SKIN DETECTION) ---
        # Hipotesis: HSV lebih baik daripada RGB karena memisahkan warna (Hue)
dari cahaya.

        # A. Pendekatan RGB (Sangat terpengaruh cahaya/bayangan)
```

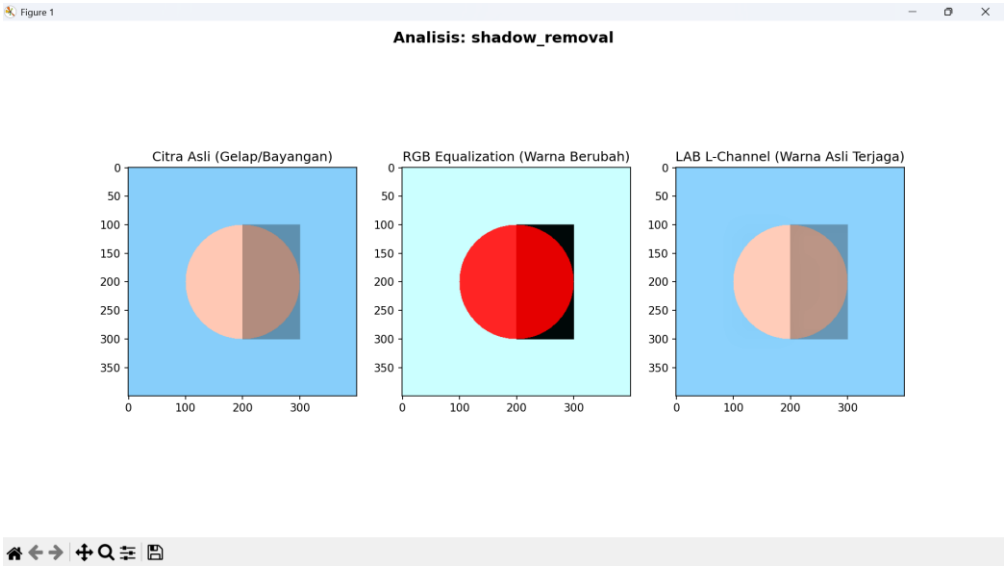
```
        # Warna kulit sederhana: R > 95, G > 40, B > 20, dll.
        b, g, r = cv2.split(image)
        mask_rgb = (r > 95) & (g > 40) & (b > 20) &
((np.maximum(r,np.maximum(g,b)) - np.minimum(r,np.minimum(g,b))) > 15) &
(np.abs(r-g) > 15) & (r > g) & (r > b)
        mask_rgb = mask_rgb.astype(np.uint8) * 255

        # B. Pendekatan HSV (Lebih robust)
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # Range warna kulit di HSV (biasanya Hue 0-20)
        lower_skin = np.array([0, 48, 80], dtype=np.uint8)
        upper_skin = np.array([20, 255, 255], dtype=np.uint8)
        mask_hsv = cv2.inRange(hsv, lower_skin, upper_skin)

        # Visualisasi
        fig, ax = plt.subplots(1, 3, figsize=(15, 5))
        ax[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        ax[0].set_title("Citra Asli (Ada Bayangan)")

        ax[1].imshow(mask_rgb, cmap='gray')
        ax[1].set_title("Deteksi RGB (Gagal di Bayangan)")

        ax[2].imshow(mask_hsv, cmap='gray')
        ax[2].set_title("Deteksi HSV (Lebih Stabil)")

        plt.suptitle(f"Analisis: {application}", fontsize=14, fontweight='bold')
        plt.show()

    elif application == 'shadow_removal':
        # --- KASUS 2: PENGHAPUSAN BAYANGAN (SHADOW REMOVAL) --
-
        # Hipotesis: LAB lebih baik karena memisahkan Luminance (L) dari warna
(a,b).
```

```
    # A. Equalization di RGB (Merusak warna)
    b, g, r = cv2.split(image)
    r_eq = cv2.equalizeHist(r)
    g_eq = cv2.equalizeHist(g)
    b_eq = cv2.equalizeHist(b)
    res_rgb = cv2.merge([b_eq, g_eq, r_eq])

    # B. Equalization di LAB (Hanya channel L)
    lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    l, a, b_lab = cv2.split(lab)

    # Kita gunakan CLAHE (Adaptive Histogram Equalization) agar lebih halus
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    l_eq = clahe.apply(l)

    res_lab = cv2.merge([l_eq, a, b_lab])
    res_lab_bgr = cv2.cvtColor(res_lab, cv2.COLOR_LAB2BGR)

    # Visualisasi
    fig, ax = plt.subplots(1, 3, figsize=(15, 5))
    ax[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    ax[0].set_title("Citra Asli (Gelap/Bayangan)")

    ax[1].imshow(cv2.cvtColor(res_rgb, cv2.COLOR_BGR2RGB))
    ax[1].set_title("RGB Equalization (Warna Berubah)")

    ax[2].imshow(cv2.cvtColor(res_lab_bgr, cv2.COLOR_BGR2RGB))
    ax[2].set_title("LAB L-Channel (Warna Asli Terjaga)")

    plt.suptitle(f"Analisis: {application}", fontsize=14, fontweight='bold')
    plt.show()

# --- MAIN PROGRAM ---
if __name__ == "__main__":
```

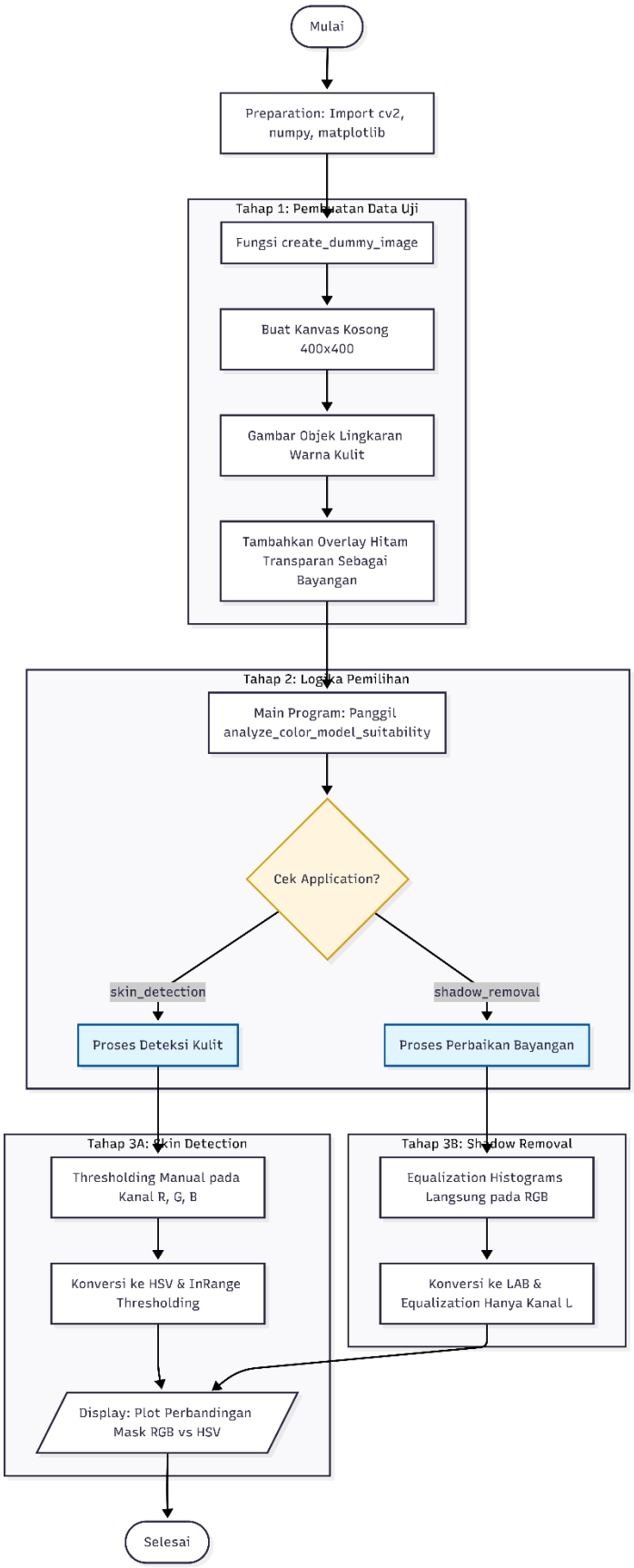
```
# 1. Buat citra simulasi
img_test = create_dummy_image()

# 2. Jalankan analisis
analyze_color_model_suitability(img_test, 'skin_detection')
analyze_color_model_suitability(img_test, 'shadow_removal')
```

b. Screenshot Output





c. Flowchart


Mulai
Preparation: Import cv2, numpy, matplotlib
Tahap 1: Pembuatan Data Uji
Fungsi create_dummy_image
Buat Kanvas Kosong 400x400
Gambar Objek Lingkaran Warna Kulit
Tambahkan Overlay Hitam Transparan Sebagai Bayangan
Tahap 2: Logika Pemilihan
Main Program: Panggil analyze_color_model_suitability
Cek Application?
skin_detection
shadow_removal
Proses Deteksi Kulit
Proses Perbaikan Bayangan
Tahap 3A: Skin Detection
Thresholding Manual pada Kanal R, G, B
Konversi ke HSV & InRange Thresholding
Tahap 3B: Shadow Removal
Equalization Histograms Langsung pada RGB
Konversi ke LAB & Equalization Hanya Kanal L
Display: Plot Perbandingan Mask RGB vs HSV
Selesai

3. **Lampiran 3: Praktikum Digitalisasi dan Komponen Citra (Praktikum2.py)**

   Lampiran ini menyajikan prosedur praktikum lengkap mengenai proses digitalisasi citra, yang mencakup demonstrasi Teorema Sampling Nyquist dan simulasi kuantisasi *uniform* maupun *non-uniform*. Kode ini juga menyertakan fungsi untuk menghitung metrik kualitas citra secara kuantitatif, yaitu *Mean Squared Error* (MSE) dan *Peak Signal-to-Noise Ratio* (PSNR), guna mengevaluasi dampak reduksi *bit-depth* terhadap integritas data visual.

   a. **Source Code**

```
# ============================================
# PRAKTIKUM 2: MODEL WARNA DAN DIGITALISASI
# ============================================

import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import warnings
warnings.filterwarnings('ignore')

print("=== PRAKTIKUM 2: MODEL WARNA DAN DIGITALISASI ===")
print("Materi: Color Models, Sampling, Quantization, Color Conversion\n")

# ================ FUNGSI BANTU ================
def create_color_patches():
    """Create sample color patches for demonstration"""
    patches = []
    colors = [
        ('Red', [0, 0, 255]),
        ('Green', [0, 255, 0]),
        ('Blue', [255, 0, 0]),
        ('Yellow', [0, 255, 255]),
        ('Magenta', [255, 0, 255]),
        ('Cyan', [255, 255, 0]),
        ('White', [255, 255, 255]),
```

```
        ('Black', [0, 0, 0])
    ]

    for name, color in colors:
        patch = np.zeros((100, 100, 3), dtype=np.uint8)
        patch[:, :] = color
        patches.append((name, patch))

    return patches

def analyze_color_model(image, model_name):
    """Analyze image in different color models"""
    if model_name == 'RGB':
        return image
    elif model_name == 'HSV':
        return cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    elif model_name == 'LAB':
        return cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    elif model_name == 'GRAY':
        return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        return image

# ================ MAIN PRAKTIKUM ================

# 1. PERBANDINGAN MODEL WARNA
print("\n1. PERBANDINGAN MODEL WARNA DASAR")

# Buat citra dengan warna-warna dasar
color_patches = create_color_patches()

# Tampilkan dalam berbagai model warna
fig, axes = plt.subplots(4, 8, figsize=(20, 10))
models = ['RGB', 'HSV', 'LAB', 'GRAY']
```

```
for row, model in enumerate(models):
    for col, (name, patch) in enumerate(color_patches):
        if model == 'RGB':
            display_img = cv2.cvtColor(patch, cv2.COLOR_BGR2RGB)
            axes[row, col].imshow(display_img)
        elif model == 'GRAY':
            gray_img = cv2.cvtColor(patch, cv2.COLOR_BGR2GRAY)
            axes[row, col].imshow(gray_img, cmap='gray')
        else:
            converted = analyze_color_model(patch, model)
            if model == 'HSV':
                # Convert HSV to RGB for display
                display_img = cv2.cvtColor(converted, cv2.COLOR_HSV2RGB)
            elif model == 'LAB':
                # LAB needs special handling for display
                lab = converted.astype(np.float32)
                lab[:,:,0] = lab[:,:,0] * 255/100  # L from [0,100] to [0,255]
                lab[:,:,1:] = lab[:,:,1:] + 128    # a,b from [-127,127] to [0,255]
                lab = np.clip(lab, 0, 255).astype(np.uint8)
                display_img = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)
            axes[row, col].imshow(display_img)

        if row == 0:
            axes[row, col].set_title(name, fontsize=10)
        axes[row, col].axis('off')

    axes[row, 0].text(-0.5, 0.5, model, transform=axes[row, 0].transAxes,
                      fontsize=12, fontweight='bold', va='center', ha='right')

plt.suptitle('Perbandingan Model Warna untuk Warna Dasar', fontsize=14,
fontweight='bold')
plt.tight_layout()
plt.show()
```

```
# 2. ANALISIS KOMPONEN WARNA
print("\n2. ANALISIS KOMPONEN WARNA PADA CITRA NYATA")

# Load sample image
sample_img = cv2.imread('sample_image.jpg')  # Ganti dengan path citra Anda
if sample_img is None:
    # Create synthetic image if file doesn't exist
    sample_img = np.zeros((300, 400, 3), dtype=np.uint8)
    cv2.rectangle(sample_img, (50, 50), (150, 150), (255, 0, 0), -1)  # Blue
    cv2.circle(sample_img, (250, 100), 50, (0, 255, 0), -1)  # Green
    cv2.ellipse(sample_img, (300, 200), (80, 40), 30, 0, 360, (0, 0, 255), -1)  # Red
    print("Menggunakan citra sintetik (file sample_image.jpg tidak ditemukan)")

# Convert to different color spaces
rgb_img = cv2.cvtColor(sample_img, cv2.COLOR_BGR2RGB)
hsv_img = cv2.cvtColor(sample_img, cv2.COLOR_BGR2HSV)
lab_img = cv2.cvtColor(sample_img, cv2.COLOR_BGR2LAB)
gray_img = cv2.cvtColor(sample_img, cv2.COLOR_BGR2GRAY)

# Split channels for each model
rgb_channels = cv2.split(sample_img)  # B, G, R
hsv_channels = cv2.split(hsv_img)     # H, S, V
lab_channels = cv2.split(lab_img)     # L, a, b

# Display channel analysis
fig, axes = plt.subplots(4, 4, figsize=(16, 12))

# Row 1: RGB Channels
axes[0, 0].imshow(rgb_img)
axes[0, 0].set_title('Original (RGB)')
axes[0, 0].axis('off')

rgb_titles = ['Blue Channel', 'Green Channel', 'Red Channel']
```

```
for i in range(3):
    axes[0, i+1].imshow(rgb_channels[i], cmap='gray')
    axes[0, i+1].set_title(rgb_titles[i])
    axes[0, i+1].axis('off')

# Row 2: HSV Channels
axes[1, 0].imshow(cv2.cvtColor(hsv_img, cv2.COLOR_HSV2RGB))
axes[1, 0].set_title('HSV Color Space')
axes[1, 0].axis('off')

hsv_titles = ['Hue (Jenis Warna)', 'Saturation (Kemurnian)', 'Value (Kecerahan)']
for i in range(3):
    axes[1, i+1].imshow(hsv_channels[i], cmap='gray')
    axes[1, i+1].set_title(hsv_titles[i])
    axes[1, i+1].axis('off')

# Row 3: LAB Channels
# Convert LAB for display
lab_display = lab_img.astype(np.float32)
lab_display[:,:,0] = lab_display[:,:,0] * 255/100
lab_display[:,:,1:] = lab_display[:,:,1:] + 128
lab_display = np.clip(lab_display, 0, 255).astype(np.uint8)
lab_display_rgb = cv2.cvtColor(lab_display, cv2.COLOR_LAB2RGB)

axes[2, 0].imshow(lab_display_rgb)
axes[2, 0].set_title('LAB Color Space')
axes[2, 0].axis('off')

lab_titles = ['Lightness', 'Green-Red (a)', 'Blue-Yellow (b)']
for i in range(3):
    if i == 0:
        # L channel needs scaling
        l_channel = lab_channels[i].astype(np.float32) * 255/100
        l_channel = np.clip(l_channel, 0, 255).astype(np.uint8)
```

```
        axes[2, i+1].imshow(l_channel, cmap='gray')
    else:
        # a and b channels need offset
        ab_channel = lab_channels[i].astype(np.float32) + 128
        ab_channel = np.clip(ab_channel, 0, 255).astype(np.uint8)
        axes[2, i+1].imshow(ab_channel, cmap='gray')
    axes[2, i+1].set_title(lab_titles[i])
    axes[2, i+1].axis('off')

# Row 4: Grayscale
axes[3, 0].imshow(gray_img, cmap='gray')
axes[3, 0].set_title('Grayscale')
axes[3, 0].axis('off')

# Show histogram of grayscale
axes[3, 1].hist(gray_img.ravel(), 256, [0, 256], color='gray')
axes[3, 1].set_title('Grayscale Histogram')
axes[3, 1].set_xlabel('Intensity')
axes[3, 1].set_ylabel('Frequency')
axes[3, 1].grid(True, alpha=0.3)

# Show color histogram
colors = ('b', 'g', 'r')
for i, color in enumerate(colors):
    hist = cv2.calcHist([sample_img], [i], None, [256], [0, 256])
    axes[3, 2].plot(hist, color=color, alpha=0.7)
axes[3, 2].set_title('RGB Channel Histograms')
axes[3, 2].set_xlabel('Intensity')
axes[3, 2].set_ylabel('Frequency')
axes[3, 2].legend(['Blue', 'Green', 'Red'])
axes[3, 2].grid(True, alpha=0.3)

# Show HSV histogram
hsv_hist = cv2.calcHist([hsv_img], [0], None, [180], [0, 180])
```

```
axes[3, 3].plot(hsv_hist, color='orange')
axes[3, 3].set_title('Hue Histogram')
axes[3, 3].set_xlabel('Hue (0-180°)')
axes[3, 3].set_ylabel('Frequency')
axes[3, 3].grid(True, alpha=0.3)

plt.suptitle('Analisis Komponen Warna pada Berbagai Model', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

# 3. DEMONSTRASI TEOREMA SAMPLING DAN ALIASING
print("\n3. DEMONSTRASI TEOREMA SAMPLING DAN ALIASING")

def demonstrate_sampling_aliasing():
    """Demonstrate sampling theorem and aliasing effect"""
    # Create high frequency signal
    t = np.linspace(0, 1, 1000)  # High resolution time
    f_high = 50  # High frequency component
    f_low = 5    # Low frequency component

    signal_high = np.sin(2 * np.pi * f_high * t)
    signal_low = np.sin(2 * np.pi * f_low * t)
    signal_combined = signal_high + 0.5 * signal_low

    # Different sampling rates
    sampling_rates = [20, 50, 100, 200]  # Hz
    nyquist_rate = 2 * f_high  # Should be 100 Hz

    fig, axes = plt.subplots(2, 2, figsize=(12, 8))
    axes = axes.ravel()

    for idx, fs in enumerate(sampling_rates):
        # Sampling
```

```
    t_sampled = np.arange(0, 1, 1/fs)
    indices = (t_sampled * 1000).astype(int)
    signal_sampled = signal_combined[indices]

    # Reconstruction (zero-order hold)
    t_recon = np.linspace(0, 1, 1000)
    signal_recon = np.zeros_like(t_recon)
    for i in range(len(t_sampled)-1):
        mask = (t_recon >= t_sampled[i]) & (t_recon < t_sampled[i+1])
        signal_recon[mask] = signal_sampled[i]

    # Plot
    axes[idx].plot(t, signal_combined, 'b-', alpha=0.5, label='Original')
    axes[idx].stem(t_sampled, signal_sampled, linefmt='r-', markerfmt='ro',
                   basefmt=' ', label=f'Sampled (fs={fs}Hz)')
    axes[idx].plot(t_recon, signal_recon, 'g--', alpha=0.7, label='Reconstructed')

    # Check for aliasing
    if fs < nyquist_rate:
        axes[idx].set_title(f'ALIASING: fs={fs}Hz < Nyquist={nyquist_rate}Hz',
                            color='red', fontweight='bold')
    else:
        axes[idx].set_title(f'No Aliasing: fs={fs}Hz ≥ Nyquist',
                            color='green', fontweight='bold')

    axes[idx].set_xlabel('Time (s)')
    axes[idx].set_ylabel('Amplitude')
    axes[idx].legend(loc='upper right')
    axes[idx].grid(True, alpha=0.3)

plt.suptitle('Demonstrasi Teorema Sampling Nyquist dan Efek Aliasing',
             fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
    return sampling_rates, nyquist_rate

sampling_rates, nyquist_rate = demonstrate_sampling_aliasing()
print(f"Frekuensi sinyal tinggi: 50 Hz")
print(f"Nyquist rate minimum: {nyquist_rate} Hz")
print(f"Sampling rates yang diuji: {sampling_rates} Hz")

# 4. DEMONSTRASI KUANTISASI
print("\n4. DEMONSTRASI KUANTISASI UNIFORM DAN NON-UNIFORM")

def demonstrate_quantization():
    """Demonstrate uniform and non-uniform quantization"""
    # Create test image
    test_img = gray_img if len(gray_img.shape) == 2 else cv2.cvtColor(sample_img,
cv2.COLOR_BGR2GRAY)

    # Uniform quantization
    quantization_levels = [256, 64, 16, 4, 2]  # 8-bit, 6-bit, 4-bit, 2-bit, 1-bit

    fig, axes = plt.subplots(2, 5, figsize=(15, 6))

    # Uniform quantization
    for idx, levels in enumerate(quantization_levels):
        # Calculate step size
        step = 256 // levels
        quantized_uniform = (test_img // step) * step

        # Display
        axes[0, idx].imshow(quantized_uniform, cmap='gray', vmin=0, vmax=255)
        axes[0, idx].set_title(f'Uniform\n{levels} levels\n({int(np.log2(levels))}-bit)')
        axes[0, idx].axis('off')

        # Calculate MSE
```

```
        mse = np.mean((test_img.astype(float) - quantized_uniform.astype(float))**2)
        psnr = 10 * np.log10(255**2 / mse) if mse > 0 else float('inf')
        axes[0, idx].text(0.5, -0.1, f'MSE: {mse:.1f}\nPSNR: {psnr:.1f}dB',
                          transform=axes[0, idx].transAxes, ha='center', fontsize=8)

    # Non-uniform quantization (using histogram equalization principle)
    for idx, levels in enumerate(quantization_levels):
        # Calculate histogram
        hist, bins = np.histogram(test_img.flatten(), 256, [0, 256])

        # Calculate cumulative distribution
        cdf = hist.cumsum()
        cdf_normalized = cdf / cdf.max()

        # Create mapping function for non-uniform quantization
        mapping = np.zeros(256, dtype=np.uint8)
        for i in range(levels):
            lower = i * 256 // levels
            upper = (i + 1) * 256 // levels
            mapping[lower:upper] = i * 255 // (levels - 1)

        # Apply non-uniform quantization
        quantized_nonuniform = mapping[test_img]

        # Display
        axes[1, idx].imshow(quantized_nonuniform, cmap='gray', vmin=0, vmax=255)
        axes[1, idx].set_title(f'Non-Uniform\n{levels} levels')
        axes[1, idx].axis('off')

        # Calculate MSE
        mse = np.mean((test_img.astype(float) - quantized_nonuniform.astype(float))**2)
```

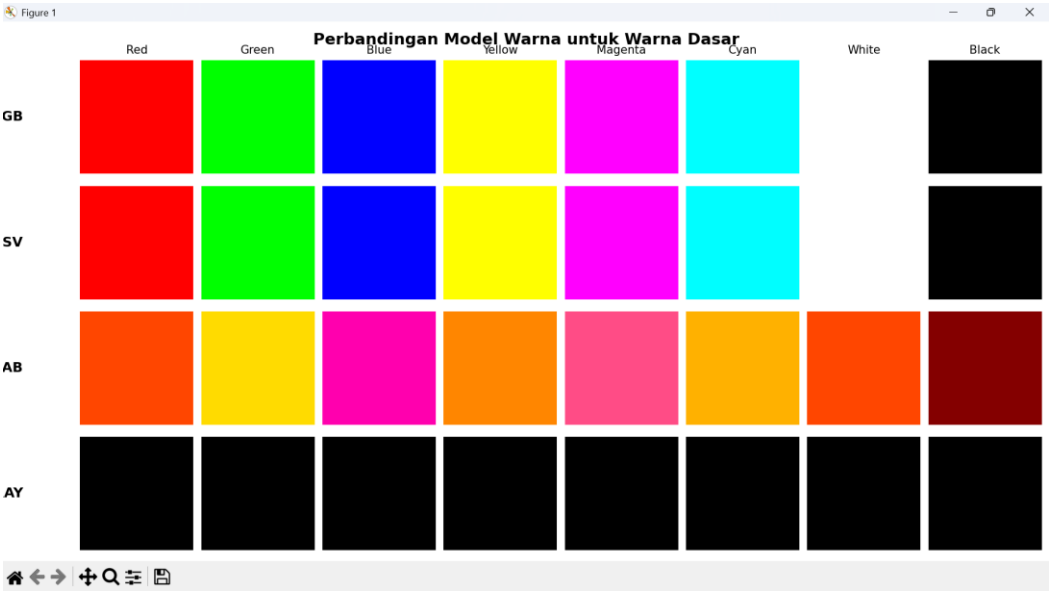
```
        psnr = 10 * np.log10(255**2 / mse) if mse > 0 else float('inf')
        axes[1, idx].text(0.5, -0.1, f'MSE: {mse:.1f}\nPSNR: {psnr:.1f}dB',
                          transform=axes[1, idx].transAxes, ha='center', fontsize=8)

    plt.suptitle('Perbandingan Kuantisasi Uniform vs Non-Uniform',
                 fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    return quantization_levels

quant_levels = demonstrate_quantization()

# 5. KONVERSI RUANG WARNA DAN APLIKASI
print("\n5. KONVERSI RUANG WARNA DAN APLIKASI PRAKTIS")

def demonstrate_color_conversion_applications():
    """Demonstrate practical applications of color space conversions"""
    # Load or create sample image
    if sample_img is None:
        # Create synthetic image for demonstration
        demo_img = np.zeros((300, 400, 3), dtype=np.uint8)
        # Add various colors and patterns
        cv2.rectangle(demo_img, (50, 50), (150, 150), (100, 150, 200), -1)  # Sky blue
        cv2.rectangle(demo_img, (200, 50), (300, 150), (200, 100, 50), -1)   # Brown
        cv2.circle(demo_img, (100, 220), 40, (50, 200, 100), -1)  # Green
        cv2.circle(demo_img, (250, 220), 40, (200, 50, 150), -1)  # Purple
    else:
        demo_img = sample_img

    # Application 1: Skin Detection using HSV
    hsv_img = cv2.cvtColor(demo_img, cv2.COLOR_BGR2HSV)

    # Define skin color range in HSV
```

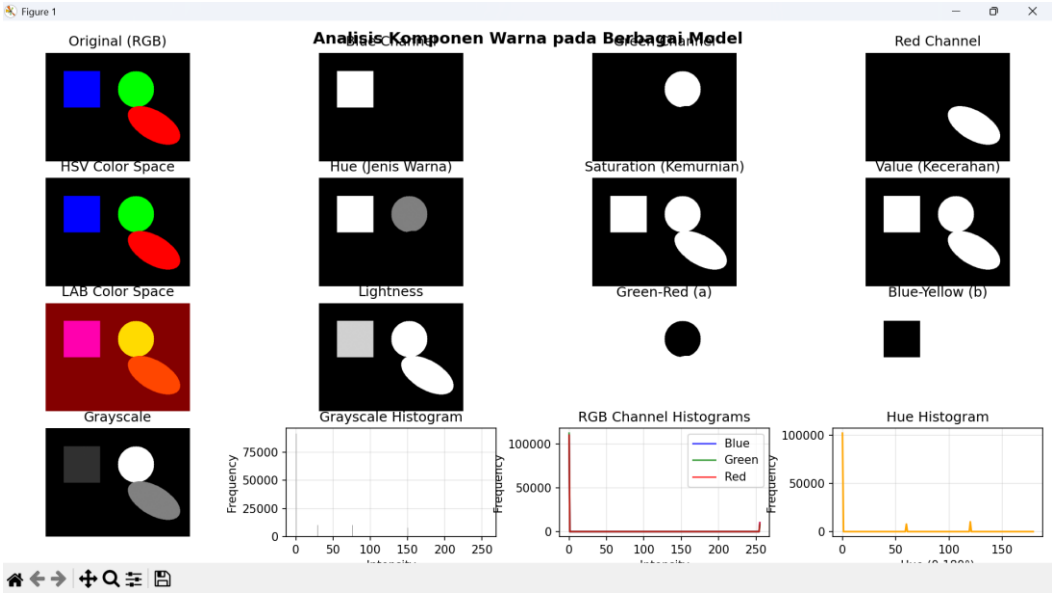
```
lower_skin = np.array([0, 20, 70], dtype=np.uint8)
upper_skin = np.array([20, 255, 255], dtype=np.uint8)

# Create skin mask
skin_mask = cv2.inRange(hsv_img, lower_skin, upper_skin)

# Apply mask to original image
skin_detected = cv2.bitwise_and(demo_img, demo_img, mask=skin_mask)

# Application 2: Shadow removal using LAB
lab_img = cv2.cvtColor(demo_img, cv2.COLOR_BGR2LAB)
L, a, b = cv2.split(lab_img)

# Apply CLAHE to L channel (improves shadow details)
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
L_enhanced = clahe.apply(L)

# Merge back
lab_enhanced = cv2.merge([L_enhanced, a, b])
rgb_enhanced = cv2.cvtColor(lab_enhanced, cv2.COLOR_LAB2BGR)

# Application 3: Color-based segmentation using RGB
# Define color ranges for segmentation
colors_to_segment = [
    ('Blue', [100, 0, 0], [255, 100, 100]),
    ('Green', [0, 100, 0], [100, 255, 100]),
    ('Red', [0, 0, 100], [100, 100, 255])
]

# Create segmentation masks
segmentation_results = []
for color_name, lower, upper in colors_to_segment:
    lower = np.array(lower, dtype=np.uint8)
    upper = np.array(upper, dtype=np.uint8)
```

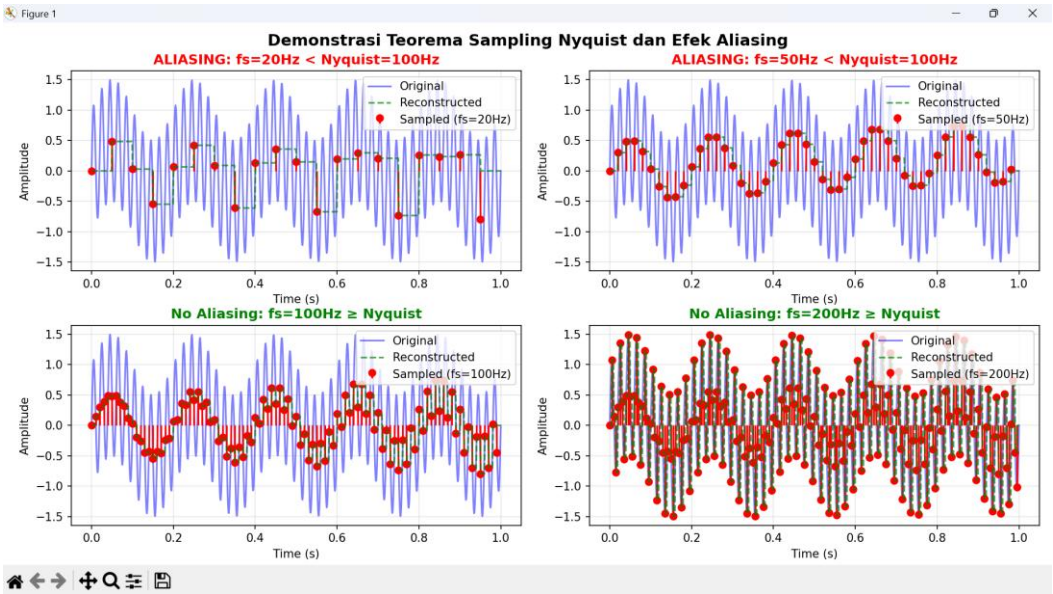
```
    mask = cv2.inRange(demo_img, lower, upper)
    segmented = cv2.bitwise_and(demo_img, demo_img, mask=mask)
    segmentation_results.append((color_name, segmented))

# Display results
fig, axes = plt.subplots(3, 3, figsize=(15, 12))

# Row 1: Original and conversions
axes[0, 0].imshow(cv2.cvtColor(demo_img, cv2.COLOR_BGR2RGB))
axes[0, 0].set_title('Original Image (RGB)')
axes[0, 0].axis('off')

axes[0, 1].imshow(cv2.cvtColor(hsv_img, cv2.COLOR_HSV2RGB))
axes[0, 1].set_title('HSV Color Space')
axes[0, 1].axis('off')

lab_display = lab_img.astype(np.float32)
lab_display[:,:,0] = lab_display[:,:,0] * 255/100
lab_display[:,:,1:] = lab_display[:,:,1:] + 128
lab_display = np.clip(lab_display, 0, 255).astype(np.uint8)
axes[0, 2].imshow(cv2.cvtColor(lab_display, cv2.COLOR_LAB2RGB))
axes[0, 2].set_title('LAB Color Space')
axes[0, 2].axis('off')

# Row 2: Applications
axes[1, 0].imshow(skin_mask, cmap='gray')
axes[1, 0].set_title('Skin Detection Mask (HSV)')
axes[1, 0].axis('off')

axes[1, 1].imshow(cv2.cvtColor(skin_detected, cv2.COLOR_BGR2RGB))
axes[1, 1].set_title('Skin Detection Result')
axes[1, 1].axis('off')

axes[1, 2].imshow(cv2.cvtColor(rgb_enhanced, cv2.COLOR_BGR2RGB))
```

```
    axes[1, 2].set_title('Shadow Removal (LAB + CLAHE)')
    axes[1, 2].axis('off')

    # Row 3: Color segmentation
    for idx, (color_name, segmented) in enumerate(segmentation_results):
        axes[2, idx].imshow(cv2.cvtColor(segmented, cv2.COLOR_BGR2RGB))
        axes[2, idx].set_title(f'{color_name} Segmentation')
        axes[2, idx].axis('off')

    # Hide unused subplots
    for i in range(3, 9):
        axes.flat[i].axis('off')

    plt.suptitle('Aplikasi Praktis Konversi Ruang Warna', fontsize=14,
fontweight='bold')
    plt.tight_layout()
    plt.show()

    return skin_mask, rgb_enhanced

skin_mask_result, enhanced_result =
demonstrate_color_conversion_applications()

print("\n=== PRAKTIKUM SELESAI ===")
print("\nRingkasan yang dipelajari:")
print("1. Perbandingan model warna RGB, HSV, LAB, Grayscale")
print("2. Teorema Sampling Nyquist dan efek aliasing")
print("3. Teknik kuantisasi uniform dan non-uniform")
print("4. Konversi antar ruang warna dan aplikasi praktis")
```

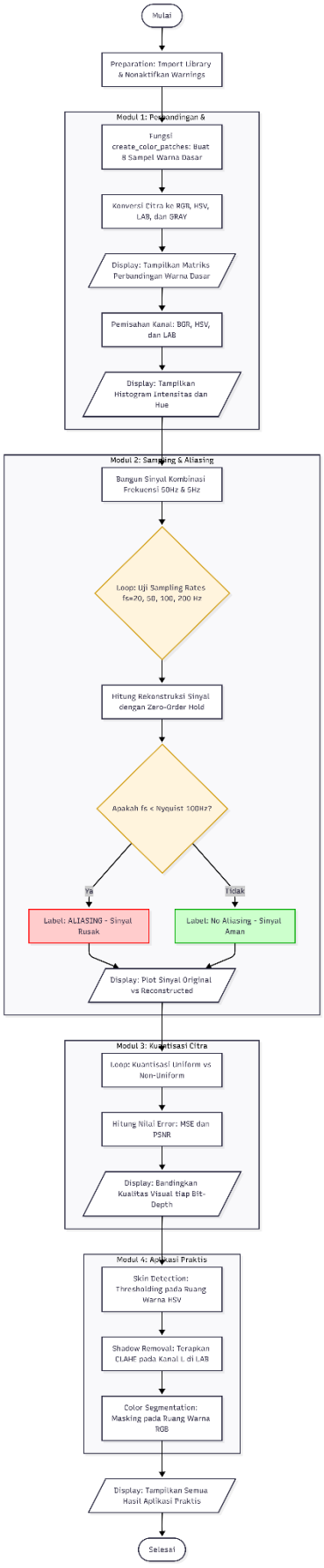
b. Screenshot Output







c. Flowchart


Mulai
Preparation: Import Library & Nonaktifkan Warnings
Modul 1: Perbandingan &
Fungsi create_color_patches: Buat 8 Sampel Warna Dasar
Konversi Citra ke RGB, HSV, LAB, dan GRAY
Display: Tampilkan Matriks Perbandingan Warna Dasar
Pemisahan Kanal: BGR, HSV, dan LAB
Display: Tampilkan Histogram Intensitas dan Hue
Modul 2: Sampling & Aliasing
Bangun Sinyal Kombinasi Frekuensi 50Hz & 5Hz
Loop: Uji Sampling Rates fs=20, 50, 100, 200 Hz
Hitung Rekonstruksi Sinyal dengan Zero-Order Hold
Apakah fs < Nyquist 100Hz?
Ya
Tidak
Label: ALIASING - Sinyal Rusak
Label: No Aliasing - Sinyal Aman
Display: Plot Sinyal Original vs Reconstructed
Modul 3: Kuantisasi Citra
Loop: Kuantisasi Uniform vs Non-Uniform
Hitung Nilai Error: MSE dan PSNR
Display: Bandingkan Kualitas Visual tiap Bit-Depth
Modul 4: Aplikasi Praktis
Skin Detection: Thresholding pada Ruang Warna HSV
Shadow Removal: Terapkan CLAHE pada Kanal L di LAB
Color Segmentation: Masking pada Ruang Warna RGB
Display: Tampilkan Semua Hasil Aplikasi Praktis
Selesai