import os

from uuid import uuid4

from fastapi import FastAPI, File, UploadFile, Request

from fastapi.responses import HTMLResponse

from fastapi.templating import Jinja2Templates

from fastapi.staticfiles import StaticFiles

import numpy as np

import cv2

app = FastAPI()

app.mount("/static", StaticFiles(*directory*="static"), *name*="static")

templates = Jinja2Templates(*directory*="templates")

if not os.path.exists("static/uploads"):

    os.makedirs("static/uploads")

@app.get("/", *response\_class*=HTMLResponse)

async def home(*request*: Request):

    return templates.TemplateResponse("home.html", {"request": request})

@app.post("/upload/", *response\_class*=HTMLResponse)

async def upload\_image(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    file\_extension = file.filename.split(".")[-1]

    filename = f"{uuid4()}.{file\_extension}"

    file\_path = os.path.join("static", "uploads", filename)

    with open(file\_path, "wb") as f:

        f.write(image\_data)

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)

    b, g, r = cv2.split(img)

    rgb\_array = {"R": r.tolist(), "G": g.tolist(), "B": b.tolist()}

    return templates.TemplateResponse("display.html", {

        "request": request,

        "image\_path": f"/static/uploads/{filename}",

        "rgb\_array": rgb\_array

    })

import os

from uuid import uuid4

from fastapi import FastAPI, File, UploadFile, Request, Form

from fastapi.responses import HTMLResponse

from fastapi.templating import Jinja2Templates

from fastapi.staticfiles import StaticFiles

from skimage.exposure import match\_histograms  # pastikan paket scikit-image sudah terinstal

import numpy as np

import cv2

import matplotlib.pyplot as plt

app = FastAPI()

app.mount("/static", StaticFiles(*directory*="static"), *name*="static")

templates = Jinja2Templates(*directory*="templates")

if not os.path.exists("static/uploads"):

    os.makedirs("static/uploads")

if not os.path.exists("static/histograms"):

    os.makedirs("static/histograms")

@app.get("/", *response\_class*=HTMLResponse)

async def home(*request*: Request):

    return templates.TemplateResponse("home.html", {"request": request})

@app.post("/upload/", *response\_class*=HTMLResponse)

async def upload\_image(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)

    file\_path = save\_image(img, "uploaded")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": file\_path,

        "modified\_image\_path": file\_path

    })

@app.post("/operation/", *response\_class*=HTMLResponse)

async def perform\_operation(

*request*: Request,

*file*: UploadFile = File(...),

*operation*: *str* = Form(...),

*value*: *int* = Form(...)

):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)

    original\_path = save\_image(img, "original")

    if operation == "add":

        result\_img = cv2.add(img, np.full(img.shape, value, *dtype*=np.uint8))

    elif operation == "subtract":

        result\_img = cv2.subtract(img, np.full(img.shape, value, *dtype*=np.uint8))

    elif operation == "max":

        result\_img = np.maximum(img, np.full(img.shape, value, *dtype*=np.uint8))

    elif operation == "min":

        result\_img = np.minimum(img, np.full(img.shape, value, *dtype*=np.uint8))

    elif operation == "inverse":

        result\_img = cv2.bitwise\_not(img)

    modified\_path = save\_image(result\_img, "modified")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": original\_path,

        "modified\_image\_path": modified\_path

    })

@app.post("/logic\_operation/", *response\_class*=HTMLResponse)

async def perform\_logic\_operation(

*request*: Request,

*file1*: UploadFile = File(...),

*file2*: UploadFile = File(None),

*operation*: *str* = Form(...)

):

    image\_data1 = await file1.read()

    np\_array1 = np.frombuffer(image\_data1, np.uint8)

    img1 = cv2.imdecode(np\_array1, cv2.IMREAD\_COLOR)

    original\_path = save\_image(img1, "original")

    if operation == "not":

        result\_img = cv2.bitwise\_not(img1)

    else:

        if file2 is None:

            return HTMLResponse("Operasi AND dan XOR memerlukan dua gambar.", *status\_code*=400)

        image\_data2 = await file2.read()

        np\_array2 = np.frombuffer(image\_data2, np.uint8)

        img2 = cv2.imdecode(np\_array2, cv2.IMREAD\_COLOR)

        if operation == "and":

            result\_img = cv2.bitwise\_and(img1, img2)

        elif operation == "xor":

            result\_img = cv2.bitwise\_xor(img1, img2)

    modified\_path = save\_image(result\_img, "modified")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": original\_path,

        "modified\_image\_path": modified\_path

    })

@app.get("/grayscale/", *response\_class*=HTMLResponse)

async def grayscale\_form(*request*: Request):

    # Menampilkan form untuk upload gambar ke grayscale

    return templates.TemplateResponse("grayscale.html", {"request": request})

@app.post("/grayscale/", *response\_class*=HTMLResponse)

async def convert\_grayscale(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)

    gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    original\_path = save\_image(img, "original")

    modified\_path = save\_image(gray\_img, "grayscale")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": original\_path,

        "modified\_image\_path": modified\_path

    })

@app.get("/histogram/", *response\_class*=HTMLResponse)

async def histogram\_form(*request*: Request):

    # Menampilkan halaman untuk upload gambar untuk histogram

    return templates.TemplateResponse("histogram.html", {"request": request})

@app.post("/histogram/", *response\_class*=HTMLResponse)

async def generate\_histogram(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)

    # Pastikan gambar berhasil diimpor

    if img is None:

        return HTMLResponse("Tidak dapat membaca gambar yang diunggah", *status\_code*=400)

    # Buat histogram grayscale dan berwarna

    gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    grayscale\_histogram\_path = save\_histogram(gray\_img, "grayscale")

    color\_histogram\_path = save\_color\_histogram(img)

    return templates.TemplateResponse("histogram.html", {

        "request": request,

        "grayscale\_histogram\_path": grayscale\_histogram\_path,

        "color\_histogram\_path": color\_histogram\_path

    })

@app.get("/equalize/", *response\_class*=HTMLResponse)

async def equalize\_form(*request*: Request):

    # Menampilkan halaman untuk upload gambar untuk equalisasi histogram

    return templates.TemplateResponse("equalize.html", {"request": request})

@app.post("/equalize/", *response\_class*=HTMLResponse)

async def equalize\_histogram(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_GRAYSCALE)

    equalized\_img = cv2.equalizeHist(img)

    original\_path = save\_image(img, "original")

    modified\_path = save\_image(equalized\_img, "equalized")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": original\_path,

        "modified\_image\_path": modified\_path

    })

@app.get("/specify/", *response\_class*=HTMLResponse)

async def specify\_form(*request*: Request):

    # Menampilkan halaman untuk upload gambar dan referensi untuk spesifikasi histogram

    return templates.TemplateResponse("specify.html", {"request": request})

@app.post("/specify/", *response\_class*=HTMLResponse)

async def specify\_histogram(*request*: Request, *file*: UploadFile = File(...), *ref\_file*: UploadFile = File(...)):

    # Baca gambar yang diunggah dan gambar referensi

    image\_data = await file.read()

    ref\_image\_data = await ref\_file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    ref\_np\_array = np.frombuffer(ref\_image\_data, np.uint8)

        #jika ingin grayscale

    #img = cv2.imdecode(np\_array, cv2.IMREAD\_GRAYSCALE)

    #ref\_img = cv2.imdecode(ref\_np\_array, cv2.IMREAD\_GRAYSCALE)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)  # Membaca gambar dalam format BGR

    ref\_img = cv2.imdecode(ref\_np\_array, cv2.IMREAD\_COLOR)  # Membaca gambar referensi dalam format BGR

    if img is None or ref\_img is None:

        return HTMLResponse("Gambar utama atau gambar referensi tidak dapat dibaca.", *status\_code*=400)

    # Spesifikasi histogram menggunakan match\_histograms dari skimage #grayscale

    #specified\_img = match\_histograms(img, ref\_img, multichannel=False)

            # Spesifikasi histogram menggunakan match\_histograms dari skimage untuk gambar berwarna

    specified\_img = match\_histograms(img, ref\_img, *channel\_axis*=-1)

    # Konversi kembali ke format uint8 jika diperlukan

    specified\_img = np.clip(specified\_img, 0, 255).astype('uint8')

    original\_path = save\_image(img, "original")

    modified\_path = save\_image(specified\_img, "specified")

    return templates.TemplateResponse("result.html", {

        "request": request,

        "original\_image\_path": original\_path,

        "modified\_image\_path": modified\_path

    })

@app.post("/statistics/", *response\_class*=HTMLResponse)

async def calculate\_statistics(*request*: Request, *file*: UploadFile = File(...)):

    image\_data = await file.read()

    np\_array = np.frombuffer(image\_data, np.uint8)

    img = cv2.imdecode(np\_array, cv2.IMREAD\_GRAYSCALE)

    mean\_intensity = np.mean(img)

    std\_deviation = np.std(img)

    image\_path = save\_image(img, "statistics")

    return templates.TemplateResponse("statistics.html", {

        "request": request,

        "mean\_intensity": mean\_intensity,

        "std\_deviation": std\_deviation,

        "image\_path": image\_path

    })

def save\_image(*image*, *prefix*):

    filename = f"{prefix}\_{uuid4()}.png"

    path = os.path.join("static/uploads", filename)

    cv2.imwrite(path, image)

    return f"/static/uploads/{filename}"

def save\_histogram(*image*, *prefix*):

    histogram\_path = f"static/histograms/{prefix}\_{uuid4()}.png"

    plt.figure()

    plt.hist(image.ravel(), 256, [0, 256])

    plt.savefig(histogram\_path)

    plt.close()

    return f"/{histogram\_path}"

def save\_color\_histogram(*image*):

    color\_histogram\_path = f"static/histograms/color\_{uuid4()}.png"

    plt.figure()

    for i, color in enumerate(['b', 'g', 'r']):

        hist = cv2.calcHist([image], [i], None, [256], [0, 256])

        plt.plot(hist, *color*=color)

    plt.savefig(color\_histogram\_path)

    plt.close()

    return f"/{color\_histogram\_path}"

import os

import cv2

import numpy as np

from fastapi import FastAPI, File, UploadFile, Form

from fastapi.responses import HTMLResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

from starlette.requests import Request

import shutil

app = FastAPI()

app.mount("/static", StaticFiles(*directory*="static"), *name*="static")

templates = Jinja2Templates(*directory*="templates")

UPLOAD\_DIR = "static/uploads"

os.makedirs(UPLOAD\_DIR, *exist\_ok*=True)

@app.get("/", *response\_class*=HTMLResponse)

async def home(*request*: Request):

    return templates.TemplateResponse("home.html", {"request": request})

@app.post("/upload/")

async def upload\_image(*file*: UploadFile = File(...)):

    file\_path = os.path.join(UPLOAD\_DIR, file.filename)

    with open(file\_path, "wb") as buffer:

        shutil.copyfileobj(file.file, buffer)

    return {"filename": file.filename, "path": f"/static/uploads/{file.filename}"}

def read\_image(*file\_path*):

    return cv2.imread(file\_path)

@app.post("/convolution/")

async def apply\_convolution(*filename*: *str* = Form(...), *kernel\_type*: *str* = Form(...)):

    img = read\_image(os.path.join(UPLOAD\_DIR, filename))

    if img is None:

        return {"error": "Image not found"}

    kernels = {

        "average": np.ones((3, 3), np.float32) / 9,

        "sharpen": np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]),

        "edge": np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])

    }

    kernel = kernels.get(kernel\_type)

    if kernel is None:

        return {"error": "Invalid kernel type"}

    processed\_img = cv2.filter2D(img, -1, kernel)

    output\_filename = f"conv\_{filename}"

    output\_path = os.path.join(UPLOAD\_DIR, output\_filename)

    cv2.imwrite(output\_path, processed\_img)

    return {"processed\_image": f"/static/uploads/{output\_filename}"}

@app.post("/padding/")

async def apply\_padding(*filename*: *str* = Form(...), *padding\_size*: *int* = Form(...)):

    img = read\_image(os.path.join(UPLOAD\_DIR, filename))

    if img is None:

        return {"error": "Image not found"}

    padded\_img = cv2.copyMakeBorder(img, padding\_size, padding\_size, padding\_size, padding\_size, cv2.BORDER\_CONSTANT, *value*=[0, 0, 0])

    output\_filename = f"padded\_{filename}"

    output\_path = os.path.join(UPLOAD\_DIR, output\_filename)

    cv2.imwrite(output\_path, padded\_img)

    return {"processed\_image": f"/static/uploads/{output\_filename}"}

@app.post("/filter/")

async def apply\_filter(*filename*: *str* = Form(...), *filter\_type*: *str* = Form(...)):

    img = read\_image(os.path.join(UPLOAD\_DIR, filename))

    if img is None:

        return {"error": "Image not found"}

    if filter\_type == "low":

        filtered\_img = cv2.GaussianBlur(img, (5, 5), 0)

    elif filter\_type == "high":

        kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])

        filtered\_img = cv2.filter2D(img, -1, kernel)

    elif filter\_type == "band":

        dft = cv2.dft(np.float32(cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)), *flags*=cv2.DFT\_COMPLEX\_OUTPUT)

        dft\_shift = np.fft.fftshift(dft)

        rows, cols = img.shape[:2]

        crow, ccol = rows // 2, cols // 2

        mask = np.zeros((rows, cols, 2), np.uint8)

        mask[crow-30:crow+30, ccol-30:ccol+30] = 1

        fshift = dft\_shift \* mask

        f\_ishift = np.fft.ifftshift(fshift)

        img\_back = cv2.idft(f\_ishift)

        filtered\_img = cv2.magnitude(img\_back[:, :, 0], img\_back[:, :, 1])

    else:

        return {"error": "Invalid filter type"}

    output\_filename = f"filter\_{filename}"

    output\_path = os.path.join(UPLOAD\_DIR, output\_filename)

    cv2.imwrite(output\_path, filtered\_img.astype(np.uint8))

    return {"processed\_image": f"/static/uploads/{output\_filename}"}

@app.post("/fourier/")

async def apply\_fourier(*filename*: *str* = Form(...)):

    img = read\_image(os.path.join(UPLOAD\_DIR, filename))

    if img is None:

        return {"error": "Image not found"}

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    f = np.fft.fft2(gray)

    fshift = np.fft.fftshift(f)

    magnitude\_spectrum = np.log(np.abs(fshift) + 1)

    magnitude\_spectrum = cv2.normalize(magnitude\_spectrum, None, 0, 255, cv2.NORM\_MINMAX)

    output\_filename = f"fourier\_{filename}"

    output\_path = os.path.join(UPLOAD\_DIR, output\_filename)

    cv2.imwrite(output\_path, magnitude\_spectrum.astype(np.uint8))

    return {"processed\_image": f"/static/uploads/{output\_filename}"}

@app.post("/reduce-noise/")

async def reduce\_periodic\_noise(*filename*: *str* = Form(...)):

    img = read\_image(os.path.join(UPLOAD\_DIR, filename))

    if img is None:

        return {"error": "Image not found"}

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    f = np.fft.fft2(gray)

    fshift = np.fft.fftshift(f)

    rows, cols = gray.shape

    crow, ccol = rows // 2 , cols // 2

    mask = np.ones((rows, cols), np.uint8)

    mask[crow-30:crow+30, ccol-30:ccol+30] = 0

    fshift = fshift \* mask

    f\_ishift = np.fft.ifftshift(fshift)

    img\_back = np.fft.ifft2(f\_ishift)

    img\_back = np.abs(img\_back)

    output\_filename = f"denoise\_{filename}"

    output\_path = os.path.join(UPLOAD\_DIR, output\_filename)

    cv2.imwrite(output\_path, img\_back.astype(np.uint8))

    return {"processed\_image": f"/static/uploads/{output\_filename}"}

import streamlit as st

import cv2

import os

import numpy as np

import time

import random

from PIL import Image, ImageEnhance

# Fungsi untuk mendeteksi wajah

def detect\_faces(*image*):

    face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    faces = face\_cascade.detectMultiScale(

        gray,

*scaleFactor*=1.1,

*minNeighbors*=5,

*minSize*=(30, 30)

    )

    return faces

# Fungsi untuk menambahkan noise Gaussian

def add\_gaussian\_noise(*image*, *mean*=0, *sigma*=25):

    # Convert to float for processing

    img\_float = image.astype(np.float32) / 255.0

    # Generate Gaussian noise

    noise = np.random.normal(mean, sigma/255.0, img\_float.shape)

    # Add noise to image

    noisy\_img = img\_float + noise

    # Clip values to be in valid range

    noisy\_img = np.clip(noisy\_img, 0, 1)

    # Convert back to original data type

    noisy\_img = (noisy\_img \* 255).astype(np.uint8)

    return noisy\_img

# Fungsi untuk menambahkan salt and pepper noise dengan efek yang lebih alami

def add\_salt\_pepper\_noise(*image*, *density*=0.02):

    output = np.copy(image)

    # Salt

    num\_salt = np.ceil(density \* image.size \* 0.5)

    coords = [np.random.randint(0, i - 1, *int*(num\_salt)) for i in image.shape]

    output[coords[0], coords[1], :] = 255

    # Pepper

    num\_pepper = np.ceil(density \* image.size \* 0.5)

    coords = [np.random.randint(0, i - 1, *int*(num\_pepper)) for i in image.shape]

    output[coords[0], coords[1], :] = 0

    return output

# Fungsi untuk menambahkan kombinasi noise

def add\_combined\_noise(*image*, *noise\_type*='gaussian', *density*=0.02, *mean*=0, *sigma*=25):

    if noise\_type == 'gaussian':

        return add\_gaussian\_noise(image, mean, sigma)

    elif noise\_type == 'salt\_pepper':

        return add\_salt\_pepper\_noise(image, density)

    elif noise\_type == 'combined':

        # Tambahkan Gaussian noise dulu

        noisy\_img = add\_gaussian\_noise(image, mean, sigma)

        # Lalu tambahkan salt & pepper

        return add\_salt\_pepper\_noise(noisy\_img, density/2)  # Kurangi density karena sudah ada gaussian

    else:

        return image  # Return original if no valid type selected

# Fungsi untuk menghilangkan noise dengan beberapa pilihan filter

def remove\_noise(*image*, *method*='median', *kernel\_size*=3):

    if method == 'median':

        return cv2.medianBlur(image, kernel\_size)

    elif method == 'gaussian':

        return cv2.GaussianBlur(image, (kernel\_size, kernel\_size), 0)

    elif method == 'bilateral':

        # Bilateral filter preserves edges better

        return cv2.bilateralFilter(image, kernel\_size, 75, 75)

    else:

        return image  # Return original if no valid method selected

# Fungsi untuk menajamkan gambar

def sharpen\_image(*image*, *method*='normal'):

    if method == 'normal':

        kernel = np.array([[-1, -1, -1],

                           [-1,  9, -1],

                           [-1, -1, -1]])

        return cv2.filter2D(image, -1, kernel)

    elif method == 'edge\_enhance':

        kernel = np.array([[-1, -1, -1, -1, -1],

                           [-1,  2,  2,  2, -1],

                           [-1,  2,  8,  2, -1],

                           [-1,  2,  2,  2, -1],

                           [-1, -1, -1, -1, -1]]) / 8.0

        return cv2.filter2D(image, -1, kernel)

    elif method == 'unsharp\_mask':

        # Unsharp masking

        gaussian = cv2.GaussianBlur(image, (5, 5), 0)

        return cv2.addWeighted(image, 1.5, gaussian, -0.5, 0)

    else:

        return image

# Fungsi untuk menampilkan gambar sebelum dan sesudah

def display\_before\_after(*original*, *processed*, *title*):

    col1, col2 = st.columns(2)

    with col1:

        st.image(original, *caption*="Gambar Asli", *use\_column\_width*=True)

    with col2:

        st.image(processed, *caption*=title, *use\_column\_width*=True)

# Judul Aplikasi

st.title("Sistem Pengenalan Wajah dengan Pengolahan Citra")

st.write("Aplikasi ini memungkinkan penambahan dataset wajah dan menerapkan pengolahan citra pada dataset.")

# Buat tab untuk dua fungsi utama

tab1, tab2 = st.tabs(["Tambah Wajah Baru", "Lihat & Proses Dataset"])

with tab1:

    st.header("Tambah Wajah Baru ke Dataset")

    # Input nama orang baru

    new\_person = st.text\_input("Masukkan nama orang baru:")

    # Tombol untuk memulai proses penambahan wajah

    capture = st.button("Tambahkan Wajah Baru")

    if capture:

        if not new\_person:

            st.warning("Silakan masukkan nama orang baru.")

        else:

            save\_path = os.path.join('dataset', new\_person)

            if not os.path.exists('dataset'):

                os.makedirs('dataset')

                st.info("Folder 'dataset' telah dibuat.")

            if not os.path.exists(save\_path):

                os.makedirs(save\_path)

                st.success(f"Folder untuk {new\_person} telah dibuat.")

                # Mulai menangkap gambar dari webcam

                cap = cv2.VideoCapture(0)

                if not cap.isOpened():

                    st.error("Tidak dapat membuka webcam. Pastikan webcam terhubung dan tidak digunakan oleh aplikasi lain.")

                else:

                    num\_images = 0

                    max\_images = 20  # Ambil 20 gambar wajah

                    frame\_placeholder = st.empty()

                    progress\_bar = st.progress(0)

                    status\_text = st.empty()

                    try:

                        while num\_images < max\_images:

                            ret, frame = cap.read()

                            if not ret:

                                st.error("Error: Tidak dapat membaca frame dari webcam.")

                                break

                            # Deteksi wajah dalam frame

                            faces = detect\_faces(frame)

                            if len(faces) > 0:

                                for (x, y, w, h) in faces:

                                    face = frame[y:y+h, x:x+w]

                                    img\_name = os.path.join(save\_path, f"img\_{num\_images}.jpg")

                                    cv2.imwrite(img\_name, face)

                                    num\_images += 1

                                    # Menggambar kotak di sekitar wajah yang terdeteksi

                                    cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

                                    # Tampilkan hasil deteksi

                                    frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

                                    frame\_placeholder.image(frame\_rgb, *channels*="RGB", *caption*=f"Gambar {num\_images}/{max\_images}")

                                    # Update progress bar

                                    progress = num\_images / max\_images

                                    progress\_bar.progress(progress)

                                    status\_text.text(f"Menyimpan gambar {num\_images} dari {max\_images}...")

                                    # Hentikan setelah menyimpan satu wajah per frame

                                    break

                            else:

                                # Tampilkan frame tanpa deteksi

                                frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

                                frame\_placeholder.image(frame\_rgb, *channels*="RGB", *caption*="Tidak ada wajah terdeteksi.")

                            time.sleep(0.1)  # Tambahkan delay untuk menghindari penggunaan CPU yang berlebihan

                        st.success(f"{num\_images} gambar telah berhasil ditambahkan ke dataset {new\_person}.")

                    finally:

                        cap.release()

                        frame\_placeholder.empty()

                        progress\_bar.empty()

                        status\_text.empty()

            else:

                st.warning("Nama sudah ada di dataset. Silakan pilih nama lain atau tambahkan lebih banyak gambar.")

with tab2:

    st.header("Lihat dan Proses Dataset")

    # Cek apakah folder dataset ada

    if not os.path.exists('dataset'):

        st.warning("Folder dataset belum dibuat. Silakan tambahkan wajah baru terlebih dahulu.")

    else:

        # Dapatkan daftar orang dalam dataset

        people = [d for d in os.listdir('dataset') if os.path.isdir(os.path.join('dataset', d))]

        if not people:

            st.warning("Belum ada data wajah dalam dataset. Silakan tambahkan wajah baru terlebih dahulu.")

        else:

            selected\_person = st.selectbox("Pilih orang dari dataset:", people)

            # Dapatkan daftar gambar untuk orang yang dipilih

            person\_path = os.path.join('dataset', selected\_person)

            images = [f for f in os.listdir(person\_path) if f.endswith(('.jpg', '.jpeg', '.png')) and os.path.isfile(os.path.join(person\_path, f))]

            if not images:

                st.warning(f"Tidak ada gambar untuk {selected\_person}.")

            else:

                selected\_image = st.selectbox("Pilih gambar untuk diproses:", images)

                # Load gambar yang dipilih

                image\_path = os.path.join(person\_path, selected\_image)

                original\_image = cv2.imread(image\_path)

                original\_image\_rgb = cv2.cvtColor(original\_image, cv2.COLOR\_BGR2RGB)

                # Tampilkan gambar asli

                st.image(original\_image\_rgb, *caption*="Gambar Asli", *use\_column\_width*=True)

                # Buat kolom untuk parameter pengolahan citra

                st.subheader("Opsi Pengolahan Citra")

                # Options for noise

                st.write("### 1. Tambahkan Noise")

                apply\_noise = st.checkbox("Terapkan Noise", *value*=True)

                if apply\_noise:

                    noise\_type = st.radio(

                        "Pilih Jenis Noise:",

                        ["gaussian", "salt\_pepper", "combined"],

*index*=2,

*horizontal*=True

                    )

                    if noise\_type in ["gaussian", "combined"]:

                        col1, col2 = st.columns(2)

                        with col1:

                            noise\_intensity = st.slider("Intensitas Gaussian Noise", 5, 70, 25)

                        with col2:

                            noise\_mean = st.slider("Mean Noise", -10, 10, 0)

                    if noise\_type in ["salt\_pepper", "combined"]:

                        noise\_density = st.slider("Kepadatan Salt & Pepper", 0.01, 0.1, 0.02, 0.01)

                # Options for denoising

                st.write("### 2. Hilangkan Noise")

                apply\_denoising = st.checkbox("Terapkan Denoising", *value*=True)

                if apply\_denoising:

                    denoising\_method = st.radio(

                        "Pilih Metode Denoising:",

                        ["median", "gaussian", "bilateral"],

*horizontal*=True

                    )

                    kernel\_size = st.select\_slider("Ukuran Kernel", *options*=[3, 5, 7, 9], *value*=5)

                # Options for sharpening

                st.write("### 3. Tajamkan Gambar")

                apply\_sharpening = st.checkbox("Terapkan Penajaman", *value*=True)

                if apply\_sharpening:

                    sharpening\_method = st.radio(

                        "Pilih Metode Penajaman:",

                        ["normal", "edge\_enhance", "unsharp\_mask"],

*horizontal*=True

                    )

                # Tombol untuk memproses gambar

                if st.button("Proses Gambar"):

                    # Buat folder untuk menyimpan hasil pengolahan citra jika belum ada

                    processed\_path = os.path.join(person\_path, 'processed')

                    if not os.path.exists(processed\_path):

                        os.makedirs(processed\_path)

                    # Inisialisasi dengan gambar asli

                    processed\_image = original\_image.copy()

                    # Area untuk menampilkan progress

                    progress\_area = st.container()

                    with progress\_area:

                        progress\_text = st.empty()

                        progress\_bar = st.progress(0)

                        # Step 1: Tambahkan noise

                        progress\_text.text("Menambahkan noise...")

                        progress\_bar.progress(0.2)

                        if apply\_noise:

                            # Sesuaikan parameter berdasarkan jenis noise

                            if noise\_type == "gaussian":

                                noisy\_image = add\_gaussian\_noise(processed\_image, noise\_mean, noise\_intensity)

                            elif noise\_type == "salt\_pepper":

                                noisy\_image = add\_salt\_pepper\_noise(processed\_image, noise\_density)

                            elif noise\_type == "combined":

                                noisy\_image = add\_combined\_noise(

                                    processed\_image,

                                    noise\_type,

                                    noise\_density,

                                    noise\_mean,

                                    noise\_intensity

                                )

                            processed\_image = noisy\_image

                            # Tampilkan hasil

                            st.subheader("Hasil Penambahan Noise")

                            display\_before\_after(

                                cv2.cvtColor(original\_image, cv2.COLOR\_BGR2RGB),

                                cv2.cvtColor(noisy\_image, cv2.COLOR\_BGR2RGB),

                                f"Dengan {noise\_type.replace('\_', ' ').title()} Noise"

                            )

                        # Step 2: Hilangkan noise

                        progress\_text.text("Menghilangkan noise...")

                        progress\_bar.progress(0.5)

                        if apply\_denoising:

                            before\_denoising = processed\_image.copy()

                            denoised\_image = remove\_noise(processed\_image, denoising\_method, kernel\_size)

                            processed\_image = denoised\_image

                            # Tampilkan hasil

                            st.subheader("Hasil Penghilangan Noise")

                            display\_before\_after(

                                cv2.cvtColor(before\_denoising, cv2.COLOR\_BGR2RGB),

                                cv2.cvtColor(denoised\_image, cv2.COLOR\_BGR2RGB),

                                f"Dengan Filter {denoising\_method.title()}"

                            )

                        # Step 3: Tajamkan gambar

                        progress\_text.text("Menajamkan gambar...")

                        progress\_bar.progress(0.8)

                        if apply\_sharpening:

                            before\_sharpening = processed\_image.copy()

                            sharpened\_image = sharpen\_image(processed\_image, sharpening\_method)

                            processed\_image = sharpened\_image

                            # Tampilkan hasil

                            st.subheader("Hasil Penajaman Gambar")

                            display\_before\_after(

                                cv2.cvtColor(before\_sharpening, cv2.COLOR\_BGR2RGB),

                                cv2.cvtColor(sharpened\_image, cv2.COLOR\_BGR2RGB),

                                f"Dengan Metode {sharpening\_method.replace('\_', ' ').title()}"

                            )

                        # Selesai

                        progress\_text.text("Pengolahan selesai!")

                        progress\_bar.progress(1.0)

                        # Simpan hasil akhir

                        processed\_image\_name = f"processed\_{selected\_image}"

                        processed\_image\_path = os.path.join(processed\_path, processed\_image\_name)

                        cv2.imwrite(processed\_image\_path, processed\_image)

                        # Tampilkan hasil akhir dibandingkan dengan gambar asli

                        st.subheader("Hasil Akhir")

                        display\_before\_after(

                            cv2.cvtColor(original\_image, cv2.COLOR\_BGR2RGB),

                            cv2.cvtColor(processed\_image, cv2.COLOR\_BGR2RGB),

                            "Hasil Pengolahan Akhir"

                        )

                        st.success(f"Gambar telah diproses dan disimpan sebagai {processed\_image\_name}")

import cv2

import numpy as np

import matplotlib.pyplot as plt

from fastapi import FastAPI, File, UploadFile, HTTPException, Request

from fastapi.responses import JSONResponse, FileResponse, HTMLResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

import base64

from io import BytesIO

import os

from pathlib import Path

import uvicorn

app = FastAPI(

    title="Image Processing API",

    description="API for image contour and shape feature analysis using OpenCV",

    version="1.0.0"

)

TEMP\_DIR = Path("temp")

TEMP\_DIR.mkdir(exist\_ok=True)

STATIC\_DIR = Path("static")

STATIC\_DIR.mkdir(exist\_ok=True)

HTML\_PATH = STATIC\_DIR / "index.html"

with open(HTML\_PATH, "w") as f:

    f.write("""<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Image Processing UI</title>

    <script src="https://cdn.jsdelivr.net/npm/react@18/umd/react.development.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/react-dom@18/umd/react-dom.development.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@babel/standalone@7/babel.min.js"></script>

    <script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-gray-100 min-h-screen flex items-center justify-center">

    <div id="root"></div>

    <script type="text/babel">

        const { useState, useEffect } = React;

        const App = () => {

            const [image, setImage] = useState(null);

            const [preview, setPreview] = useState(null);

            const [processingType, setProcessingType] = useState('freeman-chain-code');

            const [threshold, setThreshold] = useState(127);

            const [lowThreshold, setLowThreshold] = useState(50);

            const [highThreshold, setHighThreshold] = useState(150);

            const [result, setResult] = useState(null);

            const [loading, setLoading] = useState(false);

            const [error, setError] = useState(null);

            const [useSyntheticImage, setUseSyntheticImage] = useState(false);

            const [syntheticImage, setSyntheticImage] = useState(null);

            const handleImageChange = (e) => {

                const file = e.target.files[0];

                if (file) {

                    setImage(file);

                    setUseSyntheticImage(false);

                    const reader = new FileReader();

                    reader.onloadend = () => {

                        setPreview(reader.result);

                    };

                    reader.readAsDataURL(file);

                }

            };

            const handleGenerateSyntheticImage = async () => {

                setLoading(true);

                setError(null);

                setResult(null);

                setPreview(null);

                setImage(null);

                try {

                    const response = await fetch('/generate-synthetic-text-image', {

                        method: 'GET',

                    });

                    if (!response.ok) {

                        throw new Error('Failed to generate synthetic image.');

                    }

                    const blob = await response.blob();

                    const imageUrl = URL.createObjectURL(blob);

                    setSyntheticImage(blob);

                    setPreview(imageUrl);

                    setUseSyntheticImage(true);

                } catch (err) {

                    setError(err.message);

                } finally {

                    setLoading(false);

                }

            };

            const handleSubmit = async (e) => {

                e.preventDefault();

                if (!image && !useSyntheticImage) {

                    setError('Please upload an image or generate a synthetic image.');

                    return;

                }

                setLoading(true);

                setError(null);

                setResult(null);

                const formData = new FormData();

                formData.append('file', useSyntheticImage ? syntheticImage : image);

                let url = '';

                let params = {};

                if (processingType === 'freeman-chain-code') {

                    url = '/freeman-chain-code';

                    params = { threshold\_value: threshold };

                } else if (processingType === 'canny-edge-detection') {

                    url = '/canny-edge-detection';

                    params = { low\_threshold: lowThreshold, high\_threshold: highThreshold };

                } else if (processingType === 'integral-projection') {

                    url = '/integral-projection';

                }

                try {

                    const queryString = new URLSearchParams(params).toString();

                    const response = await fetch(`${url}${queryString ? '?' + queryString : ''}`, {

                        method: 'POST',

                        body: formData,

                    });

                    if (!response.ok) {

                        throw new Error('Failed to process the image.');

                    }

                    if (processingType === 'canny-edge-detection') {

                        const blob = await response.blob();

                        const imageUrl = URL.createObjectURL(blob);

                        setResult({ edgeMap: imageUrl });

                    } else {

                        const data = await response.json();

                        setResult(data);

                    }

                } catch (err) {

                    setError(err.message);

                } finally {

                    setLoading(false);

                }

            };

            return (

                <div className="max-w-4xl w-full bg-white p-8 rounded-lg shadow-lg">

                    <h1 className="text-3xl font-bold text-center mb-6 text-gray-800">

                        Image Processing UI

                    </h1>

                    <div className="space-y-6">

                        <div>

                            <label className="block text-sm font-medium text-gray-700 mb-2">

                                Upload Image

                            </label>

                            <input

                                type="file"

                                accept="image/\*"

                                onChange={handleImageChange}

                                disabled={useSyntheticImage}

                                className="block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded file:border-0 file:text-sm file:font-semibold file:bg-blue-50 file:text-blue-700 hover:file:bg-blue-100 disabled:opacity-50"

                            />

                        </div>

                        <div>

                            <button

                                onClick={handleGenerateSyntheticImage}

                                disabled={loading}

                                className={`w-full py-2 px-4 rounded text-white font-semibold ${

                                    loading ? 'bg-gray-400 cursor-not-allowed' : 'bg-green-600 hover:bg-green-700'

                                } transition duration-200`}

                            >

                                {loading ? 'Generating...' : 'Generate Synthetic Text Image'}

                            </button>

                        </div>

                        {preview && (

                            <div>

                                <h3 className="text-lg font-medium text-gray-700">Image Preview</h3>

                                <img

                                    src={preview}

                                    alt="Preview"

                                    className="mt-4 w-48 h-48 object-contain rounded border border-gray-200"

                                />

                            </div>

                        )}

                        <div>

                            <label className="block text-sm font-medium text-gray-700 mb-2">

                                Processing Type

                            </label>

                            <select

                                value={processingType}

                                onChange={(e) => setProcessingType(e.target.value)}

                                className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                            >

                                <option value="freeman-chain-code">Freeman Chain Code</option>

                                <option value="canny-edge-detection">Canny Edge Detection</option>

                                <option value="integral-projection">Integral Projection</option>

                            </select>

                        </div>

                        {processingType === 'freeman-chain-code' && (

                            <div>

                                <label className="block text-sm font-medium text-gray-700 mb-2">

                                    Threshold Value

                                </label>

                                <input

                                    type="number"

                                    value={threshold}

                                    onChange={(e) => setThreshold(Number(e.target.value))}

                                    min="0"

                                    max="255"

                                    className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                />

                            </div>

                        )}

                        {processingType === 'canny-edge-detection' && (

                            <div className="space-y-4">

                                <div>

                                    <label className="block text-sm font-medium text-gray-700 mb-2">

                                        Low Threshold

                                    </label>

                                    <input

                                        type="number"

                                        value={lowThreshold}

                                        onChange={(e) => setLowThreshold(Number(e.target.value))}

                                        min="0"

                                        max="255"

                                        className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                    />

                                </div>

                                <div>

                                    <label className="block text-sm font-medium text-gray-700 mb-2">

                                        High Threshold

                                    </label>

                                    <input

                                        type="number"

                                        value={highThreshold}

                                        onChange={(e) => setHighThreshold(Number(e.target.value))}

                                        min="0"

                                        max="255"

                                        className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                    />

                                </div>

                            </div>

                        )}

                        <button

                            onClick={handleSubmit}

                            disabled={loading}

                            className={`w-full py-2 px-4 rounded text-white font-semibold ${

                                loading ? 'bg-blue-400 cursor-not-allowed' : 'bg-blue-600 hover:bg-blue-700'

                            } transition duration-200`}

                        >

                            {loading ? 'Processing...' : 'Process Image'}

                        </button>

                    </div>

                    {error && (

                        <div className="mt-6 p-4 bg-red-100 text-red-700 rounded">

                            {error}

                        </div>

                    )}

                    {result && (

                        <div className="mt-6 space-y-4">

                            <h2 className="text-xl font-semibold text-gray-800">Results</h2>

                            {processingType === 'freeman-chain-code' && result.chain\_code && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Chain Code</h3>

                                    <pre className="bg-gray-100 p-4 rounded overflow-x-auto text-sm">

                                        {JSON.stringify(result.chain\_code, null, 2)}

                                    </pre>

                                    {result.visualization && (

                                        <img

                                            src={result.visualization}

                                            alt="Visualization"

                                            className="mt-4 w-full rounded border border-gray-200"

                                        />

                                    )}

                                </div>

                            )}

                            {processingType === 'canny-edge-detection' && result.edgeMap && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Edge Map</h3>

                                    <img

                                        src={result.edgeMap}

                                        alt="Edge Map"

                                        className="mt-4 w-full rounded border border-gray-200"

                                    />

                                </div>

                            )}

                            {processingType === 'integral-projection' && result.horizontal\_projection && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Projections</h3>

                                    <p><strong>Horizontal Projection:</strong> {JSON.stringify(result.horizontal\_projection)}</p>

                                    <p><strong>Vertical Projection:</strong> {JSON.stringify(result.vertical\_projection)}</p>

                                    {result.visualization && (

                                        <img

                                            src={result.visualization}

                                            alt="Projection Visualization"

                                            className="mt-4 w-full rounded border border-gray-200"

                                        />

                                    )}

                                </div>

                            )}

                        </div>

                    )}

                </div>

            );

        };

        ReactDOM.render(<App />, document.getElementById('root'));

    </script>

</body>

</html>""")

app.mount("/static", StaticFiles(directory=STATIC\_DIR), name="static")

IMAGE\_CONTENT\_TYPE\_PREFIX = "image/"

ERROR\_INVALID\_IMAGE = "Uploaded file must be an image"

ERROR\_READ\_IMAGE = "Could not read the image"

def save\_plot\_to\_base64():

    buffer = BytesIO()

    plt.savefig(buffer, format="png", bbox\_inches="tight")

    plt.close()

    buffer.seek(0)

    img\_str = base64.b64encode(buffer.getvalue()).decode("utf-8")

    return img\_str

def save\_image\_to\_file(image: np.ndarray, filename: str) -> Path:

    filepath = TEMP\_DIR / filename

    cv2.imwrite(str(filepath), image)

    return filepath

def generate\_freeman\_chain\_code(contour):

    chain\_code = []

    if len(contour) < 2:

        return chain\_code

    directions = {

        (1, 0): 0, (1, 1): 1, (0, 1): 2, (-1, 1): 3,

        (-1, 0): 4, (-1, -1): 5, (0, -1): 6, (1, -1): 7

    }

    for i in range(len(contour)):

        p1 = contour[i][0]

        p2 = contour[(i + 1) % len(contour)][0]

        dx = p2[0] - p1[0]

        dy = p2[1] - p1[1]

        norm\_dx = np.sign(dx)

        norm\_dy = np.sign(dy)

        code = directions.get((norm\_dx, norm\_dy))

        if code is not None:

            chain\_code.append(code)

    return chain\_code

@app.get("/", response\_class=HTMLResponse)

async def root():

    with open(HTML\_PATH) as f:

        return f.read()

@app.post("/freeman-chain-code")

async def freeman\_chain\_code\_endpoint(file: UploadFile = File(...), threshold\_value: int = 127):

    if not file.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(status\_code=400, detail=ERROR\_INVALID\_IMAGE)

    contents = await file.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        raise HTTPException(status\_code=400, detail=ERROR\_READ\_IMAGE)

    \_, binary\_img = cv2.threshold(img, threshold\_value, 255, cv2.THRESH\_BINARY\_INV)

    contours, \_ = cv2.findContours(binary\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

    if not contours:

        return JSONResponse(content={"message": "No contours detected", "chain\_code": []})

    largest\_contour = max(contours, key=cv2.contourArea)

    chain\_code\_result = generate\_freeman\_chain\_code(largest\_contour)

    img\_contour\_display = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

    cv2.drawContours(img\_contour\_display, [largest\_contour], -1, (0, 255, 0), 1)

    fig = plt.figure(figsize=(10, 8))

    axs = fig.subplots(2, 2)

    axs[0, 0].imshow(img, cmap='gray')

    axs[0, 0].set\_title('Original Image (Grayscale)')

    axs[0, 0].axis('off')

    axs[0, 1].imshow(binary\_img, cmap='gray')

    axs[0, 1].set\_title('Binary Image')

    axs[0, 1].axis('off')

    img\_rgb\_display = cv2.cvtColor(img\_contour\_display, cv2.COLOR\_BGR2RGB)

    axs[1, 0].imshow(img\_rgb\_display)

    axs[1, 0].set\_title('Largest Contour Detected')

    axs[1, 0].axis('off')

    wrapped\_code = ""

    current\_line\_len = 0

    for i, code in enumerate(chain\_code\_result):

        item = str(code) + (", " if i < len(chain\_code\_result) - 1 else "")

        if current\_line\_len + len(item) > 70:

            wrapped\_code += "\n"

            current\_line\_len = 0

        wrapped\_code += item

        current\_line\_len += len(item)

    chain\_code\_str = (

        f"Total Contours: {len(contours)}\n"

        f"Chain Code Length: {len(chain\_code\_result)}\n"

        f"{wrapped\_code}"

    )

    axs[1, 1].axis('off')

    axs[1, 1].text(0.05, 0.95, chain\_code\_str, ha='left', va='top', fontsize=9, wrap=True)

    axs[1, 1].set\_title('Chain Code Result')

    plt.tight\_layout(pad=1.5)

    plt.suptitle("Freeman Chain Code Analysis", fontsize=16)

    plt.subplots\_adjust(top=0.92)

    plot\_base64 = save\_plot\_to\_base64()

    return JSONResponse(content={

        "chain\_code": chain\_code\_result,

        "visualization": f"data:image/png;base64,{plot\_base64}"

    })

@app.post("/canny-edge-detection")

async def canny\_edge\_detection\_endpoint(

    file: UploadFile = File(...),

    low\_threshold: int = 50,

    high\_threshold: int = 150

):

    if not file.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(status\_code=400, detail=ERROR\_INVALID\_IMAGE)

    contents = await file.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_COLOR)

    if img is None:

        raise HTTPException(status\_code=400, detail=ERROR\_READ\_IMAGE)

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    edges = cv2.Canny(blurred, low\_threshold, high\_threshold)

    edge\_filepath = save\_image\_to\_file(edges, "canny\_edges.png")

    return FileResponse(

        edge\_filepath,

        media\_type="image/png",

        filename="canny\_edges.png"

    )

@app.post("/integral-projection")

async def integral\_projection\_endpoint(file: UploadFile = File(...)):

    if not file.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(status\_code=400, detail=ERROR\_INVALID\_IMAGE)

    contents = await file.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        raise HTTPException(status\_code=400, detail=ERROR\_READ\_IMAGE)

    \_, binary\_img = cv2.threshold(img, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

    binary\_norm = binary\_img / 255.0

    horizontal\_projection = np.sum(binary\_norm, axis=0).tolist()

    vertical\_projection = np.sum(binary\_norm, axis=1).tolist()

    height, width = binary\_norm.shape

    fig = plt.figure(figsize=(10, 8))

    gs = fig.add\_gridspec(2, 2, width\_ratios=(4, 1), height\_ratios=(1, 4),

                          left=0.1, right=0.9, bottom=0.1, top=0.9,

                          wspace=0.05, hspace=0.05)

    ax\_img = fig.add\_subplot(gs[1, 0])

    ax\_img.imshow(binary\_norm, cmap='gray')

    ax\_img.set\_title('Binary Image (Object=1)')

    ax\_img.set\_xlabel('Column Index')

    ax\_img.set\_ylabel('Row Index')

    ax\_hproj = fig.add\_subplot(gs[0, 0], sharex=ax\_img)

    ax\_hproj.plot(np.arange(width), horizontal\_projection)

    ax\_hproj.set\_title('Horizontal Projection')

    ax\_hproj.set\_ylabel('Pixel Sum')

    plt.setp(ax\_hproj.get\_xticklabels(), visible=False)

    ax\_vproj = fig.add\_subplot(gs[1, 1], sharey=ax\_img)

    ax\_vproj.plot(vertical\_projection, np.arange(height))

    ax\_vproj.set\_title('Vertical Projection')

    ax\_vproj.set\_xlabel('Pixel Sum')

    ax\_vproj.invert\_yaxis()

    plt.setp(ax\_vproj.get\_yticklabels(), visible=False)

    plt.suptitle("Integral Projection Analysis", fontsize=14)

    plot\_base64 = save\_plot\_to\_base64()

    return JSONResponse(content={

        "horizontal\_projection": horizontal\_projection,

        "vertical\_projection": vertical\_projection,

        "visualization": f"data:image/png;base64,{plot\_base64}"

    })

@app.get("/generate-synthetic-text-image")

async def generate\_synthetic\_text\_image():

    height, width = 150, 400

    binary\_img = np.zeros((height, width), dtype=np.uint8)

    text\_lines = ["Baris Teks Satu", "Ini Baris Dua", "Testing 123"]

    start\_y = 40

    line\_height = 40

    font = cv2.FONT\_HERSHEY\_SIMPLEX

    font\_scale = 0.8

    font\_color = 255

    thickness = 2

    y = start\_y

    for line in text\_lines:

        x = 20

        cv2.putText(binary\_img, line, (x, y), font, font\_scale, font\_color, thickness)

        y += line\_height

    filepath = save\_image\_to\_file(binary\_img, "synthetic\_text.png")

    return FileResponse(

        filepath,

        media\_type="image/png",

        filename="synthetic\_text.png"

    )

@app.on\_event("shutdown")

async def cleanup():

    import shutil

    if TEMP\_DIR.exists():

        shutil.rmtree(TEMP\_DIR)

if \_\_name\_\_ == "\_\_main\_\_":

    uvicorn.run(app, host="0.0.0.0", port=8000)

import cv2

import numpy as np

import matplotlib.pyplot as plt

from fastapi import FastAPI, File, UploadFile, HTTPException, Request

from fastapi.responses import JSONResponse, FileResponse, HTMLResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

import base64

from io import BytesIO

import os

from pathlib import Path

import uvicorn

app = FastAPI(

*title*="Image Processing API",

*description*="API for image contour and shape feature analysis using OpenCV",

*version*="1.0.0"

)

TEMP\_DIR = Path("temp")

TEMP\_DIR.mkdir(*exist\_ok*=True)

STATIC\_DIR = Path("static")

STATIC\_DIR.mkdir(*exist\_ok*=True)

HTML\_PATH = STATIC\_DIR / "index.html"

with open(HTML\_PATH, "w") as f:

    f.write("""<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Image Processing UI</title>

    <script src="https://cdn.jsdelivr.net/npm/react@18/umd/react.development.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/react-dom@18/umd/react-dom.development.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@babel/standalone@7/babel.min.js"></script>

    <script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-gray-100 min-h-screen flex items-center justify-center">

    <div id="root"></div>

    <script type="text/babel">

        const { useState, useEffect } = React;

        const App = () => {

            const [image, setImage] = useState(null);

            const [preview, setPreview] = useState(null);

            const [processingType, setProcessingType] = useState('freeman-chain-code');

            const [threshold, setThreshold] = useState(127);

            const [lowThreshold, setLowThreshold] = useState(50);

            const [highThreshold, setHighThreshold] = useState(150);

            const [result, setResult] = useState(null);

            const [loading, setLoading] = useState(false);

            const [error, setError] = useState(null);

            const [useSyntheticImage, setUseSyntheticImage] = useState(false);

            const [syntheticImage, setSyntheticImage] = useState(null);

            const handleImageChange = (e) => {

                const file = e.target.files[0];

                if (file) {

                    setImage(file);

                    setUseSyntheticImage(false);

                    const reader = new FileReader();

                    reader.onloadend = () => {

                        setPreview(reader.result);

                    };

                    reader.readAsDataURL(file);

                }

            };

            const handleGenerateSyntheticImage = async () => {

                setLoading(true);

                setError(null);

                setResult(null);

                setPreview(null);

                setImage(null);

                try {

                    const response = await fetch('/generate-synthetic-text-image', {

                        method: 'GET',

                    });

                    if (!response.ok) {

                        throw new Error('Failed to generate synthetic image.');

                    }

                    const blob = await response.blob();

                    const imageUrl = URL.createObjectURL(blob);

                    setSyntheticImage(blob);

                    setPreview(imageUrl);

                    setUseSyntheticImage(true);

                } catch (err) {

                    setError(err.message);

                } finally {

                    setLoading(false);

                }

            };

            const handleSubmit = async (e) => {

                e.preventDefault();

                if (!image && !useSyntheticImage) {

                    setError('Please upload an image or generate a synthetic image.');

                    return;

                }

                setLoading(true);

                setError(null);

                setResult(null);

                const formData = new FormData();

                formData.append('file', useSyntheticImage ? syntheticImage : image);

                let url = '';

                let params = {};

                if (processingType === 'freeman-chain-code') {

                    url = '/freeman-chain-code';

                    params = { threshold\_value: threshold };

                } else if (processingType === 'canny-edge-detection') {

                    url = '/canny-edge-detection';

                    params = { low\_threshold: lowThreshold, high\_threshold: highThreshold };

                } else if (processingType === 'integral-projection') {

                    url = '/integral-projection';

                }

                try {

                    const queryString = new URLSearchParams(params).toString();

                    const response = await fetch(`${url}${queryString ? '?' + queryString : ''}`, {

                        method: 'POST',

                        body: formData,

                    });

                    if (!response.ok) {

                        throw new Error('Failed to process the image.');

                    }

                    if (processingType === 'canny-edge-detection') {

                        const blob = await response.blob();

                        const imageUrl = URL.createObjectURL(blob);

                        setResult({ edgeMap: imageUrl });

                    } else {

                        const data = await response.json();

                        setResult(data);

                    }

                } catch (err) {

                    setError(err.message);

                } finally {

                    setLoading(false);

                }

            };

            return (

                <div className="max-w-4xl w-full bg-white p-8 rounded-lg shadow-lg">

                    <h1 className="text-3xl font-bold text-center mb-6 text-gray-800">

                        Image Processing UI

                    </h1>

                    <div className="space-y-6">

                        <div>

                            <label className="block text-sm font-medium text-gray-700 mb-2">

                                Upload Image

                            </label>

                            <input

                                type="file"

                                accept="image/\*"

                                onChange={handleImageChange}

                                disabled={useSyntheticImage}

                                className="block w-full text-sm text-gray-500 file:mr-4 file:py-2 file:px-4 file:rounded file:border-0 file:text-sm file:font-semibold file:bg-blue-50 file:text-blue-700 hover:file:bg-blue-100 disabled:opacity-50"

                            />

                        </div>

                        <div>

                            <button

                                onClick={handleGenerateSyntheticImage}

                                disabled={loading}

                                className={`w-full py-2 px-4 rounded text-white font-semibold ${

                                    loading ? 'bg-gray-400 cursor-not-allowed' : 'bg-green-600 hover:bg-green-700'

                                } transition duration-200`}

                            >

                                {loading ? 'Generating...' : 'Generate Synthetic Text Image'}

                            </button>

                        </div>

                        {preview && (

                            <div>

                                <h3 className="text-lg font-medium text-gray-700">Image Preview</h3>

                                <img

                                    src={preview}

                                    alt="Preview"

                                    className="mt-4 w-48 h-48 object-contain rounded border border-gray-200"

                                />

                            </div>

                        )}

                        <div>

                            <label className="block text-sm font-medium text-gray-700 mb-2">

                                Processing Type

                            </label>

                            <select

                                value={processingType}

                                onChange={(e) => setProcessingType(e.target.value)}

                                className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                            >

                                <option value="freeman-chain-code">Freeman Chain Code</option>

                                <option value="canny-edge-detection">Canny Edge Detection</option>

                                <option value="integral-projection">Integral Projection</option>

                            </select>

                        </div>

                        {processingType === 'freeman-chain-code' && (

                            <div>

                                <label className="block text-sm font-medium text-gray-700 mb-2">

                                    Threshold Value

                                </label>

                                <input

                                    type="number"

                                    value={threshold}

                                    onChange={(e) => setThreshold(Number(e.target.value))}

                                    min="0"

                                    max="255"

                                    className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                />

                            </div>

                        )}

                        {processingType === 'canny-edge-detection' && (

                            <div className="space-y-4">

                                <div>

                                    <label className="block text-sm font-medium text-gray-700 mb-2">

                                        Low Threshold

                                    </label>

                                    <input

                                        type="number"

                                        value={lowThreshold}

                                        onChange={(e) => setLowThreshold(Number(e.target.value))}

                                        min="0"

                                        max="255"

                                        className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                    />

                                </div>

                                <div>

                                    <label className="block text-sm font-medium text-gray-700 mb-2">

                                        High Threshold

                                    </label>

                                    <input

                                        type="number"

                                        value={highThreshold}

                                        onChange={(e) => setHighThreshold(Number(e.target.value))}

                                        min="0"

                                        max="255"

                                        className="block w-full p-2 border border-gray-300 rounded focus:outline-none focus:ring-2 focus:ring-blue-500"

                                    />

                                </div>

                            </div>

                        )}

                        <button

                            onClick={handleSubmit}

                            disabled={loading}

                            className={`w-full py-2 px-4 rounded text-white font-semibold ${

                                loading ? 'bg-blue-400 cursor-not-allowed' : 'bg-blue-600 hover:bg-blue-700'

                            } transition duration-200`}

                        >

                            {loading ? 'Processing...' : 'Process Image'}

                        </button>

                    </div>

                    {error && (

                        <div className="mt-6 p-4 bg-red-100 text-red-700 rounded">

                            {error}

                        </div>

                    )}

                    {result && (

                        <div className="mt-6 space-y-4">

                            <h2 className="text-xl font-semibold text-gray-800">Results</h2>

                            {processingType === 'freeman-chain-code' && result.chain\_code && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Chain Code</h3>

                                    <pre className="bg-gray-100 p-4 rounded overflow-x-auto text-sm">

                                        {JSON.stringify(result.chain\_code, null, 2)}

                                    </pre>

                                    {result.visualization && (

                                        <img

                                            src={result.visualization}

                                            alt="Visualization"

                                            className="mt-4 w-full rounded border border-gray-200"

                                        />

                                    )}

                                </div>

                            )}

                            {processingType === 'canny-edge-detection' && result.edgeMap && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Edge Map</h3>

                                    <img

                                        src={result.edgeMap}

                                        alt="Edge Map"

                                        className="mt-4 w-full rounded border border-gray-200"

                                    />

                                </div>

                            )}

                            {processingType === 'integral-projection' && result.horizontal\_projection && (

                                <div>

                                    <h3 className="text-lg font-medium text-gray-700">Projections</h3>

                                    <p><strong>Horizontal Projection:</strong> {JSON.stringify(result.horizontal\_projection)}</p>

                                    <p><strong>Vertical Projection:</strong> {JSON.stringify(result.vertical\_projection)}</p>

                                    {result.visualization && (

                                        <img

                                            src={result.visualization}

                                            alt="Projection Visualization"

                                            className="mt-4 w-full rounded border border-gray-200"

                                        />

                                    )}

                                </div>

                            )}

                        </div>

                    )}

                </div>

            );

        };

        ReactDOM.render(<App />, document.getElementById('root'));

    </script>

</body>

</html>""")

app.mount("/static", StaticFiles(*directory*=STATIC\_DIR), *name*="static")

IMAGE\_CONTENT\_TYPE\_PREFIX = "image/"

ERROR\_INVALID\_IMAGE = "Uploaded file must be an image"

ERROR\_READ\_IMAGE = "Could not read the image"

def save\_plot\_to\_base64():

    buffer = BytesIO()

    plt.savefig(buffer, *format*="png", *bbox\_inches*="tight")

    plt.close()

    buffer.seek(0)

    img\_str = base64.b64encode(buffer.getvalue()).decode("utf-8")

    return img\_str

def save\_image\_to\_file(*image*: np.ndarray, *filename*: str) -> Path:

    filepath = TEMP\_DIR / *filename*

    cv2.imwrite(str(filepath), *image*)

    return filepath

def generate\_freeman\_chain\_code(*contour*):

    chain\_code = []

    if len(*contour*) < 2:

        return chain\_code

    directions = {

        (1, 0): 0, (1, 1): 1, (0, 1): 2, (-1, 1): 3,

        (-1, 0): 4, (-1, -1): 5, (0, -1): 6, (1, -1): 7

    }

    for i in range(len(*contour*)):

        p1 = *contour*[i][0]

        p2 = *contour*[(i + 1) % len(*contour*)][0]

        dx = p2[0] - p1[0]

        dy = p2[1] - p1[1]

        norm\_dx = np.sign(dx)

        norm\_dy = np.sign(dy)

        code = directions.get((norm\_dx, norm\_dy))

        if code is not None:

            chain\_code.append(code)

    return chain\_code

@app.get("/", *response\_class*=HTMLResponse)

async def root():

    with open(HTML\_PATH) as f:

        return f.read()

@app.post("/freeman-chain-code")

async def freeman\_chain\_code\_endpoint(*file*: UploadFile = File(...), *threshold\_value*: int = 127):

    if not *file*.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_INVALID\_IMAGE)

    contents = await *file*.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_READ\_IMAGE)

    \_, binary\_img = cv2.threshold(img, *threshold\_value*, 255, cv2.THRESH\_BINARY\_INV)

    contours, \_ = cv2.findContours(binary\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

    if not contours:

        return JSONResponse(*content*={"message": "No contours detected", "chain\_code": []})

    largest\_contour = max(contours, *key*=cv2.contourArea)

    chain\_code\_result = generate\_freeman\_chain\_code(largest\_contour)

    img\_contour\_display = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

    cv2.drawContours(img\_contour\_display, [largest\_contour], -1, (0, 255, 0), 1)

    fig = plt.figure(*figsize*=(10, 8))

    axs = fig.subplots(2, 2)

    axs[0, 0].imshow(img, *cmap*='gray')

    axs[0, 0].set\_title('Original Image (Grayscale)')

    axs[0, 0].axis('off')

    axs[0, 1].imshow(binary\_img, *cmap*='gray')

    axs[0, 1].set\_title('Binary Image')

    axs[0, 1].axis('off')

    img\_rgb\_display = cv2.cvtColor(img\_contour\_display, cv2.COLOR\_BGR2RGB)

    axs[1, 0].imshow(img\_rgb\_display)

    axs[1, 0].set\_title('Largest Contour Detected')

    axs[1, 0].axis('off')

    wrapped\_code = ""

    current\_line\_len = 0

    for i, code in enumerate(chain\_code\_result):

        item = str(code) + (", " if i < len(chain\_code\_result) - 1 else "")

        if current\_line\_len + len(item) > 70:

            wrapped\_code += "\n"

            current\_line\_len = 0

        wrapped\_code += item

        current\_line\_len += len(item)

    chain\_code\_str = (

        f"Total Contours: {len(contours)}\n"

        f"Chain Code Length: {len(chain\_code\_result)}\n"

        f"{wrapped\_code}"

    )

    axs[1, 1].axis('off')

    axs[1, 1].text(0.05, 0.95, chain\_code\_str, *ha*='left', *va*='top', *fontsize*=9, *wrap*=True)

    axs[1, 1].set\_title('Chain Code Result')

    plt.tight\_layout(*pad*=1.5)

    plt.suptitle("Freeman Chain Code Analysis", *fontsize*=16)

    plt.subplots\_adjust(*top*=0.92)

    plot\_base64 = save\_plot\_to\_base64()

    return JSONResponse(*content*={

        "chain\_code": chain\_code\_result,

        "visualization": f"data:image/png;base64,{plot\_base64}"

    })

@app.post("/canny-edge-detection")

async def canny\_edge\_detection\_endpoint(

*file*: UploadFile = File(...),

*low\_threshold*: int = 50,

*high\_threshold*: int = 150

):

    if not *file*.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_INVALID\_IMAGE)

    contents = await *file*.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_COLOR)

    if img is None:

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_READ\_IMAGE)

    gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    blurred = cv2.GaussianBlur(gray, (5, 5), 0)

    edges = cv2.Canny(blurred, *low\_threshold*, *high\_threshold*)

    edge\_filepath = save\_image\_to\_file(edges, "canny\_edges.png")

    return FileResponse(

        edge\_filepath,

*media\_type*="image/png",

*filename*="canny\_edges.png"

    )

@app.post("/integral-projection")

async def integral\_projection\_endpoint(*file*: UploadFile = File(...)):

    if not *file*.content\_type.startswith(IMAGE\_CONTENT\_TYPE\_PREFIX):

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_INVALID\_IMAGE)

    contents = await *file*.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        raise HTTPException(*status\_code*=400, *detail*=ERROR\_READ\_IMAGE)

    \_, binary\_img = cv2.threshold(img, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

    binary\_norm = binary\_img / 255.0

    horizontal\_projection = np.sum(binary\_norm, *axis*=0).tolist()

    vertical\_projection = np.sum(binary\_norm, *axis*=1).tolist()

    height, width = binary\_norm.shape

    fig = plt.figure(*figsize*=(10, 8))

    gs = fig.add\_gridspec(2, 2, *width\_ratios*=(4, 1), *height\_ratios*=(1, 4),

*left*=0.1, *right*=0.9, *bottom*=0.1, *top*=0.9,

*wspace*=0.05, *hspace*=0.05)

    ax\_img = fig.add\_subplot(gs[1, 0])

    ax\_img.imshow(binary\_norm, *cmap*='gray')

    ax\_img.set\_title('Binary Image (Object=1)')

    ax\_img.set\_xlabel('Column Index')

    ax\_img.set\_ylabel('Row Index')

    ax\_hproj = fig.add\_subplot(gs[0, 0], *sharex*=ax\_img)

    ax\_hproj.plot(np.arange(width), horizontal\_projection)

    ax\_hproj.set\_title('Horizontal Projection')

    ax\_hproj.set\_ylabel('Pixel Sum')

    plt.setp(ax\_hproj.get\_xticklabels(), *visible*=False)

    ax\_vproj = fig.add\_subplot(gs[1, 1], *sharey*=ax\_img)

    ax\_vproj.plot(vertical\_projection, np.arange(height))

    ax\_vproj.set\_title('Vertical Projection')

    ax\_vproj.set\_xlabel('Pixel Sum')

    ax\_vproj.invert\_yaxis()

    plt.setp(ax\_vproj.get\_yticklabels(), *visible*=False)

    plt.suptitle("Integral Projection Analysis", *fontsize*=14)

    plot\_base64 = save\_plot\_to\_base64()

    return JSONResponse(*content*={

        "horizontal\_projection": horizontal\_projection,

        "vertical\_projection": vertical\_projection,

        "visualization": f"data:image/png;base64,{plot\_base64}"

    })

@app.get("/generate-synthetic-text-image")

async def generate\_synthetic\_text\_image():

    height, width = 150, 400

    binary\_img = np.zeros((height, width), *dtype*=np.uint8)

    text\_lines = ["Baris Teks Satu", "Ini Baris Dua", "Testing 123"]

    start\_y = 40

    line\_height = 40

    font = cv2.FONT\_HERSHEY\_SIMPLEX

    font\_scale = 0.8

    font\_color = 255

    thickness = 2

    y = start\_y

    for line in text\_lines:

        x = 20

        cv2.putText(binary\_img, line, (x, y), font, font\_scale, font\_color, thickness)

        y += line\_height

    filepath = save\_image\_to\_file(binary\_img, "synthetic\_text.png")

    return FileResponse(

        filepath,

*media\_type*="image/png",

*filename*="synthetic\_text.png"

    )

@app.on\_event("shutdown")

async def cleanup():

    import shutil

    if TEMP\_DIR.exists():

        shutil.rmtree(TEMP\_DIR)

if \_\_name\_\_ == "\_\_main\_\_":

    uvicorn.run(app, *host*="0.0.0.0", *port*=8000)

import cv2

import numpy as np

import matplotlib.pyplot as plt

def generate\_freeman\_chain\_code(*contour*):

    """

    Menghasilkan Kode Rantai Freeman 8-arah dari kontur OpenCV.

    ASUMSI: kontur didapat dari findContours dengan CHAIN\_APPROX\_NONE.

    """

    chain\_code = []

    if len(contour) < 2:

        return chain\_code # Kontur harus punya minimal 2 titik

    # Pemetaan (dx, dy) ke kode arah Freeman (sumbu Y positif ke bawah)

    directions = {

        (1, 0): 0, (1, 1): 1, (0, 1): 2, (-1, 1): 3,

        (-1, 0): 4, (-1, -1): 5, (0, -1): 6, (1, -1): 7

    }

    for i in range(len(contour)):

        p1 = contour[i][0] # Titik saat ini (format: [[x, y]])

        # Dapatkan titik berikutnya, gunakan modulo % untuk kembali ke titik awal

        # pada iterasi terakhir (menangani kontur tertutup).

        p2 = contour[(i + 1) % len(contour)][0]

        dx = p2[0] - p1[0] # Perbedaan X

        dy = p2[1] - p1[1] # Perbedaan Y (Ingat: Y positif ke bawah)

        norm\_dx = np.sign(dx)

        norm\_dy = np.sign(dy)

        code = directions.get((norm\_dx, norm\_dy))

        if code is not None:

            chain\_code.append(code)

    return chain\_code

# --- Alur Proses Utama ---

# 1. Pemuatan Citra (langsung grayscale)

# Ganti dengan path citra Anda yang valid

img\_path = 'hurufA.png' # <--- GANTI PATH INI

try:

    img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

    if img is None:

        # Coba path relatif jika absolut gagal (atau sebaliknya)

        # Jika tetap gagal, lempar error

        raise *FileNotFoundError*(f"Citra tidak ditemukan atau tidak dapat dibaca di: {img\_path}")

except *Exception* as e:

    print(f"Error saat memuat citra: {e}")

    print("Pastikan path citra sudah benar dan file citra tidak rusak.")

    exit()

# 2. Binarisasi (Sesuaikan threshold & type berdasarkan citra Anda)

threshold\_value = 127

# Asumsi objek gelap di latar terang (misal huruf hitam di kertas putih)

\_, binary\_img = cv2.threshold(img, threshold\_value, 255, cv2.THRESH\_BINARY\_INV)

# Jika objek terang di latar gelap, gunakan:

# \_, binary\_img = cv2.threshold(img, threshold\_value, 255, cv2.THRESH\_BINARY)

# 3. Deteksi Kontur (Wajib CHAIN\_APPROX\_NONE)

contours, hierarchy = cv2.findContours(binary\_img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

# --- Persiapan Visualisasi dengan Matplotlib ---

fig, axs = plt.subplots(2, 2, *figsize*=(10, 8)) # Buat grid 2x2

# Plot Citra Asli (Grayscale)

axs[0, 0].imshow(img, *cmap*='gray')

axs[0, 0].set\_title('Citra Asli (Grayscale)')

axs[0, 0].axis('off') # Sembunyikan sumbu

# Plot Citra Biner

axs[0, 1].imshow(binary\_img, *cmap*='gray')

axs[0, 1].set\_title('Citra Biner (Hasil Threshold)')

axs[0, 1].axis('off')

# Variabel untuk menyimpan hasil jika kontur ditemukan

chain\_code\_str = "Tidak ada kontur ditemukan."

img\_contour\_display = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR) # Default display (jika tak ada kontur)

# 4. Proses Kontur dan Generasi Kode Rantai (jika kontur ada)

if contours:

    # Pilih kontur terbesar

    largest\_contour = max(contours, *key*=cv2.contourArea)

    # Gambar kontur pada citra BGR untuk visualisasi warna

    # Buat salinan BGR dari citra asli

    img\_contour\_display = cv2.cvtColor(img, cv2.COLOR\_GRAY2BGR)

    cv2.drawContours(img\_contour\_display, [largest\_contour], -1, (0, 255, 0), 1) # Gambar kontur hijau

    # Generasi Kode Rantai

    chain\_code\_result = generate\_freeman\_chain\_code(largest\_contour)

    chain\_code\_list\_str = ', '.join(map(*str*, chain\_code\_result))

    # Format teks kode rantai untuk ditampilkan (dengan wrapping sederhana)

    max\_line\_len = 70 # Maks karakter per baris

    wrapped\_code = ""

    current\_line\_len = 0

    for i, code\_str in enumerate(map(*str*, chain\_code\_result)):

        item = code\_str + (", " if i < len(chain\_code\_result) - 1 else "")

        if current\_line\_len + len(item) > max\_line\_len:

            wrapped\_code += "\n"

            current\_line\_len = 0

        wrapped\_code += item

        current\_line\_len += len(item)

    chain\_code\_str = (

        f"Jumlah Kontur Total: {len(contours)}\n"

        f"Kode Rantai Kontur Terbesar (Panjang {len(chain\_code\_result)}):\n"

        f"{wrapped\_code}"

    )

    # Print ke konsol juga (opsional)

    # print(f"Jumlah Kontur Ditemukan: {len(contours)}")

    # print(f"Kode Rantai Kontur Terbesar (Panjang {len(chain\_code\_result)}):")

    # print(chain\_code\_result) # Print list mentah

# Plot Citra dengan Kontur Terdeteksi

# Konversi BGR (OpenCV) ke RGB (Matplotlib) sebelum display

img\_rgb\_display = cv2.cvtColor(img\_contour\_display, cv2.COLOR\_BGR2RGB)

axs[1, 0].imshow(img\_rgb\_display)

axs[1, 0].set\_title('Kontur Terbesar Terdeteksi')

axs[1, 0].axis('off')

# Area untuk menampilkan Teks Kode Rantai

axs[1, 1].axis('off') # Sembunyikan sumbu plot teks

# Tampilkan teks di area plot keempat

axs[1, 1].text(0.05, 0.95, # Posisi x, y (0-1 relatif thd axes)

               chain\_code\_str, # Teks yang ditampilkan

*ha*='left', # Horizontal alignment

*va*='top', # Vertical alignment

*fontsize*=9, # Ukuran font

*wrap*=True) # Coba wrap otomatis (tergantung backend matplotlib)

axs[1, 1].set\_title('Hasil Kode Rantai')

# --- Tampilkan Figure Matplotlib ---

plt.tight\_layout(*pad*=1.5) # Atur layout agar tidak tumpang tindih

plt.suptitle("Analisis Kode Rantai", *fontsize*=16) # Judul keseluruhan

plt.subplots\_adjust(*top*=0.92) # Beri ruang untuk suptitle

plt.show()

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 1. Pemuatan Citra

img\_path = 'cameraman.png'

img = cv2.imread(img\_path)

if img is None:

    raise *FileNotFoundError*(f"Citra tidak ditemukan di: {img\_path}")

# 2. Konversi ke Grayscale

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# 3. Reduksi Noise (Langkah pra-pemrosesan penting!)

# Eksperimen dengan ukuran kernel (ksize) dan sigma

blurred = cv2.GaussianBlur(gray, (5, 5), 0) # Kernel 5x5 umum digunakan

# 4. Deteksi Tepi Canny

# Nilai threshold SANGAT bergantung pada citra. Perlu eksperimen!

low\_threshold = 50

high\_threshold = 150 # Aturan umum: high sekitar 2x-3x low

edges = cv2.Canny(blurred, low\_threshold, high\_threshold)

# 5. Visualisasi Hasil

plt.figure(*figsize*=(12, 6))

plt.subplot(1, 3, 1)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)) # Tampilkan citra asli (konversi warna BGR->RGB)

plt.title('Citra Asli')

plt.axis('off')

plt.subplot(1, 3, 2)

plt.imshow(blurred, *cmap*='gray') # Tampilkan hasil blur

plt.title('Grayscale + Gaussian Blur')

plt.axis('off')

plt.subplot(1, 3, 3)

plt.imshow(edges, *cmap*='gray') # Tampilkan peta tepi hasil Canny

plt.title(f'Tepi Canny (Th={low\_threshold},{high\_threshold})')

plt.axis('off')

plt.tight\_layout()

plt.show()

import cv2

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

# --- 1. Buat Contoh Citra Biner dengan Teks ---

# Ukuran citra

height, width = 150, 400

# Buat canvas hitam

binary\_img = np.zeros((height, width), *dtype*=np.uint8)

# Teks yang akan ditulis

text\_lines = ["Baris Teks Satu", "Ini Baris Dua", "Testing 123"]

start\_y = 40  # Posisi Y awal untuk baris pertama

line\_height = 40 # Jarak vertikal antar baris

font = cv2.FONT\_HERSHEY\_SIMPLEX

font\_scale = 0.8

font\_color = 255  # Putih

thickness = 2

# Tulis teks ke citra

y = start\_y

for line in text\_lines:

    # Dapatkan ukuran teks untuk centering (opsional)

    # (text\_width, text\_height), \_ = cv2.getTextSize(line, font, font\_scale, thickness)

    # x = (width - text\_width) // 2 # Center X

    x = 20 # Mulai dari kiri

    cv2.putText(binary\_img, line, (x, y), font, font\_scale, font\_color, thickness)

    y += line\_height

# --- 2. Hitung Proyeksi Integral ---

# Normalisasi citra ke 0 (latar) dan 1 (objek/teks)

# Ini penting agar hasil sum langsung merepresentasikan jumlah piksel

binary\_norm = binary\_img / 255.0

# Proyeksi Horizontal (jumlah per kolom -> Profil Vertikal)

# axis=0: menjumlahkan sepanjang dimensi baris (secara vertikal)

horizontal\_projection = np.sum(binary\_norm, *axis*=0)

# Proyeksi Vertikal (jumlah per baris -> Profil Horizontal)

# axis=1: menjumlahkan sepanjang dimensi kolom (secara horizontal)

vertical\_projection = np.sum(binary\_norm, *axis*=1)

# --- 3. Buat Plot Menggunakan Matplotlib dan GridSpec ---

# Buat figure dan axes dengan GridSpec untuk kontrol layout

fig = plt.figure(*figsize*=(10, 7)) # Sesuaikan ukuran figure jika perlu

# Grid 2x2, atur rasio tinggi & lebar, dan spasi

# Rasio tinggi: baris atas (proyeksi H) lebih pendek dari baris bawah (gambar)

# Rasio lebar: kolom kiri (gambar) lebih lebar dari kolom kanan (proyeksi V)

gs = fig.add\_gridspec(2, 2, *width\_ratios*=(4, 1), *height\_ratios*=(1, 4),

*left*=0.1, *right*=0.9, *bottom*=0.1, *top*=0.9,

*wspace*=0.05, *hspace*=0.05)

# Axes untuk citra biner (pojok kiri bawah - indeks [1, 0])

ax\_img = fig.add\_subplot(gs[1, 0])

ax\_img.imshow(binary\_img, *cmap*='gray', *aspect*='auto') # aspect='auto' agar tidak terdistorsi

ax\_img.set\_title('Contoh Citra Biner')

ax\_img.set\_xlabel('Indeks Kolom')

ax\_img.set\_ylabel('Indeks Baris (0 di atas)')

# Axes untuk Proyeksi Horizontal (di atas citra biner - indeks [0, 0])

# Bagikan sumbu X dengan plot citra agar sejajar

ax\_hproj = fig.add\_subplot(gs[0, 0], *sharex*=ax\_img)

ax\_hproj.plot(np.arange(width), horizontal\_projection, *color*='blue')

ax\_hproj.set\_title('Proyeksi Horizontal (Profil Vertikal)')

ax\_hproj.set\_ylabel('Jumlah Piksel Putih')

# Sembunyikan label tick X karena sudah ada di plot bawah (citra)

plt.setp(ax\_hproj.get\_xticklabels(), *visible*=False)

ax\_hproj.grid(*axis*='y', *linestyle*='--', *alpha*=0.6) # Grid bantu

# Axes untuk Proyeksi Vertikal (di kanan citra biner - indeks [1, 1])

# Bagikan sumbu Y dengan plot citra agar sejajar

ax\_vproj = fig.add\_subplot(gs[1, 1], *sharey*=ax\_img)

# Perhatikan: plot(nilai\_proyeksi, indeks\_baris)

ax\_vproj.plot(vertical\_projection, np.arange(height), *color*='red')

ax\_vproj.set\_title('Proyeksi Vertikal')

ax\_vproj.set\_xlabel('Jumlah Piksel Putih')

# Invert sumbu Y agar 0 ada di atas, cocok dengan citra

ax\_vproj.invert\_yaxis()

# Sembunyikan label tick Y karena sudah ada di plot kiri (citra)

plt.setp(ax\_vproj.get\_yticklabels(), *visible*=False)

ax\_vproj.grid(*axis*='x', *linestyle*='--', *alpha*=0.6) # Grid bantu

# Judul keseluruhan

plt.suptitle("Visualisasi Proyeksi Integral pada Citra Teks", *fontsize*=14)

# Tampilkan plot

plt.show()

# (Opsional) Simpan citra biner jika diperlukan

# cv2.imwrite("contoh\_teks\_biner.png", binary\_img)

import cv2

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

# --- 1. Buat Contoh Citra Biner dengan Teks ---

# Ukuran citra

height, width = 150, 400

# Buat canvas hitam

binary\_img = np.zeros((height, width), *dtype*=np.uint8)

# Teks yang akan ditulis

text\_lines = ["Baris Teks Satu", "Ini Baris Dua", "Testing 123"]

start\_y = 40  # Posisi Y awal untuk baris pertama

line\_height = 40 # Jarak vertikal antar baris

font = cv2.FONT\_HERSHEY\_SIMPLEX

font\_scale = 0.8

font\_color = 255  # Putih

thickness = 2

# Tulis teks ke citra

y = start\_y

for line in text\_lines:

    # Dapatkan ukuran teks untuk centering (opsional)

    # (text\_width, text\_height), \_ = cv2.getTextSize(line, font, font\_scale, thickness)

    # x = (width - text\_width) // 2 # Center X

    x = 20 # Mulai dari kiri

    cv2.putText(binary\_img, line, (x, y), font, font\_scale, font\_color, thickness)

    y += line\_height

# --- 2. Hitung Proyeksi Integral ---

# Normalisasi citra ke 0 (latar) dan 1 (objek/teks)

# Ini penting agar hasil sum langsung merepresentasikan jumlah piksel

binary\_norm = binary\_img / 255.0

# Proyeksi Horizontal (jumlah per kolom -> Profil Vertikal)

# axis=0: menjumlahkan sepanjang dimensi baris (secara vertikal)

horizontal\_projection = np.sum(binary\_norm, *axis*=0)

# Proyeksi Vertikal (jumlah per baris -> Profil Horizontal)

# axis=1: menjumlahkan sepanjang dimensi kolom (secara horizontal)

vertical\_projection = np.sum(binary\_norm, *axis*=1)

# --- 3. Buat Plot Menggunakan Matplotlib dan GridSpec ---

# Buat figure dan axes dengan GridSpec untuk kontrol layout

fig = plt.figure(*figsize*=(10, 7)) # Sesuaikan ukuran figure jika perlu

# Grid 2x2, atur rasio tinggi & lebar, dan spasi

# Rasio tinggi: baris atas (proyeksi H) lebih pendek dari baris bawah (gambar)

# Rasio lebar: kolom kiri (gambar) lebih lebar dari kolom kanan (proyeksi V)

gs = fig.add\_gridspec(2, 2, *width\_ratios*=(4, 1), *height\_ratios*=(1, 4),

*left*=0.1, *right*=0.9, *bottom*=0.1, *top*=0.9,

*wspace*=0.05, *hspace*=0.05)

# Axes untuk citra biner (pojok kiri bawah - indeks [1, 0])

ax\_img = fig.add\_subplot(gs[1, 0])

ax\_img.imshow(binary\_img, *cmap*='gray', *aspect*='auto') # aspect='auto' agar tidak terdistorsi

ax\_img.set\_title('Contoh Citra Biner')

ax\_img.set\_xlabel('Indeks Kolom')

ax\_img.set\_ylabel('Indeks Baris (0 di atas)')

# Axes untuk Proyeksi Horizontal (di atas citra biner - indeks [0, 0])

# Bagikan sumbu X dengan plot citra agar sejajar

ax\_hproj = fig.add\_subplot(gs[0, 0], *sharex*=ax\_img)

ax\_hproj.plot(np.arange(width), horizontal\_projection, *color*='blue')

ax\_hproj.set\_title('Proyeksi Horizontal (Profil Vertikal)')

ax\_hproj.set\_ylabel('Jumlah Piksel Putih')

# Sembunyikan label tick X karena sudah ada di plot bawah (citra)

plt.setp(ax\_hproj.get\_xticklabels(), *visible*=False)

ax\_hproj.grid(*axis*='y', *linestyle*='--', *alpha*=0.6) # Grid bantu

# Axes untuk Proyeksi Vertikal (di kanan citra biner - indeks [1, 1])

# Bagikan sumbu Y dengan plot citra agar sejajar

ax\_vproj = fig.add\_subplot(gs[1, 1], *sharey*=ax\_img)

# Perhatikan: plot(nilai\_proyeksi, indeks\_baris)

ax\_vproj.plot(vertical\_projection, np.arange(height), *color*='red')

ax\_vproj.set\_title('Proyeksi Vertikal')

ax\_vproj.set\_xlabel('Jumlah Piksel Putih')

# Invert sumbu Y agar 0 ada di atas, cocok dengan citra

ax\_vproj.invert\_yaxis()

# Sembunyikan label tick Y karena sudah ada di plot kiri (citra)

plt.setp(ax\_vproj.get\_yticklabels(), *visible*=False)

ax\_vproj.grid(*axis*='x', *linestyle*='--', *alpha*=0.6) # Grid bantu

# Judul keseluruhan

plt.suptitle("Visualisasi Proyeksi Integral pada Citra Teks", *fontsize*=14)

# Tampilkan plot

plt.show()

# (Opsional) Simpan citra biner jika diperlukan

# cv2.imwrite("contoh\_teks\_biner.png", binary\_img)

import cv2

import numpy as np

import matplotlib.pyplot as plt

# 1. Pemuatan Citra (langsung grayscale)

img\_path = 'tier1.png' # Ganti dgn path citra teks/objek

img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

if img is None:

    raise FileNotFoundError(f"Citra tidak ditemukan di: {img\_path}")

# 2. Binarisasi (KRUSIAL: Objek harus PUTIH/NON-NOL, Latar HITAM/NOL)

# Gunakan Otsu untuk otomatisasi jika kontras baik

# Jika teks hitam di latar putih, gunakan THRESH\_BINARY\_INV

\_, binary\_img = cv2.threshold(img, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

# Jika perlu, pastikan objek = 255, latar = 0. Jika terbalik: binary\_img = 255 - binary\_img

# Normalisasi ke 0 dan 1 (Objek=1, Latar=0) untuk interpretasi mudah

binary\_norm = binary\_img / 255.0

# 3. Proyeksi Horizontal (Sum per Kolom -> Profil Vertikal)

# axis=0: menjumlahkan sepanjang dimensi baris (secara vertikal)

horizontal\_projection = np.sum(binary\_norm, *axis*=0)

# 4. Proyeksi Vertikal (Sum per Baris -> Profil Horizontal)

# axis=1: menjumlahkan sepanjang dimensi kolom (secara horizontal)

vertical\_projection = np.sum(binary\_norm, *axis*=1)

# 5. Visualisasi Hasil (Layout ditingkatkan)

height, width = binary\_norm.shape

# Buat figure dan axes dengan GridSpec untuk kontrol layout lebih baik

fig = plt.figure(*figsize*=(10, 8))

gs = fig.add\_gridspec(2, 2, *width\_ratios*=(4, 1), *height\_ratios*=(1, 4),

*left*=0.1, *right*=0.9, *bottom*=0.1, *top*=0.9,

*wspace*=0.05, *hspace*=0.05)

# Axes untuk citra biner (pojok kiri bawah)

ax\_img = fig.add\_subplot(gs[1, 0])

ax\_img.imshow(binary\_norm, *cmap*='gray')

ax\_img.set\_title('Citra Biner (Objek=1)')

ax\_img.set\_xlabel('Indeks Kolom')

ax\_img.set\_ylabel('Indeks Baris')

# Axes untuk Proyeksi Horizontal (di atas citra biner)

ax\_hproj = fig.add\_subplot(gs[0, 0], *sharex*=ax\_img) # Bagikan sumbu X

ax\_hproj.plot(np.arange(width), horizontal\_projection)

ax\_hproj.set\_title('Proyeksi Horizontal (Profil Vertikal)')

ax\_hproj.set\_ylabel('Jumlah Piksel')

plt.setp(ax\_hproj.get\_xticklabels(), *visible*=False) # Sembunyikan label X

# Axes untuk Proyeksi Vertikal (di kanan citra biner)

ax\_vproj = fig.add\_subplot(gs[1, 1], *sharey*=ax\_img) # Bagikan sumbu Y

ax\_vproj.plot(vertical\_projection, np.arange(height))

ax\_vproj.set\_title('Proyeksi Vertikal')

ax\_vproj.set\_xlabel('Jumlah Piksel')

ax\_vproj.invert\_yaxis() # Cocokkan orientasi Y citra

plt.setp(ax\_vproj.get\_yticklabels(), *visible*=False) # Sembunyikan label Y

plt.suptitle("Analisis Proyeksi Integral", *fontsize*=14)

plt.show()

import streamlit as st

import cv2

import os

import numpy as np

from skimage.metrics import structural\_similarity as ssim

import matplotlib.pyplot as plt

import pandas as pd

import subprocess

from pathlib import Path

from io import BytesIO

from PIL import Image

import uuid

OUTPUT\_DIR = 'output'

TEMP\_DIR = os.path.join(OUTPUT\_DIR, 'temp')

CSV\_PATH = os.path.join(OUTPUT\_DIR, 'results.csv')

JPEG\_QUALITIES = [95, 75, 50, 25, 10]

PNG\_COMPRESSION\_LEVELS = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Path(OUTPUT\_DIR).mkdir(*parents*=True, *exist\_ok*=True)

Path(TEMP\_DIR).mkdir(*parents*=True, *exist\_ok*=True)

def calculate\_metrics(*img\_original*, *img\_compressed*, *is\_color*, *win\_size*):

    if img\_original.shape != img\_compressed.shape:

        raise *ValueError*("Dimensi citra berbeda!")

    psnr\_value = cv2.PSNR(img\_original, img\_compressed)

    mse = np.mean((img\_original.astype(*float*) - img\_compressed.astype(*float*)) \*\* 2)

    psnr\_manual = *float*('inf') if mse == 0 else 20 \* np.log10(255.0 / np.sqrt(mse))

    try:

        if is\_color:

            ssim\_value = ssim(

                img\_original, img\_compressed,

*channel\_axis*=2, *win\_size*=win\_size,

*data\_range*=img\_original.max() - img\_original.min()

            )

        else:

            ssim\_value = ssim(

                img\_original, img\_compressed,

*win\_size*=win\_size,

*data\_range*=img\_original.max() - img\_original.min()

            )

    except *ValueError* as e:

        st.warning(f"Error calculating SSIM: {e}")

        ssim\_value = None

    is\_identical = np.array\_equal(img\_original, img\_compressed)

    return psnr\_value, psnr\_manual, ssim\_value, is\_identical

def append\_to\_csv(*results*, *csv\_path*):

    df\_new = pd.DataFrame(results)

    try:

        if os.path.exists(csv\_path) and os.stat(csv\_path).st\_size > 0:

            df\_existing = pd.read\_csv(csv\_path)

        else:

            df\_existing = pd.DataFrame()

    except pd.errors.EmptyDataError:

        df\_existing = pd.DataFrame()

    df\_combined = pd.concat([df\_existing, df\_new], *ignore\_index*=True)

    df\_combined.to\_csv(csv\_path, *index*=False)

def process\_image(*uploaded\_file*):

    results = []

    image\_name = uploaded\_file.name

    st.write(f"Memproses citra: {image\_name}")

    img\_bytes = uploaded\_file.read()

    img\_pil = Image.open(BytesIO(img\_bytes))

    img\_original\_bgr = np.array(img\_pil)

    if img\_original\_bgr.shape[-1] == 4:

        img\_original\_bgr = cv2.cvtColor(img\_original\_bgr, cv2.COLOR\_RGBA2BGR)

    elif img\_original\_bgr.shape[-1] == 3:

        img\_original\_bgr = cv2.cvtColor(img\_original\_bgr, cv2.COLOR\_RGB2BGR)

    if len(img\_original\_bgr.shape) == 3:

        is\_color = True

        img\_original\_cv = cv2.cvtColor(img\_original\_bgr, cv2.COLOR\_BGR2RGB)

    else:

        is\_color = False

        img\_original\_cv = img\_original\_bgr

        st.info("Citra grayscale dimuat.")

    temp\_path = os.path.join(TEMP\_DIR, f"original\_{uuid.uuid4().hex}.png")

    cv2.imwrite(temp\_path, img\_original\_bgr)

    original\_size\_bytes = os.path.getsize(temp\_path)

    st.write(f"Ukuran file asli: {original\_size\_bytes / 1024:.2f} KB")

    min\_dim = min(img\_original\_cv.shape[:2])

    win\_size = min(7, min\_dim if min\_dim % 2 == 1 else min\_dim - 1)

    if win\_size < 3:

        win\_size = 3

    st.write(f"SSIM win\_size: {win\_size}")

    for quality in JPEG\_QUALITIES:

        jpeg\_path = os.path.join(TEMP\_DIR, f"{uuid.uuid4().hex}\_jpeg\_q{quality}.jpg")

        img\_to\_save = cv2.cvtColor(img\_original\_cv, cv2.COLOR\_RGB2BGR) if is\_color else img\_original\_cv

        cv2.imwrite(jpeg\_path, img\_to\_save, [cv2.IMWRITE\_JPEG\_QUALITY, quality])

        if is\_color:

            img\_compressed\_bgr = cv2.imread(jpeg\_path)

            img\_compressed\_cv = cv2.cvtColor(img\_compressed\_bgr, cv2.COLOR\_BGR2RGB)

        else:

            img\_compressed\_cv = cv2.imread(jpeg\_path, cv2.IMREAD\_GRAYSCALE)

        if img\_compressed\_cv is None:

            st.error(f"Gagal memuat citra terkompresi {jpeg\_path}")

            continue

        try:

            psnr\_value, \_, ssim\_value, is\_identical = calculate\_metrics(

                img\_original\_cv, img\_compressed\_cv, is\_color, win\_size

            )

        except *ValueError* as e:

            st.error(f"Error processing JPEG Q{quality}: {e}")

            continue

        compressed\_size\_bytes = os.path.getsize(jpeg\_path)

        results.append({

            'Citra Input': image\_name,

            'Metode Kompresi': 'JPEG',

            'Kualitas/Level': quality,

            'Ukuran File (KB)': compressed\_size\_bytes / 1024,

            'Rasio Kompresi': original\_size\_bytes / compressed\_size\_bytes if compressed\_size\_bytes > 0 else *float*('inf'),

            'PSNR (dB)': psnr\_value,

            'SSIM': ssim\_value,

            'Identik?': is\_identical

        })

    for level in PNG\_COMPRESSION\_LEVELS:

        png\_path = os.path.join(TEMP\_DIR, f"{uuid.uuid4().hex}\_compressed\_level{level}.png")

        img\_to\_save = cv2.cvtColor(img\_original\_cv, cv2.COLOR\_RGB2BGR) if is\_color else img\_original\_cv

        cv2.imwrite(png\_path, img\_to\_save, [cv2.IMWRITE\_PNG\_COMPRESSION, level])

        if is\_color:

            img\_png\_bgr = cv2.imread(png\_path)

            img\_png\_cv = cv2.cvtColor(img\_png\_bgr, cv2.COLOR\_BGR2RGB)

        else:

            img\_png\_cv = cv2.imread(png\_path, cv2.IMREAD\_GRAYSCALE)

        if img\_png\_cv is None:

            st.error(f"Gagal memuat citra terkompresi {png\_path}")

            continue

        try:

            psnr\_value, \_, ssim\_value, is\_identical = calculate\_metrics(

                img\_original\_cv, img\_png\_cv, is\_color, win\_size

            )

        except *ValueError* as e:

            st.error(f"Error processing PNG Level {level}: {e}")

            continue

        try:

            subprocess.run(['optipng', '-o7', png\_path], *check*=True, *capture\_output*=True)

            png\_size\_bytes = os.path.getsize(png\_path)

        except (subprocess.CalledProcessError, *FileNotFoundError*):

            png\_size\_bytes = os.path.getsize(png\_path)

        results.append({

            'Citra Input': image\_name,

            'Metode Kompresi': f'PNG (Level {level})',

            'Kualitas/Level': 'Lossless',

            'Ukuran File (KB)': png\_size\_bytes / 1024,

            'Rasio Kompresi': original\_size\_bytes / png\_size\_bytes if png\_size\_bytes > 0 else *float*('inf'),

            'PSNR (dB)': 'Infinity' if psnr\_value == *float*('inf') else psnr\_value,

            'SSIM': ssim\_value,

            'Identik?': is\_identical

        })

    append\_to\_csv(results, CSV\_PATH)

    return results, img\_original\_cv, is\_color

def plot\_metrics(*df*):

    st.subheader("Grafik Analisis")

    metrics = ['Ukuran File (KB)', 'Rasio Kompresi', 'PSNR (dB)', 'SSIM']

    for metric in metrics:

        fig, ax = plt.subplots()

        for method in df['Metode Kompresi'].unique():

            subset = df[df['Metode Kompresi'] == method]

            if metric == 'PSNR (dB)':

                subset = subset[subset['PSNR (dB)'] != 'Infinity']

                subset['PSNR (dB)'] = subset['PSNR (dB)'].astype(*float*)

            ax.plot(subset['Kualitas/Level'], subset[metric], *marker*='o', *label*=method)

        ax.set\_xlabel('Kualitas/Level')

        ax.set\_ylabel(metric)

        ax.set\_title(f'{metric} vs Kualitas/Level')

        ax.legend()

        ax.grid(True)

        st.pyplot(fig)

st.title("Eksperimen Kompresi Citra")

st.write("Unggah citra untuk melakukan kompresi JPEG dan PNG, lalu lihat hasilnya dalam tabel dan grafik.")

uploaded\_file = st.file\_uploader("Pilih citra", *type*=['png', 'jpg', 'jpeg', 'tif'])

if uploaded\_file is not None:

    with st.spinner("Memproses citra..."):

        results, img\_original\_cv, is\_color = process\_image(uploaded\_file)

    st.subheader("Tabel Hasil")

    df\_results = pd.DataFrame(results)

    df\_results["Kualitas/Level"] = df\_results["Kualitas/Level"].astype(*str*)

    st.dataframe(df\_results)

    plot\_metrics(df\_results)

    st.subheader("Isi File CSV (Semua Hasil)")

    if os.path.exists(CSV\_PATH):

        try:

            df\_csv = pd.read\_csv(CSV\_PATH)

            df\_csv["Kualitas/Level"] = df\_csv["Kualitas/Level"].astype(*str*)

            st.dataframe(df\_csv)

        except pd.errors.EmptyDataError:

            st.write("Belum ada data di CSV.")

    else:

        st.write("Belum ada data di CSV.")

    with open(CSV\_PATH, 'rb') as f:

        st.download\_button("Unduh Results CSV", f, *file\_name*="results.csv")

for f in os.listdir(TEMP\_DIR):

    os.remove(os.path.join(TEMP\_DIR, f))

import cv2

import os

import numpy as np

from skimage.metrics import structural\_similarity as ssim

import matplotlib.pyplot as plt

# --- Persiapan ---

image\_path\_original = 'lena.png' # Ganti path citra Anda

# Muat citra menggunakan OpenCV

img\_original\_bgr = cv2.imread(image\_path\_original)

# Muat sebagai grayscale jika menggunakan Cameraman.tif

# img\_original\_gray = cv2.imread('cameraman.tif', cv2.IMREAD\_GRAYSCALE)

# img\_original\_cv = img\_original\_gray # Gunakan variabel ini untuk grayscale

if img\_original\_bgr is None:

    print(f"Error: Tidak dapat memuat citra dari {image\_path\_original}")

    exit()

# Tentukan apakah citra berwarna atau grayscale

if len(img\_original\_bgr.shape) == 3:

    is\_color = True

    # Konversi BGR ke RGB untuk pemrosesan/tampilan internal

    img\_original\_rgb = cv2.cvtColor(img\_original\_bgr, cv2.COLOR\_BGR2RGB)

    img\_original\_cv = img\_original\_rgb # Gunakan variabel ini untuk citra berwarna

else:

    is\_color = False

    img\_original\_cv = img\_original\_bgr # Langsung gunakan jika sudah grayscale

    print("Citra grayscale dimuat.")

original\_size\_bytes = os.path.getsize(image\_path\_original)

print(f"Ukuran file asli ({image\_path\_original}): {original\_size\_bytes / 1024:.2f} KB")

jpeg\_qualities = [95, 75, 50, 25, 10]

results = []

# Tentukan win\_size untuk SSIM (harus ganjil dan lebih kecil dari dimensi citra)

min\_dim = min(img\_original\_cv.shape[:2])

win\_size = min(7, min\_dim if min\_dim % 2 == 1 else min\_dim - 1)

if win\_size < 3:

    win\_size = 3  # Minimum win\_size yang valid

print(f"SSIM win\_size: {win\_size}")

for quality in jpeg\_qualities:

    jpeg\_path = f'lena\_jpeg\_q{quality}.jpg'

    # Simpan citra dengan kualitas JPEG tertentu

    if is\_color:

        img\_to\_save = cv2.cvtColor(img\_original\_cv, cv2.COLOR\_RGB2BGR)

    else:

        img\_to\_save = img\_original\_cv

    cv2.imwrite(jpeg\_path, img\_to\_save, [cv2.IMWRITE\_JPEG\_QUALITY, quality])

    compressed\_size\_bytes = os.path.getsize(jpeg\_path)

    # Muat kembali citra terkompresi

    img\_compressed\_bgr = cv2.imread(jpeg\_path)

    if img\_compressed\_bgr is None:

        print(f"Error: Tidak dapat memuat citra terkompresi {jpeg\_path}")

        continue

    # Konversi ke format yang sesuai

    if is\_color:

        img\_compressed\_cv = cv2.cvtColor(img\_compressed\_bgr, cv2.COLOR\_BGR2RGB)

    else:

        img\_compressed\_cv = cv2.imread(jpeg\_path, cv2.IMREAD\_GRAYSCALE)

        if img\_compressed\_cv is None:

            print(f"Error: Gagal memuat ulang {jpeg\_path} sebagai grayscale.")

            continue

    # Pastikan dimensi sama

    if img\_original\_cv.shape != img\_compressed\_cv.shape:

        print(f"Warning: Dimensi citra berbeda setelah kompresi JPEG Q{quality}!")

        continue

    # Hitung PSNR

    psnr\_value = cv2.PSNR(img\_original\_cv, img\_compressed\_cv)

    # Hitung SSIM

    try:

        if is\_color:

            ssim\_value = ssim(

                img\_original\_cv,

                img\_compressed\_cv,

*channel\_axis*=2,  # Eksplisitly set channel axis for RGB

*win\_size*=win\_size,

*data\_range*=img\_original\_cv.max() - img\_original\_cv.min()

            )

        else:

            ssim\_value = ssim(

                img\_original\_cv,

                img\_compressed\_cv,

*win\_size*=win\_size,

*data\_range*=img\_original\_cv.max() - img\_original\_cv.min()

            )

    except *ValueError* as e:

        print(f"Error calculating SSIM for Q{quality}: {e}")

        ssim\_value = None

    results.append({

        'Method': 'JPEG',

        'Quality': quality,

        'FileSize (KB)': compressed\_size\_bytes / 1024,

        'CompressionRatio': original\_size\_bytes / compressed\_size\_bytes if compressed\_size\_bytes > 0 else *float*('inf'),

        'PSNR (dB)': psnr\_value,

        'SSIM': ssim\_value

    })

    ssim\_display = f"{ssim\_value:.4f}" if ssim\_value is not None else "N/A"

    print(f"JPEG Q{quality}: Size={compressed\_size\_bytes / 1024:.2f} KB, PSNR={psnr\_value:.2f} dB, SSIM={ssim\_display}")

    # Contoh menampilkan perbandingan (pastikan citra yang ditampilkan sudah dalam format RGB atau Grayscale)

img\_q95\_bgr = cv2.imread('lena\_jpeg\_q95.jpg') # Muat hasil

img\_q10\_bgr = cv2.imread('lena\_jpeg\_q10.jpg')

if is\_color:

    img\_q95\_vis = cv2.cvtColor(img\_q95\_bgr, cv2.COLOR\_BGR2RGB)

    img\_q10\_vis = cv2.cvtColor(img\_q10\_bgr, cv2.COLOR\_BGR2RGB)

    cmap\_val = None # Default cmap untuk color

else:

    # Muat ulang sebagai gray jika aslinya gray

    img\_q95\_vis = cv2.imread('lena\_jpeg\_q95.jpg', cv2.IMREAD\_GRAYSCALE)

    img\_q10\_vis = cv2.imread('lena\_jpeg\_q10.jpg', cv2.IMREAD\_GRAYSCALE)

    cmap\_val = 'gray' # cmap untuk grayscale

fig, axes = plt.subplots(1, 3, *figsize*=(15, 5))

# Tampilkan img\_original\_cv (yang sudah RGB atau Grayscale)

axes[0].imshow(img\_original\_cv, *cmap*=cmap\_val)

axes[0].set\_title(f'Original ({original\_size\_bytes / 1024:.2f} KB)')

axes[0].axis('off')

if img\_q95\_vis is not None:

    q95\_size = os.path.getsize('lena\_jpeg\_q95.jpg')

    axes[1].imshow(img\_q95\_vis, *cmap*=cmap\_val)

    axes[1].set\_title(f'JPEG Q95 ({q95\_size / 1024:.2f} KB)')

    axes[1].axis('off')

else:

     axes[1].set\_title('JPEG Q95 (Error Loading)')

     axes[1].axis('off')

if img\_q10\_vis is not None:

    q10\_size = os.path.getsize('lena\_jpeg\_q10.jpg')

    axes[2].imshow(img\_q10\_vis, *cmap*=cmap\_val)

    axes[2].set\_title(f'JPEG Q10 ({q10\_size / 1024:.2f} KB)')

    axes[2].axis('off')

else:

     axes[2].set\_title('JPEG Q10 (Error Loading)')

     axes[2].axis('off')

plt.tight\_layout()

plt.show()

import cv2

import os

import numpy as np

from skimage.metrics import structural\_similarity as ssim

import matplotlib.pyplot as plt

import pandas as pd

import subprocess

# --- Persiapan ---

image\_path\_original = 'lena.png'  # Ganti path citra Anda

img\_original\_bgr = cv2.imread(image\_path\_original)

if img\_original\_bgr is None:

    print(f"Error: Tidak dapat memuat citra dari {image\_path\_original}")

    exit()

# Tentukan apakah citra berwarna atau grayscale

if len(img\_original\_bgr.shape) == 3:

    is\_color = True

    img\_original\_rgb = cv2.cvtColor(img\_original\_bgr, cv2.COLOR\_BGR2RGB)

    img\_original\_cv = img\_original\_rgb

else:

    is\_color = False

    img\_original\_cv = img\_original\_bgr

    print("Citra grayscale dimuat.")

original\_size\_bytes = os.path.getsize(image\_path\_original)

print(f"Ukuran file asli ({image\_path\_original}): {original\_size\_bytes / 1024:.2f} KB")

print(f"Original shape: {img\_original\_cv.shape}, dtype: {img\_original\_cv.dtype}")

results = []

# Tentukan win\_size untuk SSIM

min\_dim = min(img\_original\_cv.shape[:2])

win\_size = min(7, min\_dim if min\_dim % 2 == 1 else min\_dim - 1)

if win\_size < 3:

    win\_size = 3

print(f"SSIM win\_size: {win\_size}")

# --- PNG Compression untuk semua level ---

png\_compression\_levels = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for level in png\_compression\_levels:

    png\_path = f'lena\_compressed\_level{level}.png'

    # Siapkan citra untuk disimpan

    if is\_color:

        img\_to\_save\_png = cv2.cvtColor(img\_original\_cv, cv2.COLOR\_RGB2BGR)

    else:

        img\_to\_save\_png = img\_original\_cv

    # Simpan citra sebagai PNG

    cv2.imwrite(png\_path, img\_to\_save\_png, [cv2.IMWRITE\_PNG\_COMPRESSION, level])

    png\_size\_bytes = os.path.getsize(png\_path)

    # Optimize dengan optipng (opsional)

    try:

        subprocess.run(['optipng', '-o7', png\_path], *check*=True)

        png\_size\_bytes\_opt = os.path.getsize(png\_path)

    except (subprocess.CalledProcessError, *FileNotFoundError*):

        print(f"optipng not found or failed for level {level}; using original size.")

        png\_size\_bytes\_opt = png\_size\_bytes

    # Muat kembali citra terkompresi

    img\_png\_compressed\_bgr = cv2.imread(png\_path)

    if img\_png\_compressed\_bgr is None:

        print(f"Error: Tidak dapat memuat citra terkompresi {png\_path}")

        continue

    if is\_color:

        img\_png\_compressed\_cv = cv2.cvtColor(img\_png\_compressed\_bgr, cv2.COLOR\_BGR2RGB)

    else:

        img\_png\_compressed\_cv = cv2.imread(png\_path, cv2.IMREAD\_GRAYSCALE)

        if img\_png\_compressed\_cv is None:

            print(f"Error: Gagal memuat ulang {png\_path} sebagai grayscale.")

            continue

    print(f"Level {level} compressed shape: {img\_png\_compressed\_cv.shape}, dtype: {img\_png\_compressed\_cv.dtype}")

    # Verifikasi Lossless

    psnr\_png = cv2.PSNR(img\_original\_cv, img\_png\_compressed\_cv)

    # Manual PSNR

    mse = np.mean((img\_original\_cv.astype(*float*) - img\_png\_compressed\_cv.astype(*float*)) \*\* 2)

    psnr\_manual = *float*('inf') if mse == 0 else 20 \* np.log10(255.0 / np.sqrt(mse))

    # Hitung SSIM

    try:

        if is\_color:

            ssim\_png = ssim(

                img\_original\_cv,

                img\_png\_compressed\_cv,

*channel\_axis*=2,

*win\_size*=win\_size,

*data\_range*=img\_original\_cv.max() - img\_original\_cv.min()

            )

        else:

            ssim\_png = ssim(

                img\_original\_cv,

                img\_png\_compressed\_cv,

*win\_size*=win\_size,

*data\_range*=img\_original\_cv.max() - img\_original\_cv.min()

            )

    except *ValueError* as e:

        print(f"Error calculating SSIM for Level {level}: {e}")

        ssim\_png = None

    is\_identical = np.array\_equal(img\_original\_cv, img\_png\_compressed\_cv)

    results.append({

        'Method': f'PNG (Level {level})',

        'Quality': 'Lossless',

        'FileSize (KB)': png\_size\_bytes / 1024,

        'FileSize Opt (KB)': png\_size\_bytes\_opt / 1024,

        'CompressionRatio': original\_size\_bytes / png\_size\_bytes if png\_size\_bytes > 0 else *float*('inf'),

        'PSNR (dB)': psnr\_png if psnr\_png != *float*('inf') else 'Infinity',

        'PSNR Manual': psnr\_manual if psnr\_manual != *float*('inf') else 'Infinity',

        'SSIM': ssim\_png,

        'Identical': is\_identical

    })

    # Print hasil per level

    ssim\_display = f"{ssim\_png:.4f}" if ssim\_png is not None else "N/A"

    psnr\_display = f"{psnr\_png:.2f}" if psnr\_png != *float*('inf') else "Infinity"

    print(f"PNG (Level {level}): Size={png\_size\_bytes / 1024:.2f} KB, Opt Size={png\_size\_bytes\_opt / 1024:.2f} KB, PSNR={psnr\_display}, PSNR Manual={psnr\_manual}, SSIM={ssim\_display}, Identical={is\_identical}")

# --- Visualisasi (hanya untuk Level 9) ---

png\_path\_vis = 'lena\_compressed\_level9.png'

img\_png\_vis = cv2.imread(png\_path\_vis)

if is\_color:

    img\_png\_vis = cv2.cvtColor(img\_png\_vis, cv2.COLOR\_BGR2RGB)

    cmap\_val = None

else:

    img\_png\_vis = cv2.imread(png\_path\_vis, cv2.IMREAD\_GRAYSCALE)

    cmap\_val = 'gray'

fig, axes = plt.subplots(1, 2, *figsize*=(10, 5))

axes[0].imshow(img\_original\_cv, *cmap*=cmap\_val)

axes[0].set\_title(f'Original ({original\_size\_bytes / 1024:.2f} KB)')

axes[0].axis('off')

if img\_png\_vis is not None:

    png\_size\_vis = os.path.getsize(png\_path\_vis)

    axes[1].imshow(img\_png\_vis, *cmap*=cmap\_val)

    axes[1].set\_title(f'PNG Level 9 ({png\_size\_vis / 1024:.2f} KB)')

    axes[1].axis('off')

else:

    axes[1].set\_title('PNG Level 9 (Error Loading)')

    axes[1].axis('off')

plt.tight\_layout()

plt.savefig('compression\_comparison.png')

plt.show()

# --- Tabel Hasil ---

df\_results = pd.DataFrame(results)

print("\n--- Hasil Kompresi ---")

print(df\_results.to\_string())

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import graycomatrix, graycoprops

from skimage.feature import local\_binary\_pattern

from skimage import io, util

# =============================================================================

# Definisi Konstanta

# =============================================================================

KOMPONEN\_Y\_LUMINANSI = 'Komponen Y (Luminansi)'

# =============================================================================

# Fungsi Utility

# =============================================================================

def display\_image(*image*, *title*, *cmap*=None, *figsize*=(8, 6)):

    plt.figure(*figsize*=figsize)

    plt.imshow(image, *cmap*=cmap)

    plt.title(title)

    plt.axis('off')

    plt.tight\_layout()

    plt.show()

def display\_multiple(*images*, *titles*, *cmaps*=None, *figsize*=(12, 8)):

    n = len(images)

    rows = *int*(np.ceil(n / 3))

    cols = min(n, 3)

    plt.figure(*figsize*=figsize)

    for i in range(n):

        plt.subplot(rows, cols, i+1)

        if cmaps is None:

            cmap = None

        elif isinstance(cmaps, *list*):

            cmap = cmaps[i] if i < len(cmaps) else None

        else:

            cmap = cmaps

        plt.imshow(images[i], *cmap*=cmap)

        plt.title(titles[i])

        plt.axis('off')

    plt.tight\_layout()

    plt.show()

# =============================================================================

# Proses

# =============================================================================

def process\_rgb(*image\_rgb*):

    display\_image(image\_rgb, 'Citra RGB Asli')

    r, g, b = cv2.split(image\_rgb)

    display\_multiple([r, g, b], ['Kanal Red', 'Kanal Green', 'Kanal Blue'], *cmaps*='gray')

def process\_xyz(*image\_rgb*):

    image\_xyz = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2XYZ)

    display\_image(image\_xyz, 'Citra dalam Ruang Warna XYZ')

    x, y\_xyz, z = cv2.split(image\_xyz)

    display\_multiple([x, y\_xyz, z], ['Komponen X', KOMPONEN\_Y\_LUMINANSI, 'Komponen Z'], *cmaps*='gray')

def process\_lab(*image\_rgb*):

    image\_lab = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2Lab)

    display\_image(image\_lab, 'Citra dalam Ruang Warna Lab')

    l, a, b = cv2.split(image\_lab)

    display\_multiple([l, a, b], ['Komponen L (Luminansi)', 'Komponen a (Hijau-Merah)', 'Komponen b (Biru-Kuning)'], *cmaps*='gray')

def process\_ycbcr(*image\_rgb*):

    image\_ycbcr = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2YCrCb)

    display\_image(image\_ycbcr, 'Citra dalam Ruang Warna YCbCr')

    y\_ycbcr, cr, cb = cv2.split(image\_ycbcr)

    display\_multiple([y\_ycbcr, cb, cr], [KOMPONEN\_Y\_LUMINANSI, 'Komponen Cb', 'Komponen Cr'], *cmaps*='gray')

def process\_hsv(*image\_rgb*):

    image\_hsv = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2HSV)

    display\_image(image\_hsv, 'Citra dalam Ruang Warna HSV')

    h, s, v = cv2.split(image\_hsv)

    display\_multiple([h, s, v], ['Komponen Hue', 'Komponen Saturation', 'Komponen Value'], *cmaps*='gray')

def rgb\_to\_yiq(*rgb*):

    rgb\_norm = rgb.astype(np.float32) / 255.0

    transform\_matrix = np.array([

        [0.299, 0.587, 0.114],

        [0.596, -0.274, -0.322],

        [0.211, -0.523, 0.312]

    ])

    height, width, \_ = rgb.shape

    rgb\_reshaped = rgb\_norm.reshape(height \* width, 3)

    yiq\_reshaped = np.dot(rgb\_reshaped, transform\_matrix.T)

    yiq = yiq\_reshaped.reshape(height, width, 3)

    return yiq

def process\_yiq(*image\_rgb*):

    image\_yiq = rgb\_to\_yiq(image\_rgb)

    display\_image(image\_yiq, 'Citra dalam Ruang Warna YIQ')

    y\_yiq = image\_yiq[:, :, 0]

    i = image\_yiq[:, :, 1]

    q = image\_yiq[:, :, 2]

    display\_multiple([y\_yiq, i, q], [KOMPONEN\_Y\_LUMINANSI, 'Komponen I (In-phase)', 'Komponen Q (Quadrature)'], *cmaps*='gray')

def process\_luminance\_analysis(*image\_rgb*):

    image\_lab = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2Lab)

    l, \_, \_ = cv2.split(image\_lab)

    image\_ycbcr = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2YCrCb)

    y\_ycbcr, \_, \_ = cv2.split(image\_ycbcr)

    image\_hsv = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2HSV)

    \_, \_, v = cv2.split(image\_hsv)

    image\_yiq = rgb\_to\_yiq(image\_rgb)

    y\_yiq = image\_yiq[:, :, 0]

    luminance\_components = {

        'Y dari YCbCr': y\_ycbcr,

        'L dari Lab': l,

        'Y dari YIQ': (y\_yiq \* 255),

        'V dari HSV': v

    }

    plt.figure(*figsize*=(12, 8))

    i = 1

    for name, component in luminance\_components.items():

        plt.subplot(2, 2, i)

        plt.imshow(component, *cmap*='gray')

        plt.title(name)

        plt.axis('off')

        i += 1

    plt.tight\_layout()

    plt.show()

# =============================================================================

# Menu Interaktif

# =============================================================================

def show\_menu():

    print("Pilih opsi analisis:")

    print("1. Tampilkan RGB dan Kanal")

    print("2. Konversi ke XYZ")

    print("3. Konversi ke Lab")

    print("4. Konversi ke YCbCr")

    print("5. Konversi ke HSV")

    print("6. Konversi ke YIQ")

    print("7. Analisis Luminansi")

    print("0. Keluar")

if \_\_name\_\_ == '\_\_main\_\_':

    image\_path = 'sample\_image.jpg'  # Ganti sesuai path citra Anda

    image = cv2.imread(image\_path)

    if image is None:

        raise *FileNotFoundError*(f"Citra tidak ditemukan di path: {image\_path}")

    image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

    while True:

        show\_menu()

        choice = input("Masukkan pilihan (0-7): ")

        if choice == '1':

            process\_rgb(image\_rgb)

        elif choice == '2':

            process\_xyz(image\_rgb)

        elif choice == '3':

            process\_lab(image\_rgb)

        elif choice == '4':

            process\_ycbcr(image\_rgb)

        elif choice == '5':

            process\_hsv(image\_rgb)

        elif choice == '6':

            process\_yiq(image\_rgb)

        elif choice == '7':

            process\_luminance\_analysis(image\_rgb)

        elif choice == '0':

            print("Keluar dari program.")

            break

        else:

            print("Pilihan tidak valid. Silakan coba lagi.")

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import graycomatrix, graycoprops, local\_binary\_pattern

from skimage import io, util

# --- Persiapan Data ---

image\_path = "sample\_image.jpg"

if not os.path.exists(image\_path):

    raise *FileNotFoundError*(f"File citra tidak ditemukan: {image\_path}")

# Baca dan konversi

image = cv2.imread(image\_path)

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Kanal RGB

R, G, B = cv2.split(image\_rgb)

# Ruang warna XYZ

image\_xyz = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2XYZ)

X, Y\_xyz, Z = cv2.split(image\_xyz)

# Ruang warna Lab

image\_lab = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2Lab)

L, a, b = cv2.split(image\_lab)

# Ruang warna YCbCr

image\_ycbcr = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2YCrCb)

Y\_ycbcr, Cr, Cb = cv2.split(image\_ycbcr)

# Ruang warna HSV

image\_hsv = cv2.cvtColor(image\_rgb, cv2.COLOR\_RGB2HSV)

H, S, V = cv2.split(image\_hsv)

# Konversi manual RGB ke YIQ

def rgb\_to\_yiq(*rgb*):

    rgb\_norm = rgb.astype(np.float32) / 255.0

    T = np.array([[0.299,  0.587,  0.114],

                  [0.596, -0.274, -0.322],

                  [0.211, -0.523,  0.312]], *dtype*=np.float32)

    h, w, \_ = rgb\_norm.shape

    flat = rgb\_norm.reshape(-1, 3)

    yiq\_flat = flat.dot(T.T)

    return yiq\_flat.reshape(h, w, 3)

image\_yiq = rgb\_to\_yiq(image\_rgb)

Y\_yiq = (image\_yiq[:, :, 0] \* 255).astype(np.uint8)

I = image\_yiq[:, :, 1]

Q = image\_yiq[:, :, 2]

# Histogram luminansi

luminance\_components = {

    'Histogram YCbCr (Y)': Y\_ycbcr,

    'Histogram Lab (L)': L,

    'Histogram YIQ (Y)': Y\_yiq,

    'Histogram HSV (V)': V

}

# Segmentasi warna merah (HSV)

lower\_red1 = np.array([0, 100, 100]); upper\_red1 = np.array([10, 255, 255])

lower\_red2 = np.array([160, 100, 100]); upper\_red2 = np.array([180, 255, 255])

mask1 = cv2.inRange(image\_hsv, lower\_red1, upper\_red1)

mask2 = cv2.inRange(image\_hsv, lower\_red2, upper\_red2)

mask\_red = mask1 | mask2

segmented\_red = cv2.bitwise\_and(image\_rgb, image\_rgb, *mask*=mask\_red)

# --- Dictionary Output ---

outputs = {

    1: ('RGB Asli',           lambda: plt.imshow(image\_rgb)),

    2: ('Kanal R',            lambda: plt.imshow(R, *cmap*='Reds')),

    3: ('Kanal G',            lambda: plt.imshow(G, *cmap*='Greens')),

    4: ('Kanal B',            lambda: plt.imshow(B, *cmap*='Blues')),

    5: ('XYZ',                lambda: plt.imshow(image\_xyz)),

    6: ('Lab',                lambda: plt.imshow(image\_lab)),

    7: ('YCbCr',              lambda: plt.imshow(image\_ycbcr)),

    8: ('HSV',                lambda: plt.imshow(image\_hsv)),

    9: ('YIQ',                lambda: plt.imshow(Y\_yiq, *cmap*='gray')),

   10: ('Segmentasi Merah',   lambda: plt.imshow(segmented\_red)),

}

# Tambahkan histogram ke dict (mulai index 11)

idx = 11

for name, comp in luminance\_components.items():

    outputs[idx] = (

        name,

        lambda *comp*=comp: plt.hist(comp.ravel(), *bins*=256, *range*=(0,255), *alpha*=0.7)

    )

    idx += 1

def show\_menu():

    print("\n=== MENU OUTPUT ===")

    for k, (name, \_) in outputs.items():

        print(f"{k:2d}. {name}")

    print(" 0. Keluar")

def display\_choice(*choice*):

    name, func = outputs[choice]

    plt.figure(*figsize*=(6,6))

    func()

    plt.title(name)

    # Untuk histogram, tambahkan label

    if name.startswith('Histogram'):

        plt.xlabel('Intensitas')

        plt.ylabel('Frekuensi')

    else:

        plt.axis('off')

    plt.tight\_layout()

    plt.show()

def main():

    while True:

        show\_menu()

        try:

            sel = *int*(input("Pilih nomor output: "))

        except *ValueError*:

            print("Masukkan angka yang valid.")

            continue

        if sel == 0:

            print("Selesai.")

            break

        if sel not in outputs:

            print("Pilihan tidak tersedia.")

            continue

        display\_choice(sel)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import graycomatrix, graycoprops, local\_binary\_pattern

from skimage import util

# =============================================================================

# Definisi Konstanta

# =============================================================================

KOMPONEN\_Y\_LUMINANSI = 'Komponen Y (Luminansi)'

# =============================================================================

# Fungsi Utility

# =============================================================================

def display\_image(*image*, *title*, *cmap*=None, *figsize*=(8, 6)):

    plt.figure(*figsize*=*figsize*)

    plt.imshow(*image*, *cmap*=*cmap*)

    plt.title(*title*)

    plt.axis('off')

    plt.tight\_layout()

    plt.show()

def display\_multiple(*images*, *titles*, *cmaps*=None, *figsize*=(12, 8)):

    n = len(*images*)

    rows = int(np.ceil(n / 3))

    cols = min(n, 3)

    plt.figure(*figsize*=*figsize*)

    for i, img in enumerate(*images*):

        plt.subplot(rows, cols, i+1)

        if *cmaps* is None:

            cmap = None

        elif isinstance(*cmaps*, list):

            cmap = *cmaps*[i] if i < len(*cmaps*) else None

        else:

            cmap = *cmaps*

        plt.imshow(img, *cmap*=cmap)

        plt.title(*titles*[i])

        plt.axis('off')

    plt.tight\_layout()

    plt.show()

# =============================================================================

# Fungsi Analisis Tekstur

# =============================================================================

def compute\_texture\_statistics(*image*, *window\_size*=15):

    # Pastikan grayscale

    if *image*.ndim == 3:

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

*image* = *image*.astype(np.float32)

    mean\_map = cv2.boxFilter(*image*, -1, (*window\_size*, *window\_size*))

    mean\_sq\_map = cv2.boxFilter(*image* \* *image*, -1, (*window\_size*, *window\_size*))

    var\_map = np.maximum(mean\_sq\_map - mean\_map\*\*2, 0)

    std\_map = np.sqrt(var\_map)

    norm\_maps = {}

    for name, m in [('mean', mean\_map), ('variance', var\_map), ('std\_dev', std\_map)]:

        norm\_maps[name] = cv2.normalize(m, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

    return norm\_maps

def compute\_glcm\_features(*image*, *distances*, *angles*, *levels*=256):

    # Pastikan grayscale uint8

    if *image*.ndim == 3:

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

*image* = util.img\_as\_ubyte(*image*)

    glcm = graycomatrix(*image*, *distances*=*distances*, *angles*=*angles*,

*levels*=*levels*, *symmetric*=True, *normed*=True)

    props = ['contrast', 'dissimilarity', 'homogeneity', 'energy', 'correlation', 'ASM']

    features = {p: graycoprops(glcm, p) for p in props}

    return features, glcm

def compute\_lbp(*image*, *radius*=3, *n\_points*=8\*3, *method*='uniform'):

    if *image*.ndim == 3:

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

    lbp = local\_binary\_pattern(*image*, *n\_points*, *radius*, *method*)

    bins = int(lbp.max() + 1)

    hist, \_ = np.histogram(lbp.ravel(), *bins*=bins, *range*=(0, bins), *density*=True)

    return lbp, hist

def compute\_gabor\_filters(*image*, *frequencies*, *orientations*):

    if *image*.ndim == 3:

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

    image\_float = *image*.astype(np.float32)

    filtered\_imgs, titles = [], []

    magnitude = np.zeros\_like(image\_float)

    for freq in *frequencies*:

        for theta in *orientations*:

            ksize = int(2 \* np.ceil(freq) + 1)

            if ksize % 2 == 0:

                ksize += 1

            kernel = cv2.getGaborKernel((ksize, ksize), *sigma*=freq/3,

*theta*=theta, *lambd*=freq,

*gamma*=0.5, *psi*=0)

            filtered = cv2.filter2D(image\_float, cv2.CV\_32F, kernel)

            magnitude += filtered\*\*2

            norm = cv2.normalize(filtered, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

            filtered\_imgs.append(norm)

            titles.append(f'Gabor f={freq:.1f}, θ={theta:.2f}')

    mag = np.sqrt(magnitude)

    mag\_norm = cv2.normalize(mag, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

    filtered\_imgs.append(mag\_norm)

    titles.append('Gabor Magnitude')

    return filtered\_imgs, titles

def compute\_laws\_texture(*image*):

    if *image*.ndim == 3:

*image* = cv2.cvtColor(*image*, cv2.COLOR\_BGR2GRAY)

    imgf = *image*.astype(np.float32)

    L5 = np.array([1, 4, 6, 4, 1])

    E5 = np.array([-1, -2, 0, 2, 1])

    S5 = np.array([-1, 0, 2, 0, -1])

    R5 = np.array([1, -4, 6, -4, 1])

    W5 = np.array([-1, 2, 0, -2, 1])

    filters = {'L5': L5, 'E5': E5, 'S5': S5, 'R5': R5, 'W5': W5}

    texture\_maps = {}

    for n1, f1 in filters.items():

        for n2, f2 in filters.items():

            name = n1 + n2

            kernel = np.outer(f1, f2)

            filtered = cv2.filter2D(imgf, -1, kernel)

            energy = cv2.boxFilter(np.abs(filtered), -1, (15, 15), *normalize*=True)

            norm = cv2.normalize(energy, None, 0, 255, cv2.NORM\_MINMAX).astype(np.uint8)

            texture\_maps[name] = norm

    selected = ['L5E5','E5S5','S5S5','R5R5','L5S5','E5E5']

    imgs = [texture\_maps[n] for n in selected]

    titles = [f'Law {n}' for n in selected]

    return imgs, titles

# =============================================================================

# Menu Interaktif

# =============================================================================

def show\_menu():

    menu = [

        '1. Statistik Tekstur (Mean, Var, StdDev)',

        '2. Fitur GLCM',

        '3. Fitur LBP',

        '4. Respons Filter Gabor',

        "5. Law's Texture Energy",

        "0. Keluar"

    ]

    print('\n'.join(menu))

if \_\_name\_\_ == '\_\_main\_\_':

    image\_path = 'texture\_sample.jpg'

    img\_bgr = cv2.imread(image\_path)

    if img\_bgr is None:

        raise FileNotFoundError(f"Gagal membaca {image\_path}")

    img\_rgb = cv2.cvtColor(img\_bgr, cv2.COLOR\_BGR2RGB)

    img\_gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_RGB2GRAY)

    while True:

        show\_menu()

        choice = input('Pilih menu (0-5): ')

        if choice == '1':

            stats = compute\_texture\_statistics(img\_gray)

            display\_multiple(

                [stats['mean'], stats['variance'], stats['std\_dev']],

                ['Rerata Lokal','Variansi Lokal','Deviasi Standar'],

*cmaps*='jet'

            )

        elif choice == '2':

            feats, glcm = compute\_glcm\_features(img\_gray, [1,3,5], [0,np.pi/4,np.pi/2,3\*np.pi/4])

            print('GLCM Features (d=1,θ=0):')

            for k,v in feats.items():

                print(f'  {k}: {v[0,0]:.4f}')

            plt.figure(*figsize*=(6,5))

            plt.imshow(glcm[:,:,0,0], *cmap*='viridis')

            plt.colorbar()

            plt.title('GLCM (d=1,θ=0)')

            plt.show()

        elif choice == '3':

            lbp\_img, lbp\_hist = compute\_lbp(img\_gray)

            display\_multiple([lbp\_img], ['Peta LBP'], *cmaps*='jet')

            plt.figure()

            plt.bar(range(len(lbp\_hist)), lbp\_hist)

            plt.title('Histogram LBP')

            plt.show()

        elif choice == '4':

            imgs, titles = compute\_gabor\_filters(img\_gray, [5,10,15], [0, np.pi/4, np.pi/2, 3\*np.pi/4])

            display\_multiple(imgs[:6]+[imgs[-1]], titles[:6]+['Magnitude'], *cmaps*='jet')

        elif choice == '5':

            imgs, titles = compute\_laws\_texture(img\_gray)

            display\_multiple(imgs, titles, *cmaps*='jet')

        elif choice == '0':

            print('Keluar.')

            break

        else:

            print('Pilihan tidak valid.')

import cv2

import mediapipe as mp

import tkinter as tk

from tkinter import simpledialog, messagebox

from PIL import Image, ImageTk

import json

import os

import datetime

# Directory to save gesture images and JSON

GESTURE\_DIR = 'gestures'

GESTURE\_JSON = os.path.join(GESTURE\_DIR, 'gestures.json')

if not os.path.exists(GESTURE\_DIR):

    os.makedirs(GESTURE\_DIR)

# Load or initialize gesture database

if os.path.exists(GESTURE\_JSON):

    with open(GESTURE\_JSON, 'r') as f:

        gesture\_db = json.load(f)

else:

    gesture\_db = []

# MediaPipe setup

mp\_hands = mp.solutions.hands

mp\_drawing = mp.solutions.drawing\_utils

hands = mp\_hands.Hands(*static\_image\_mode*=False, *max\_num\_hands*=1, *min\_detection\_confidence*=0.7, *min\_tracking\_confidence*=0.5)

class GestureCaptureApp:

    def \_\_init\_\_(*self*, *master*):

*self*.master = master

*self*.master.title('Gesture Capture GUI')

*self*.cap = cv2.VideoCapture(0)

*self*.frame = None

*self*.landmarks = None

*self*.imgtk = None

*self*.mp\_drawing\_styles = mp.solutions.drawing\_styles

*self*.panel = tk.Label(master)

*self*.panel.pack()

        # Dropdown for command selection

        from modules.drone\_simulator import DroneSimulator

*self*.gesture\_labels = DroneSimulator().gesture\_labels

        # Mapping: command -> gesture

*self*.gesture\_command\_mapping = {

            'TAKEOFF': 'OPEN\_PALM',

            'LAND': 'FIST',

            'FORWARD': 'INDEX\_UP',

            'BACKWARD': 'INDEX\_DOWN',

            'LEFT': 'INDEX\_LEFT',

            'RIGHT': 'INDEX\_RIGHT',

            'UP': 'THUMB\_UP',

            'DOWN': 'THUMB\_DOWN',

            'ROTATE CW': 'VICTORY\_UP',

            'ROTATE CCW': 'VICTORY\_LEFT',

            'NONE': 'NONE'

        }

*self*.command\_var = tk.StringVar(master)

        gesture\_names = [

            f"{cmd} ({*self*.gesture\_command\_mapping.get(cmd, '-')})"

            for cmd in *self*.gesture\_labels.values()

        ]

*self*.command\_var.set(gesture\_names[0])

*self*.gesture\_colors = {

            'TAKEOFF': '#43a047',   # Green

            'LAND': '#e53935',      # Red

            'FORWARD': '#1e88e5',   # Blue

            'BACKWARD': '#8e24aa',  # Purple

            'LEFT': '#fbc02d',      # Yellow

            'RIGHT': '#fb8c00',     # Orange

            'UP': '#00acc1',        # Cyan

            'DOWN': '#6d4c41',      # Brown

            'ROTATE CW': '#3949ab', # Indigo

            'ROTATE CCW': '#00897b',# Teal

            'NONE': '#757575'       # Grey

        }

*self*.command\_dropdown = tk.OptionMenu(master, *self*.command\_var, \*gesture\_names, *command*=*self*.update\_status\_color)

*self*.command\_dropdown.pack()

*self*.capture\_btn = tk.Button(master, *text*='Capture & Save Gesture', *command*=*self*.capture\_gesture, *bg*='#43a047', *fg*='white', *font*=("Arial", 11, "bold"))

*self*.capture\_btn.pack(*pady*=3)

*self*.multi\_capture\_btn = tk.Button(master, *text*='Capture & Save 10x', *command*=*self*.capture\_multiple\_gestures, *bg*='#1e88e5', *fg*='white', *font*=("Arial", 11, "bold"))

*self*.multi\_capture\_btn.pack(*pady*=3)

        # Removed 'Save to JSON' button (redundant)

*self*.status = tk.Label(master, *text*='', *font*=("Arial", 12, "bold"))

*self*.status.pack()

*self*.update\_status\_color(*self*.command\_var.get())

*self*.update\_frame()

    def update\_frame(*self*):

        ret, frame = *self*.cap.read()

        if not ret:

*self*.status.config(*text*='Camera error!')

            return

*self*.frame = frame.copy()

        rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        results = hands.process(rgb)

*self*.landmarks = None

        if results.multi\_hand\_landmarks:

            for hand\_landmarks in results.multi\_hand\_landmarks:

                mp\_drawing.draw\_landmarks(

                    frame,

                    hand\_landmarks,

                    mp\_hands.HAND\_CONNECTIONS,

*self*.mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),

*self*.mp\_drawing\_styles.get\_default\_hand\_connections\_style()

                )

                # Convert landmarks to list of dicts

*self*.landmarks = [

                    {'x': lm.x, 'y': lm.y, 'z': lm.z}

                    for lm in hand\_landmarks.landmark

                ]

        img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        img = Image.fromarray(img)

*self*.imgtk = ImageTk.PhotoImage(*image*=img)

*self*.panel.imgtk = *self*.imgtk

*self*.panel.config(*image*=*self*.imgtk)

*self*.master.after(10, *self*.update\_frame)

    def capture\_gesture(*self*):

        if *self*.landmarks is None:

            messagebox.showerror('Error', 'No hand detected!')

            return

        command\_full = *self*.command\_var.get().strip()

        command = command\_full.split(' (')[0]

        if not command:

            messagebox.showerror('Error', 'Please select command!')

            return

        timestamp = datetime.datetime.now().strftime('%Y%m%d\_%H%M%S')

        img\_filename = f'{command}\_{timestamp}.png'

        img\_path = os.path.join(GESTURE\_DIR, img\_filename)

        # Save cropped hand image (focus on hand)

        hand\_img = *self*.crop\_hand(*self*.frame, *self*.landmarks)

        cv2.imwrite(img\_path, hand\_img)

        gesture\_entry = {

            'name': command,

            'landmarks': *self*.landmarks,

            'image\_path': img\_path

        }

        gesture\_db.append(gesture\_entry)

        with open(GESTURE\_JSON, 'w') as f:

            json.dump(gesture\_db, f, *indent*=2)

*self*.update\_status\_color(command)

*self*.status.config(*text*=f'Gesture captured & saved: {command}')

    def crop\_hand(*self*, *frame*, *landmarks*):

        # Calculate bounding box from landmarks

        h, w, \_ = frame.shape

        xs = [*int*(lm['x'] \* w) for lm in landmarks]

        ys = [*int*(lm['y'] \* h) for lm in landmarks]

        x\_min, x\_max = max(min(xs)-20, 0), min(max(xs)+20, w)

        y\_min, y\_max = max(min(ys)-20, 0), min(max(ys)+20, h)

        hand\_img = frame[y\_min:y\_max, x\_min:x\_max].copy()

        return hand\_img

    # save\_gesture is now obsolete (handled in capture\_gesture)

    def capture\_multiple\_gestures(*self*):

        import time

        count = 0

        target = 10

        max\_total\_retry = 40 \* target

        total\_retry = 0

        while count < target and total\_retry < max\_total\_retry:

            retry = 0

            found = False

            while retry < 30:

                ret, frame = *self*.cap.read()

                if not ret:

*self*.status.config(*text*='Camera error!')

*self*.master.update\_idletasks()

                    time.sleep(0.2)

                    retry += 1

                    total\_retry += 1

                    continue

                rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

                results = hands.process(rgb)

                if results.multi\_hand\_landmarks:

                    for hand\_landmarks in results.multi\_hand\_landmarks:

                        landmarks = [

                            {'x': lm.x, 'y': lm.y, 'z': lm.z}

                            for lm in hand\_landmarks.landmark

                        ]

                        # Save gesture

                        command\_full = *self*.command\_var.get().strip()

                        command = command\_full.split(' (')[0]

                        timestamp = datetime.datetime.now().strftime('%Y%m%d\_%H%M%S')

                        img\_filename = f'{command}\_{timestamp}\_{count+1}.png'

                        img\_path = os.path.join(GESTURE\_DIR, img\_filename)

                        hand\_img = *self*.crop\_hand(frame, landmarks)

                        cv2.imwrite(img\_path, hand\_img)

                        gesture\_entry = {

                            'name': command,

                            'landmarks': landmarks,

                            'image\_path': img\_path

                        }

                        gesture\_db.append(gesture\_entry)

                        with open(GESTURE\_JSON, 'w') as f:

                            json.dump(gesture\_db, f, *indent*=2)

                        count += 1

*self*.status.config(*text*=f'Captured {count}/{target}')

*self*.master.update\_idletasks()

                        time.sleep(0.33)

                        found = True

                        break

                if found:

                    break

                else:

*self*.status.config(*text*=f'Hand not detected! Retry {retry+1}/30 (sample {count+1}/{target})')

*self*.master.update\_idletasks()

                    time.sleep(0.13)

                    retry += 1

                    total\_retry += 1

            if not found:

*self*.status.config(*text*=f'Skipped (no hand) {count+1}/{target}')

*self*.master.update\_idletasks()

                time.sleep(0.2)

        if count < target:

*self*.status.config(*text*=f'Capture stopped: only {count}/{target} captured (hand not detected enough)')

        else:

*self*.status.config(*text*=f'Selesai capture {count}/{target} data!')

    def update\_status\_color(*self*, *gesture\_name*):

        command = gesture\_name.split(' (')[0]

        color = *self*.gesture\_colors.get(command, '#757575')

*self*.status.config(*bg*=color, *fg*='white')

    def \_\_del\_\_(*self*):

        if *self*.cap.isOpened():

*self*.cap.release()

if \_\_name\_\_ == '\_\_main\_\_':

    root = tk.Tk()

    app = GestureCaptureApp(root)

    root.mainloop()

import json

import os

import numpy as np

from sklearn.model\_selection import StratifiedKFold

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import joblib

import tkinter as tk

from tkinter import filedialog, messagebox, scrolledtext

from threading import Thread

class GestureClassifierApp:

    def \_\_init\_\_(*self*, *root*):

*self*.root = *root*

*self*.root.title("Gesture Classifier Trainer with Cross-Validation")

*self*.root.geometry("600x400")

        # Label

        tk.Label(*root*, *text*="Gesture Classifier Trainer (Cross-Validation)", *font*=("Arial", 14)).pack(*pady*=10)

        # Tombol untuk memilih file

        tk.Button(*root*, *text*="Pilih File JSON", *command*=*self*.select\_file).pack(*pady*=5)

        # Tombol untuk memilih folder

        tk.Button(*root*, *text*="Pilih Folder JSON", *command*=*self*.select\_folder).pack(*pady*=5)

        # Area teks untuk menampilkan output

*self*.output\_text = scrolledtext.ScrolledText(*root*, *height*=15, *width*=70, *wrap*=tk.WORD)

*self*.output\_text.pack(*pady*=10)

        # Tombol untuk memulai pelatihan

*self*.train\_button = tk.Button(*root*, *text*="Mulai Pelatihan", *command*=*self*.start\_training, *state*=tk.DISABLED)

*self*.train\_button.pack(*pady*=5)

        # Variabel untuk menyimpan data

*self*.data = []

*self*.is\_folder = False

*self*.input\_path = ""

    def select\_file(*self*):

        file\_path = filedialog.askopenfilename(*filetypes*=[("JSON files", "\*.json")])

        if file\_path:

*self*.input\_path = file\_path

*self*.is\_folder = False

*self*.train\_button.config(*state*=tk.NORMAL)

*self*.output\_text.delete(1.0, tk.END)

*self*.output\_text.insert(tk.END, f"File dipilih: {file\_path}\n")

*self*.load\_file\_data()

    def select\_folder(*self*):

        folder\_path = filedialog.askdirectory()

        if folder\_path:

*self*.input\_path = folder\_path

*self*.is\_folder = True

*self*.train\_button.config(*state*=tk.NORMAL)

*self*.output\_text.delete(1.0, tk.END)

*self*.output\_text.insert(tk.END, f"Folder dipilih: {folder\_path}\n")

*self*.load\_folder\_data()

    def load\_file\_data(*self*):

        try:

            with open(*self*.input\_path, 'r') as f:

*self*.data = json.load(f)

*self*.output\_text.insert(tk.END, f"Memuat {len(*self*.data)} sampel dari file.\n")

        except FileNotFoundError:

            messagebox.showerror("Kesalahan", "File tidak ditemukan.")

*self*.data = []

        except json.JSONDecodeError:

            messagebox.showerror("Kesalahan", "File JSON tidak valid.")

*self*.data = []

    def load\_folder\_data(*self*):

        try:

            json\_files = [f for f in os.listdir(*self*.input\_path) if f.endswith('.json')]

            if not json\_files:

                messagebox.showerror("Kesalahan", "Tidak ada file JSON di dalam folder.")

*self*.data = []

                return

*self*.data = []

            for json\_file in json\_files:

                with open(os.path.join(*self*.input\_path, json\_file), 'r') as f:

                    file\_data = json.load(f)

*self*.data.extend(file\_data)

*self*.output\_text.insert(tk.END, f"Memuat {len(*self*.data)} sampel dari {len(json\_files)} file.\n")

        except FileNotFoundError:

            messagebox.showerror("Kesalahan", "Folder tidak ditemukan.")

*self*.data = []

        except json.JSONDecodeError:

            messagebox.showerror("Kesalahan", "Ada file JSON yang tidak valid di dalam folder.")

*self*.data = []

    def start\_training(*self*):

        # Nonaktifkan tombol selama pelatihan

*self*.train\_button.config(*state*=tk.DISABLED)

*self*.output\_text.insert(tk.END, "Memulai pelatihan dengan 5-fold cross-validation...\n")

        # Jalankan pelatihan di thread terpisah agar GUI tetap responsif

        Thread(*target*=*self*.train\_model).start()

    def train\_model(*self*):

        # Langkah 1: Ekstrak fitur dan label

        features = []

        labels = []

        if not *self*.data:

*self*.root.after(0, lambda: messagebox.showerror("Kesalahan", "Tidak ada data yang dimuat."))

*self*.root.after(0, lambda: *self*.train\_button.config(*state*=tk.NORMAL))

            return

        expected\_num\_landmarks = len(*self*.data[0]['landmarks'])

        for sample in *self*.data:

            try:

                gesture = sample['name']

                landmarks = sample['landmarks']

                if len(landmarks) != expected\_num\_landmarks:

*self*.output\_text.insert(tk.END, f"Sampel memiliki jumlah landmark yang salah: {len(landmarks)} (diharapkan {expected\_num\_landmarks})\n")

                    continue

                landmark\_coords = []

                for landmark in landmarks:

                    landmark\_coords.extend([landmark['x'], landmark['y'], landmark['z']])

                features.append(landmark\_coords)

                labels.append(gesture)

            except KeyError:

*self*.output\_text.insert(tk.END, "Ada kunci yang hilang di dalam sampel.\n")

                continue

        if not features:

*self*.root.after(0, lambda: messagebox.showerror("Kesalahan", "Tidak ada data valid untuk dilatih."))

*self*.root.after(0, lambda: *self*.train\_button.config(*state*=tk.NORMAL))

            return

        # Konversi ke array numpy

        X = np.array(features)

        y = np.array(labels)

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, f"Bentuk fitur: {X.shape}\nJumlah label: {len(y)}\n"))

        # Langkah 2: Lakukan 5-fold cross-validation

        k = 5

        skf = StratifiedKFold(*n\_splits*=k, *shuffle*=True, *random\_state*=42)

        fold\_accuracies = []

        fold\_reports = []

        for fold, (train\_idx, test\_idx) in enumerate(skf.split(X, y), 1):

            X\_train, X\_test = X[train\_idx], X[test\_idx]

            y\_train, y\_test = y[train\_idx], y[test\_idx]

            # Latih model untuk fold ini

            model = RandomForestClassifier(*n\_estimators*=100, *random\_state*=42)

            model.fit(X\_train, y\_train)

            # Evaluasi model

            y\_pred = model.predict(X\_test)

            accuracy = accuracy\_score(y\_test, y\_pred)

            fold\_accuracies.append(accuracy)

            report = classification\_report(y\_test, y\_pred, *target\_names*=sorted(set(y)), *output\_dict*=True)

            # Simpan laporan untuk fold ini

            fold\_reports.append((fold, accuracy, report))

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, f"Fold {fold} - Akurasi: {accuracy:.2f}\n"))

        # Hitung akurasi rata-rata dan deviasi standar

        mean\_accuracy = np.mean(fold\_accuracies)

        std\_accuracy = np.std(fold\_accuracies)

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END,

            f"\nAkurasi Rata-rata: {mean\_accuracy:.2f} ± {std\_accuracy:.2f}\n"))

        # Tampilkan laporan klasifikasi untuk setiap fold

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, "\nLaporan Klasifikasi per Fold:\n"))

        for fold, accuracy, report in fold\_reports:

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, f"\nFold {fold} (Akurasi: {accuracy:.2f}):\n"))

            for gesture, metrics in report.items():

                if isinstance(metrics, dict):  # Hanya tampilkan metrik per kelas

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END,

                        f"  {gesture}: Precision={metrics['precision']:.2f}, Recall={metrics['recall']:.2f}, F1-Score={metrics['f1-score']:.2f}\n"))

        # Langkah 3: Buat visualisasi akurasi per fold

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, "\nVisualisasi Akurasi per Fold:\n"))

        chart\_config = {

            "type": "bar",

            "data": {

                "labels": [f"Fold {i}" for i in range(1, k + 1)],

                "datasets": [{

                    "label": "Akurasi",

                    "data": fold\_accuracies,

                    "backgroundColor": "#4e73df",

                    "borderColor": "#2e59d9",

                    "borderWidth": 1

                }]

            },

            "options": {

                "scales": {

                    "y": {

                        "beginAtZero": True,

                        "max": 1,

                        "title": {

                            "display": True,

                            "text": "Akurasi"

                        }

                    },

                    "x": {

                        "title": {

                            "display": True,

                            "text": "Fold"

                        }

                    }

                },

                "plugins": {

                    "legend": {

                        "display": True

                    },

                    "title": {

                        "display": True,

                        "text": "Akurasi per Fold dari 5-Fold Cross-Validation"

                    }

                }

            }

        }

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END,

            "\n```chartjs\n" + json.dumps(chart\_config, *indent*=2) + "\n```\n"))

        # Langkah 4: Latih model akhir pada seluruh data

        final\_model = RandomForestClassifier(*n\_estimators*=100, *random\_state*=42)

        final\_model.fit(X, y)

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, "\nMelatih model akhir pada seluruh data...\n"))

        # Langkah 5: Simpan model menggunakan joblib

        joblib.dump(final\_model, 'gesture\_classifier\_cv.pkl')

*self*.root.after(0, lambda: *self*.output\_text.insert(tk.END, "Model akhir disimpan sebagai 'gesture\_classifier\_cv.pkl'\n"))

        # Aktifkan kembali tombol

*self*.root.after(0, lambda: *self*.train\_button.config(*state*=tk.NORMAL))

*self*.root.after(0, lambda: messagebox.showinfo("Sukses", "Pelatihan dengan cross-validation selesai dan model disimpan!"))

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = GestureClassifierApp(root)

    root.mainloop()

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

import math

import logging

# Configure logging

logging.basicConfig(*level*=logging.INFO, *format*='%(asctime)s - %(name)s - %(levelname)s - %(message)s')

logger = logging.getLogger(\_\_name\_\_)

class DroneSimulator:

    """

    Simulates a drone's state and responds to control commands.

    This class maintains the drone's position, rotation, and flying state,

    and provides methods to update the drone state based on gesture commands.

    """

    # Gesture IDs

    GESTURE\_TAKEOFF = 0   # Open palm

    GESTURE\_LAND = 1      # Closed fist

    GESTURE\_FORWARD = 2   # Index finger pointing forward

    GESTURE\_BACKWARD = 3  # Index finger pointing backward

    GESTURE\_LEFT = 4      # Index finger pointing left

    GESTURE\_RIGHT = 5     # Index finger pointing right

    GESTURE\_UP = 6        # Thumb up

    GESTURE\_DOWN = 7      # Thumb down

    GESTURE\_ROTATE\_CW = 8 # Two fingers (index and middle) extended

    GESTURE\_ROTATE\_CCW = 9 # Three fingers (index, middle, ring) extended

    GESTURE\_NONE = -1     # No gesture detected

    # Movement speed constants

    MOVE\_SPEED = 0.2  # Increased units per update

    ROTATE\_SPEED = 0.2  # Increased radians per update

    def \_\_init\_\_(*self*):

        # Drone state (position and rotation)

*self*.position = {"x": 0, "y": 0, "z": 0}

*self*.rotation = {"x": 0, "y": 0, "z": 0}  # Euler angles

*self*.is\_flying = False

*self*.last\_gesture = *self*.GESTURE\_NONE

*self*.gesture\_buffer = []

*self*.buffer\_size = 5  # Number of frames to buffer for gesture stability

*self*.gesture\_labels = {

*self*.GESTURE\_TAKEOFF: "TAKEOFF",

*self*.GESTURE\_LAND: "LAND",

*self*.GESTURE\_FORWARD: "FORWARD",

*self*.GESTURE\_BACKWARD: "BACKWARD",

*self*.GESTURE\_LEFT: "LEFT",

*self*.GESTURE\_RIGHT: "RIGHT",

*self*.GESTURE\_UP: "UP",

            self.GESTURE\_DOWN: "DOWN",

            self.GESTURE\_ROTATE\_CW: "ROTATE CW",

            self.GESTURE\_ROTATE\_CCW: "ROTATE CCW",

            self.GESTURE\_NONE: "NONE"

        }

    def get\_state(self):

        """Return the current state of the drone"""

        return {

            "position": self.position,

            "rotation": self.rotation,

            "is\_flying": self.is\_flying,

            "gesture": self.gesture\_labels.get(self.last\_gesture, "UNKNOWN")

        }

    def update\_state\_from\_gesture(self, gesture\_id):

        """Update the drone state based on the detected gesture"""

        # Log the incoming gesture

        logger.info(f"Received gesture ID: {gesture\_id} ({self.gesture\_labels.get(gesture\_id, 'UNKNOWN')})")

        # Add gesture to buffer

        self.gesture\_buffer.append(gesture\_id)

        if len(self.gesture\_buffer) > self.buffer\_size:

            self.gesture\_buffer.pop(0)

        # Get the most common gesture in buffer

        if len(self.gesture\_buffer) == self.buffer\_size:

            counter = {}

            for g in self.gesture\_buffer:

                if g not in counter:

                    counter[g] = 0

                counter[g] += 1

            # Find the most common gesture

            most\_common\_gesture = max(counter.items(), key=lambda x: x[1])

            logger.info(f"Most common gesture: {most\_common\_gesture[0]} ({self.gesture\_labels.get(most\_common\_gesture[0], 'UNKNOWN')}) count: {most\_common\_gesture[1]}")

            # Only update if the gesture appears at least 60% of the time

            threshold = 0.6 \* self.buffer\_size

            if most\_common\_gesture[1] >= threshold:

                gesture\_id = most\_common\_gesture[0]

                if gesture\_id != self.last\_gesture:

                    self.last\_gesture = gesture\_id

                    logger.info(f"Gesture detected: {self.gesture\_labels.get(gesture\_id, 'UNKNOWN')}")

            else:

                gesture\_id = self.GESTURE\_NONE

        else:

            gesture\_id = self.GESTURE\_NONE

        # Update drone state based on gesture

        self.\_process\_gesture\_command(gesture\_id)

        # Log the current state

        state = self.get\_state()

        logger.info(f"Current drone state: position={state['position']}, is\_flying={state['is\_flying']}, gesture={state['gesture']}")

        return state

    def \_process\_gesture\_command(self, gesture\_id):

        """Process a gesture command and update the drone state accordingly"""

        if gesture\_id == self.GESTURE\_TAKEOFF and not self.is\_flying:

            self.is\_flying = True

            logger.info("Drone taking off")

        elif gesture\_id == self.GESTURE\_LAND and self.is\_flying:

            self.is\_flying = False

            # Set Y position to 0 when landing (just like keyboard control)

            self.position["y"] = 0

            logger.info("Drone landing - setting height to zero")

        # Only process movement commands if the drone is flying

        if self.is\_flying:

            if gesture\_id == self.GESTURE\_FORWARD:

                self.position["x"] += self.MOVE\_SPEED \* math.cos(self.rotation["y"])

                self.position["z"] -= self.MOVE\_SPEED \* math.sin(self.rotation["y"])

            elif gesture\_id == self.GESTURE\_BACKWARD:

                self.position["x"] -= self.MOVE\_SPEED \* math.cos(self.rotation["y"])

                self.position["z"] += self.MOVE\_SPEED \* math.sin(self.rotation["y"])

            elif gesture\_id == self.GESTURE\_LEFT:

                self.position["x"] -= self.MOVE\_SPEED \* math.sin(self.rotation["y"])

                self.position["z"] -= self.MOVE\_SPEED \* math.cos(self.rotation["y"])

            elif gesture\_id == self.GESTURE\_RIGHT:

                self.position["x"] += self.MOVE\_SPEED \* math.sin(self.rotation["y"])

                self.position["z"] += self.MOVE\_SPEED \* math.cos(self.rotation["y"])

            elif gesture\_id == self.GESTURE\_UP:

                self.position["y"] += self.MOVE\_SPEED

            elif gesture\_id == self.GESTURE\_DOWN:

                self.position["y"] = max(0, self.position["y"] - self.MOVE\_SPEED)

            elif gesture\_id == self.GESTURE\_ROTATE\_CW:

                self.rotation["y"] += self.ROTATE\_SPEED

                # Normalize angle to [0, 2π]

                self.rotation["y"] %= 2 \* math.pi

            elif gesture\_id == self.GESTURE\_ROTATE\_CCW:

                self.rotation["y"] -= self.ROTATE\_SPEED

                # Normalize angle to [0, 2π]

                self.rotation["y"] %= 2 \* math.pi