function varargout = projectcitra(varargin)

% PROJECTCITRA MATLAB code for projectcitra.fig

% PROJECTCITRA, by itself, creates a new PROJECTCITRA or raises the existing

% singleton\*.

%

% H = PROJECTCITRA returns the handle to a new PROJECTCITRA or the handle to

% the existing singleton\*.

%

% PROJECTCITRA('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in PROJECTCITRA.M with the given input arguments.

%

% PROJECTCITRA('Property','Value',...) creates a new PROJECTCITRA or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before projectcitra\_OpeningFcn gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to projectcitra\_OpeningFcn via varargin.

%

% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one

% instance to run (singleton)".

%

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help projectcitra

% Last Modified by GUIDE v2.5 11-Nov-2024 14:09:26

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @projectcitra\_OpeningFcn, ...

'gui\_OutputFcn', @projectcitra\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before projectcitra is made visible.

function projectcitra\_OpeningFcn(hObject, eventdata, handles, varargin)

% Initialize the undo and redo stacks

handles.undoStack = {}; % Initialize undo stack

handles.redoStack = {}; % Initialize redo stack

% Initialize any other variables or GUI components

guidata(hObject, handles); % Save updated handles

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to projectcitra (see VARARGIN)

% Choose default command line output for projectcitra

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes projectcitra wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = projectcitra\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.

function pushbutton1\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Periksa apakah gambar berwarna, jika iya, konversi ke grayscale

if size(handles.im, 3) == 3

G = rgb2gray(handles.im); % Ubah ke grayscale

else

G = handles.im; % Jika sudah grayscale

end

% Tampilkan gambar grayscale di axes yang sudah disediakan

axes(handles.imageProcessed); % Akses area untuk gambar terproses

imshow(G); % Tampilkan gambar yang sudah diproses

% Update gambar terproses di handles.im

handles.im = G;

% Menampilkan histogram setelah gambar diproses

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Simpan handles terbaru

guidata(hObject, handles);

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% --- Executes on button press in bw.

function bw\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Convert the image to black and white (binary image)

bwImage = imbinarize(rgb2gray(handles.im)); % Convert to binary (black and white)

% Display the binary image

axes(handles.imageProcessed);

imshow(bwImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = bwImage;

guidata(hObject, handles);

% --- Executes on button press in smoothing.

function smoothing\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Apply smoothing using a Gaussian filter

smoothedImage = imgaussfilt(handles.im, 2); % Gaussian filter with sigma = 2

% Display the smoothed image

axes(handles.imageProcessed);

imshow(smoothedImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = smoothedImage;

guidata(hObject, handles);

% --- Executes on button press in rotation.

function rotation\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Rotate the image by 90 degrees

rotatedImage = imrotate(handles.im, 90);

% Display the rotated image

axes(handles.imageProcessed);

imshow(rotatedImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = rotatedImage;

guidata(hObject, handles);

% --- Executes on button press in sharpening.

function sharpening\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Apply sharpening to the image

sharpenedImage = imsharpen(handles.im);

% Display the sharpened image

axes(handles.imageProcessed);

imshow(sharpenedImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = sharpenedImage;

guidata(hObject, handles);

% --- Executes on button press in blur.

function blur\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Apply blurring using a simple averaging filter

blurredImage = imgaussfilt(handles.im, 5); % Gaussian filter with sigma = 5 for more blur

% Display the blurred image

axes(handles.imageProcessed);

imshow(blurredImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = blurredImage;

guidata(hObject, handles);

% --- Executes on button press in edge\_detect.

function edge\_detect\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Perform edge detection using the Sobel method

edgeImage = edge(rgb2gray(handles.im), 'Sobel');

% Display the edge-detected image

axes(handles.imageProcessed);

imshow(edgeImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = edgeImage;

guidata(hObject, handles);

% --- Executes on button press in hist\_equal.

function hist\_equal\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Periksa apakah gambar grayscale atau RGB

if size(handles.im, 3) == 3

imGray = rgb2gray(handles.im); % Jika RGB, ubah ke grayscale

else

imGray = handles.im; % Jika sudah grayscale, gunakan langsung

end

% Hitung histogram equalization secara manual (tanpa toolbox)

numBins = 256; % Untuk gambar 8-bit

pixelCounts = zeros(1, numBins); % Inisialisasi histogram dengan 0

[m, n] = size(imGray); % Dapatkan dimensi gambar

% Hitung frekuensi tiap level intensitas

for i = 1:m

for j = 1:n

intensity = imGray(i, j) + 1; % Intensitas dari 0-255

pixelCounts(intensity) = pixelCounts(intensity) + 1;

end

end

% Hitung CDF (Cumulative Distribution Function)

cdf = cumsum(pixelCounts) / numel(imGray);

% Remap nilai pixel lama ke nilai pixel baru (equalized)

imEq = zeros(m, n, 'uint8');

for i = 1:m

for j = 1:n

imEq(i, j) = uint8(255 \* cdf(imGray(i, j) + 1)); % Mapping pixel values

end

end

% Update gambar terproses di handles.im SEBELUM memanggil show\_hist\_Callback

handles.im = imEq;

% Tampilkan gambar yang sudah di-equalized di axes

axes(handles.imageProcessed); % Akses area untuk gambar terproses

imshow(imEq); % Tampilkan gambar hasil equalization

% Panggil fungsi untuk memperbarui histogram

show\_hist\_Callback(hObject, eventdata, handles);

% Simpan handles terbaru

guidata(hObject, handles);

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in vertical.

function vertical\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Flip the image vertically

flippedImage = flipud(handles.im); % Flip image upside down

% Display the flipped image

axes(handles.imageProcessed);

imshow(flippedImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = flippedImage;

guidata(hObject, handles);

% --- Executes on button press in pick\_image.

function pick\_image\_Callback(hObject, eventdata, handles)

[FileName, PathName] = uigetfile({'\*.jpg;\*.jpeg;\*.png;\*.webp;\*.tif;\*.bmp', 'Pilih Citra (JPG, JPEG, PNG, WEBP, TIF, BMP)'});

if FileName ~= 0 % Memeriksa apakah file dipilih

% Baca gambar dengan dukungan untuk WEBP

[~, ~, ext] = fileparts(FileName); % Mendapatkan ekstensi file

if strcmpi(ext, '.webp')

im = imread([PathName, FileName], 'webp'); % Membaca file WEBP

else

im = imread([PathName, FileName]); % Membaca format lain

end

% Set gambar yang dimuat sebagai gambar saat ini

handles.im = im;

handles.original\_im = im; % Menyimpan gambar asli di handles

% Mendapatkan dimensi gambar asli

[height, width, ~] = size(im);

% Menampilkan dimensi gambar asli di kolom w\_ori dan h\_ori

set(handles.w\_ori, 'String', num2str(width)); % Set lebar di w\_ori

set(handles.h\_ori, 'String', num2str(height)); % Set tinggi di h\_ori

% Memperbarui struktur handles

guidata(hObject, handles);

% Menampilkan gambar asli di axes gambarRaw

axes(handles.imageRaw);

imshow(im);

% Opsional: Menghapus area gambar yang telah diproses

cla(handles.imageProcessed); % Menghapus area gambar yang diproses sebelumnya

% Menampilkan histogram gambar asli

show\_hist\_Callback(hObject, eventdata, handles); % Memanggil fungsi untuk menampilkan histogram gambar

end

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in horizontal.

function horizontal\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Flip the image horizontally

flippedImage = fliplr(handles.im); % Flip image left to right

% Display the flipped image

axes(handles.imageProcessed);

imshow(flippedImage);

show\_hist\_Callback(hObject, eventdata, handles); % Call the histogram update function

% Update the processed image in handles

handles.im = flippedImage;

guidata(hObject, handles);

% --- Executes on button press in save\_image.

function save\_image\_Callback(hObject, eventdata, handles)

thresh = handles.im;

[name\_file\_save, path\_save] = uiputfile({ ...

'\*.bmp', 'File Bitmap (\*.bmp)'; ...

'\*.jpg', 'File JPG (\*.jpg)'; ...

'\*.jpeg', 'File JPEG (\*.jpeg)'; ...

'\*.png', 'File PNG (\*.png)'; ...

'\*.tif', 'File TIF (\*.tif)'; ...

'\*.webp', 'File WEBP (\*.webp)'; ...

'\*.\*', 'All Files (\*.\*)'}, ...

'Save Image');

if ~isequal(name\_file\_save, 0)

% Menyimpan file dalam format WEBP jika dipilih

[~, ~, ext] = fileparts(name\_file\_save);

if strcmpi(ext, '.webp')

imwrite(thresh, fullfile(path\_save, name\_file\_save), 'webp');

else

imwrite(thresh, fullfile(path\_save, name\_file\_save)); % Save other formats

end

else

return

end

% hObject handle to save\_image (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --------------------------------------------------------------------

function proses\_Callback(hObject, eventdata, handles)

set(handles.panelProses, 'visible', 'on');

% hObject handle to proses (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in reset.

function reset\_Callback(hObject, eventdata, handles)

% Cek apakah field 'original\_im' ada dan sudah di-set

if isfield(handles, 'original\_im') && ~isempty(handles.original\_im)

% Reset gambar ke gambar asli dan tampilkan di imageProcessed

handles.im = handles.original\_im; % Kembalikan gambar asli ke handles.im

axes(handles.imageProcessed);

imshow(handles.im); % Tampilkan gambar asli di axes untuk processed image

% Kosongkan axes untuk processed image jika perlu

%cla(handles.imageProcessed); % Kosongkan gambar yang sudah diproses jika perlu

cla(handles.histogram\_asli);

cla(handles.histogram\_edit);

% Menyembunyikan angka pada sumbu dan garis grid, hanya menyisakan kotak axes

set(handles.histogram\_asli, 'XTick', [], 'YTick', []); % Sembunyikan angka sumbu di histogram asli

set(handles.histogram\_asli, 'XColor', 'k', 'YColor', 'k'); % Warna kotak tetap hitam

box(handles.histogram\_asli, 'on'); % Pastikan kotak tetap terlihat

set(handles.histogram\_edit, 'XTick', [], 'YTick', []); % Sembunyikan angka sumbu di histogram edit

set(handles.histogram\_edit, 'XColor', 'k', 'YColor', 'k'); % Warna kotak tetap hitam

box(handles.histogram\_edit, 'on'); % Pastikan kotak tetap terlihat

set(handles.w, 'String', '');

set(handles.h, 'String', '');

set(handles.adj\_brightness, 'Value', 0);

set(handles.adj\_contrast, 'Value', 0);

set(handles.adj\_clarity, 'Value', 0);

set(handles.adj\_saturation, 'Value', 0);

set(handles.adj\_hue, 'Value', 0);

set(handles.adj\_shadows, 'Value', 0);

set(handles.adj\_highlights, 'Value', 0);

set(handles.adj\_temperature, 'Value', 0);

set(handles.rgb\_red, 'Value', 0);

set(handles.rgb\_green, 'Value', 0);

set(handles.rgb\_blue, 'Value', 0);

% Update handles

guidata(hObject, handles);

% Menampilkan histogram untuk gambar asli (memanggil fungsi show\_hist untuk gambar asli)

show\_hist\_Callback(hObject, eventdata, handles, 'original');

else

% Jika 'original\_im' belum ada, beri notifikasi

msgbox('Original image not loaded. Please load an image first.');

end

% hObject handle to reset (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

function w\_Callback(hObject, eventdata, handles)

% hObject handle to w (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of w as text

% str2double(get(hObject,'String')) returns contents of w as a double

% --- Executes during object creation, after setting all properties.

function w\_CreateFcn(hObject, eventdata, handles)

% hObject handle to w (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function h\_Callback(hObject, eventdata, handles)

% hObject handle to h (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of h as text

% str2double(get(hObject,'String')) returns contents of h as a double

% --- Executes during object creation, after setting all properties.

function h\_CreateFcn(hObject, eventdata, handles)

% hObject handle to h (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in resize.

function resize\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Only save the state if it's a valid image

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Save updated handles

end

% Get height and width values from input fields

height = str2double(get(handles.h, 'string'));

weight = str2double(get(handles.w, 'string'));

% Check if height and weight values are valid

if isnan(height) || isnan(weight) || height <= 0 || weight <= 0

msgbox('Please enter valid height and weight values.', 'Error', 'error');

return;

end

% Resize the image

resizedImage = imresize(handles.im, [weight height]);

% Display the resized image

axes(handles.imageProcessed);

imshow(resizedImage);

% Update the processed image in handles

handles.im = resizedImage;

guidata(hObject, handles);

% hObject handle to resize (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

function w\_ori\_Callback(hObject, eventdata, handles)

% hObject handle to w\_ori (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of w\_ori as text

% str2double(get(hObject,'String')) returns contents of w\_ori as a double

% --- Executes during object creation, after setting all properties.

function w\_ori\_CreateFcn(hObject, eventdata, handles)

% hObject handle to w\_ori (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

function h\_ori\_Callback(hObject, eventdata, handles)

% hObject handle to h\_ori (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of h\_ori as text

% str2double(get(hObject,'String')) returns contents of h\_ori as a double

% --- Executes during object creation, after setting all properties.

function h\_ori\_CreateFcn(hObject, eventdata, handles)

% hObject handle to h\_ori (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on slider movement.

function adj\_brightness\_Callback(hObject, eventdata, handles)

brightness\_value = get(hObject, 'Value');

if brightness\_value == 0

imshow(handles.originalImage, 'Parent', handles.imageAxes); % Display the original image

else

updateImageAdjustments(handles, brightness\_value, 'brightness');

end

% --- Executes during object creation, after setting all properties.

function adj\_brightness\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_clarity\_Callback(hObject, eventdata, handles)

clarity\_value = get(hObject, 'Value');

updateImageAdjustments(handles, clarity\_value, 'clarity');

% --- Executes during object creation, after setting all properties.

function adj\_clarity\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_contrast\_Callback(hObject, eventdata, handles)

contrast\_value = get(hObject, 'Value');

updateImageAdjustments(handles, contrast\_value, 'contrast');

% --- Executes during object creation, after setting all properties.

function adj\_contrast\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_saturation\_Callback(hObject, eventdata, handles)

saturation\_value = get(hObject, 'Value');

updateImageAdjustments(handles, saturation\_value, 'saturation');

% --- Executes during object creation, after setting all properties.

function adj\_saturation\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_hue\_Callback(hObject, eventdata, handles)

hue\_value = get(hObject, 'Value');

updateImageAdjustments(handles, hue\_value, 'hue');

% --- Executes during object creation, after setting all properties.

function adj\_hue\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_shadows\_Callback(hObject, eventdata, handles)

shadows\_value = get(hObject, 'Value');

updateImageAdjustments(handles, shadows\_value, 'shadows');

% --- Executes during object creation, after setting all properties.

function adj\_shadows\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_highlights\_Callback(hObject, eventdata, handles)

highlights\_value = get(hObject, 'Value');

updateImageAdjustments(handles, highlights\_value, 'highlights');

% --- Executes during object creation, after setting all properties.

function adj\_highlights\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on slider movement.

function adj\_temperature\_Callback(hObject, eventdata, handles)

temperature\_value = get(hObject, 'Value');

updateImageAdjustments(handles, temperature\_value, 'temperature');

% --- Executes during object creation, after setting all properties.

function adj\_temperature\_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject, 'BackgroundColor'), get(0, 'defaultUicontrolBackgroundColor'))

set(hObject, 'BackgroundColor', [0.9 0.9 0.9]);

end

set(hObject, 'Min', -1);

set(hObject, 'Max', 1);

set(hObject, 'Value', 0);

% --- Executes on button press in exit.

function exit\_Callback(hObject, eventdata, handles)

% hObject handle to exit (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton20.

function pushbutton20\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton20 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in undo.

function undo\_Callback(hObject, eventdata, handles)

if ~isempty(handles.undoStack)

% Pop the last image from the undo stack

previousImage = handles.undoStack{end};

handles.undoStack(end) = []; % Remove it from the undo stack

% Push the current image to the redo stack before making the change

handles.redoStack{end+1} = handles.im; % Store the image in redo stack

% Update the displayed image to the previous state

handles.im = previousImage;

axes(handles.imageProcessed);

imshow(handles.im);

% Update the GUI state

guidata(handles.figure1, handles); % Save updated handles

else

msgbox('No more actions to undo', 'Undo Error', 'error');

end

% hObject handle to undo (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in redo.

function redo\_Callback(hObject, eventdata, handles)

if ~isempty(handles.redoStack)

% Pop the last image from the redo stack

nextImage = handles.redoStack{end};

handles.redoStack(end) = []; % Remove it from the redo stack

% Push the current image to the undo stack before making the change

handles.undoStack{end+1} = handles.im; % Store the image in undo stack

% Update the displayed image to the next state

handles.im = nextImage;

axes(handles.imageProcessed);

imshow(handles.im);

% Update the GUI state

guidata(handles.figure1, handles); % Save updated handles

else

msgbox('No more actions to redo', 'Redo Error', 'error');

end

% hObject handle to redo (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton22.

function pushbutton22\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton22 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

function saveState(handles)

% Save the current image state in the undo stack before making any changes

if isfield(handles, 'im')

handles.undoStack{end+1} = handles.im; % Store the current image

guidata(handles.figure1, handles); % Save updated handles

end

% --- Executes on slider movement.

function slidered\_Callback(hObject, eventdata, handles)

red\_value = get(hObject, 'Value');

updateImageAdjustments(handles, red\_value, 'red');

% hObject handle to slidered (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider

% get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes during object creation, after setting all properties.

function slidered\_CreateFcn(hObject, eventdata, handles)

% hObject handle to slidered (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on slider movement.

function slidegreen\_Callback(hObject, eventdata, handles)

green\_value = get(hObject, 'Value');

updateImageAdjustments(handles, green\_value, 'green');

% hObject handle to slidegreen (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider

% get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes during object creation, after setting all properties.

function slidegreen\_CreateFcn(hObject, eventdata, handles)

% hObject handle to slidegreen (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

% --- Executes on slider movement.

function slideblue\_Callback(hObject, eventdata, handles)

blue\_value = get(hObject, 'Value');

updateImageAdjustments(handles, blue\_value, 'blue');

% hObject handle to slideblue (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider

% get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes during object creation, after setting all properties.

function slideblue\_CreateFcn(hObject, eventdata, handles)

% hObject handle to slideblue (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.

if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor',[.9 .9 .9]);

end

function updateImageAdjustments(handles, adjustment\_value, adjustment\_type)

% Check if the original image is available

if isfield(handles, 'original\_im') && ~isempty(handles.original\_im)

% Initialize adjusted\_image as the original image

adjusted\_image = double(handles.original\_im) / 255;

% Apply adjustment based on type

switch adjustment\_type

case 'brightness'

adjusted\_image = adjusted\_image + adjustment\_value;

case 'contrast'

factor = 1 + adjustment\_value;

adjusted\_image = 0.5 + factor \* (adjusted\_image - 0.5);

case 'clarity'

if adjustment\_value ~= 0

kernel = [0 -1 0; -1 5 -1; 0 -1 0]; % Sharpening filter

for channel = 1:3

adjusted\_image(:, :, channel) = applyKernel(adjusted\_image(:, :, channel), kernel);

end

end

case 'saturation'

grayscale = 0.2989 \* adjusted\_image(:, :, 1) + 0.5870 \* adjusted\_image(:, :, 2) + 0.1140 \* adjusted\_image(:, :, 3);

adjusted\_image = adjusted\_image + adjustment\_value \* (adjusted\_image - cat(3, grayscale, grayscale, grayscale));

case 'hue'

if adjustment\_value ~= 0

hsvImage = rgb2hsv(adjusted\_image);

hsvImage(:, :, 1) = mod(hsvImage(:, :, 1) + adjustment\_value, 1);

adjusted\_image = hsv2rgb(hsvImage);

end

case 'shadows'

shadows\_mask = adjusted\_image < 0.5;

adjusted\_image = adjusted\_image - shadows\_mask .\* adjustment\_value;

case 'highlights'

highlights\_mask = adjusted\_image >= 0.5;

adjusted\_image = adjusted\_image + highlights\_mask .\* adjustment\_value;

case 'temperature'

warm\_filter = cat(3, 1.1, 1.0, 0.9);

cool\_filter = cat(3, 0.9, 1.0, 1.1);

if adjustment\_value > 0

adjusted\_image = adjusted\_image .\* (1 + adjustment\_value \* (warm\_filter - 1));

else

adjusted\_image = adjusted\_image .\* (1 + abs(adjustment\_value) \* (cool\_filter - 1));

end

case 'grayscale'

% Convert to grayscale

if size(adjusted\_image, 3) == 3

adjusted\_image = rgb2gray(adjusted\_image); % Convert RGB to grayscale

end

case 'bw'

adjusted\_image = imbinarize(rgb2gray(adjusted\_image));

case 'smoothing'

adjusted\_image = imgaussfilt(adjusted\_image, 2);

case 'rotation'

adjusted\_image = imrotate(adjusted\_image, 90);

case 'sharpening'

adjusted\_image = imsharpen(adjusted\_image);

case 'blur'

adjusted\_image = imgaussfilt(adjusted\_image, 5);

case 'edge\_detect'

adjusted\_image = edge(rgb2gray(adjusted\_image), 'Sobel');

case 'hist\_equal'

imGray = rgb2gray(adjusted\_image);

adjusted\_image = histeq(imGray);

case 'red'

adjusted\_image(:, :, 1) = adjusted\_image(:, :, 1) + adjustment\_value;

case 'green'

adjusted\_image(:, :, 2) = adjusted\_image(:, :, 2) + adjustment\_value;

case 'blue'

adjusted\_image(:, :, 3) = adjusted\_image(:, :, 3) + adjustment\_value;

end

% Clip values to [0, 1] and scale to uint8 for display

adjusted\_image = uint8(min(max(adjusted\_image, 0), 1) \* 255);

% Display the adjusted image

axes(handles.imageProcessed);

imshow(adjusted\_image);

% Save adjusted image in handles

handles.adjusted\_im = adjusted\_image;

handles.im = adjusted\_image; % Update for histogram

guidata(handles.figure1, handles); % Save updated handles

% Update histogram for the processed image

show\_hist\_Callback([], [], handles);

else

msgbox('Original image not loaded. Please load an image first.', 'Error', 'error');

end

function output = applyKernel(input\_image, kernel)

% Terapkan kernel 3x3 secara manual (tanpa toolbox)

[rows, cols] = size(input\_image);

% Membuat citra dengan padding manual

pad\_image = zeros(rows + 2, cols + 2);

pad\_image(2:end-1, 2:end-1) = input\_image; % Tempatkan citra asli di tengah

% Isi padding dengan mereplikasi tepi

pad\_image(1, 2:end-1) = input\_image(1, :); % Baris atas

pad\_image(end, 2:end-1) = input\_image(end, :); % Baris bawah

pad\_image(2:end-1, 1) = input\_image(:, 1); % Kolom kiri

pad\_image(2:end-1, end) = input\_image(:, end); % Kolom kanan

pad\_image(1, 1) = input\_image(1, 1); % Pojok kiri atas

pad\_image(1, end) = input\_image(1, end); % Pojok kanan atas

pad\_image(end, 1) = input\_image(end, 1); % Pojok kiri bawah

pad\_image(end, end) = input\_image(end, end); % Pojok kanan bawah

output = zeros(rows, cols); % Inisialisasi citra keluaran

% Konvolusi dengan kernel

for r = 1:rows

for c = 1:cols

sub\_matrix = pad\_image(r:r+2, c:c+2); % Ambil wilayah 3x3

output(r, c) = sum(sum(sub\_matrix .\* kernel)); % Terapkan kernel dan jumlahkan

end

end

% Pemangkasan hasil konvolusi agar tetap dalam rentang [0, 1]

output = min(max(output, 0), 1);

function removebg\_Callback(hObject, eventdata, handles)

if isfield(handles, 'im')

% Hanya simpan keadaan jika itu gambar yang valid

handles.undoStack{end+1} = handles.im;

guidata(handles.figure1, handles); % Simpan handles yang diperbarui

end

% Periksa apakah gambar asli atau yang sudah difilter ada

if isfield(handles, 'adjusted\_im') && ~isempty(handles.adjusted\_im)

im = handles.adjusted\_im; % Gunakan gambar yang sudah difilter

elseif isfield(handles, 'original\_im') && ~isempty(handles.original\_im)

im = handles.original\_im; % Jika tidak ada gambar yang difilter, gunakan gambar asli

else

msgbox('Gambar belum dimuat.', 'Error', 'error');

return;

end

% Ubah gambar menjadi presisi ganda [0, 1] untuk pengolahan

im\_double = double(im) / 255;

% Ubah gambar menjadi grayscale untuk mendeteksi latar belakang

gray\_im = 0.2989 \* im\_double(:, :, 1) + 0.5870 \* im\_double(:, :, 2) + 0.1140 \* im\_double(:, :, 3);

% Hitung intensitas rata-rata gambar untuk menentukan latar belakang

avg\_intensity = mean(gray\_im(:));

% Tentukan threshold untuk latar belakang terang atau gelap

if avg\_intensity > 0.5

threshold = 0.8; % Latar belakang terang

bg\_mask = gray\_im > threshold; % Masker untuk latar belakang terang

else

threshold = 0.2; % Latar belakang gelap

bg\_mask = gray\_im < threshold; % Masker untuk latar belakang gelap

end

% Buat saluran alpha untuk transparansi

alpha\_channel = ones(size(im, 1), size(im, 2)); % Default transparan penuh

alpha\_channel(bg\_mask) = 0; % Set piksel latar belakang menjadi transparan

% Terapkan masker ke gambar RGB, set piksel latar belakang menjadi transparan

im\_double(bg\_mask) = 0; % Set piksel latar belakang menjadi 0 (untuk transparansi)

% Konversi kembali gambar menjadi uint8

im\_rgb = uint8(im\_double \* 255); % Ubah gambar menjadi uint8 untuk disimpan

% Tampilkan gambar dengan latar belakang transparan

axes(handles.imageProcessed);

imshow(im\_rgb); % Gambar dengan latar belakang transparan

% Simpan gambar yang disesuaikan dengan latar belakang transparan

handles.adjusted\_im = im\_rgb; % Simpan gambar yang telah diubah

guidata(handles.figure1, handles); % Simpan struktur handles yang diperbarui

% Tanyakan lokasi dan nama file untuk menyimpan gambar

[fileName, filePath] = uiputfile('\*.png', 'Save Image As');

if fileName

% Simpan gambar dengan transparansi (alpha channel)

imwrite(im\_rgb, fullfile(filePath, fileName), 'png', 'Alpha', alpha\_channel);

end

% --- Executes on button press in show\_hist.

function show\_hist\_Callback(hObject, eventdata, handles)

% Display histogram for the original image if available

if isfield(handles, 'original\_im') && ~isempty(handles.original\_im)

img\_rgb\_original = handles.original\_im;

% If grayscale, convert to RGB

if size(img\_rgb\_original, 3) == 1

img\_rgb\_original = cat(3, img\_rgb\_original, img\_rgb\_original, img\_rgb\_original);

end

% Calculate histogram for each RGB channel

hist\_r\_original = calculate\_histogram(img\_rgb\_original(:, :, 1));

hist\_g\_original = calculate\_histogram(img\_rgb\_original(:, :, 2));

hist\_b\_original = calculate\_histogram(img\_rgb\_original(:, :, 3));

% Display the histogram for the original image

axes(handles.histogram\_asli); % Ensure correct axes are targeted

cla; % Clear previous plots

hold on;

bar(hist\_r\_original, 'FaceColor', 'r', 'BarWidth', 1);

bar(hist\_g\_original, 'FaceColor', 'g', 'BarWidth', 1);

bar(hist\_b\_original, 'FaceColor', 'b', 'BarWidth', 1);

hold off;

xticks([]);

yticks([]);

ylim([0, max([hist\_r\_original, hist\_g\_original, hist\_b\_original]) \* 1.1]);

end

if isfield(handles, 'im') && ~isempty(handles.im)

img\_rgb\_processed = handles.im;

% If grayscale, convert to RGB

if size(img\_rgb\_processed, 3) == 1

img\_rgb\_processed = cat(3, img\_rgb\_processed, img\_rgb\_processed, img\_rgb\_processed);

end

% Calculate histogram for each RGB channel

hist\_r\_processed = calculate\_histogram(img\_rgb\_processed(:, :, 1));

hist\_g\_processed = calculate\_histogram(img\_rgb\_processed(:, :, 2));

hist\_b\_processed = calculate\_histogram(img\_rgb\_processed(:, :, 3));

% Display the histogram for the processed image

axes(handles.histogram\_edit); % Ensure correct axes are targeted

cla; % Clear previous plots

hold on;

bar(hist\_r\_processed, 'FaceColor', 'r', 'BarWidth', 1);

bar(hist\_g\_processed, 'FaceColor', 'g', 'BarWidth', 1);

bar(hist\_b\_processed, 'FaceColor', 'b', 'BarWidth', 1);

hold off;

xticks([]);

yticks([]);

ylim([0, max([hist\_r\_processed, hist\_g\_processed, hist\_b\_processed]) \* 1.1]);

end

% Save updated handles

guidata(hObject, handles);

% Fungsi untuk menghitung histogram manual

function hist = calculate\_histogram(img)

img = double(img);

hist = zeros(1, 256);

for i = 1:numel(img)

pixel\_value = round(img(i));

if pixel\_value >= 0 && pixel\_value <= 255

hist(pixel\_value + 1) = hist(pixel\_value + 1) + 1;

end

end