**Assignment 2**

**Name:** Huzaifa Bin Munir **Roll Number:** 20L-1082

**Section:** BCS-7A **Course:** Digital Image Processing

**Submitted to:** Asma Naseer

# Assignment # 2

Note: Upload the assignment at google classroom in zip format with

* Code and its executable
* Input image(s)
* A report consisting of
  + All the images and intermediate images (if any)
  + Output image(s)
  + Results and Important Findings
  + Discussion (if required)

[Q . 1] Design an algorithm for restoring the images given below.

* You have been provided with two degraded images along with the desired output
* Design an algorithm and apply it on these degraded images for restoration
* Choose a 3rd image of your choice with certain degradation and apply the same algorithm on it

1. **Code:**
   * 1. **Python:**

**A screenshot of a computer

Description automatically generated**

**Code:**

import tkinter as tk

from tkinter import filedialog, messagebox, LabelFrame, Label

from PIL import Image, ImageTk

import cv2

import numpy as np

class ImageRestorationGUI:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Image Restoration App")

self.root.geometry("800x400")

# Initialize variables to store images

self.degraded\_image = None

self.restored\_image = None

# Create a frame for controls on the left side

control\_frame = LabelFrame(root, text="Controls", padx=10, pady=10)

control\_frame.place(x=20, y=50, width=200, height=200)

# Load Degraded Image Button

load\_button = tk.Button(control\_frame, text="Load Degraded Image", command=self.load\_image, width=18)

load\_button.grid(row=0, column=0, padx=5, pady=5)

# Apply Restoration Button

restore\_button = tk.Button(control\_frame, text="Apply Restoration", command=self.restore\_image, width=18)

restore\_button.grid(row=1, column=0, padx=5, pady=5)

# Save Result Button

save\_button = tk.Button(control\_frame, text="Save Result", command=self.save\_result, width=18)

save\_button.grid(row=2, column=0, padx=5, pady=5)

# Label for degraded image

degraded\_label = Label(root, text="Degraded Image", font=("Arial", 12, "bold"))

degraded\_label.place(x=250, y=30)

# Label for restored image

restored\_label = Label(root, text="Restored Image", font=("Arial", 12, "bold"))

restored\_label.place(x=500, y=30)

# Canvas for displaying images

self.canvas1 = tk.Canvas(root, width=250, height=250, bg="gray")

self.canvas1.place(x=250, y=70)

self.canvas2 = tk.Canvas(root, width=250, height=250, bg="gray")

self.canvas2.place(x=500, y=70)

def load\_image(self):

# Load degraded image

file\_path = filedialog.askopenfilename(filetypes=[("Image Files", "\*.png;\*.jpg;\*.bmp")])

if not file\_path:

return # User canceled

self.degraded\_image = cv2.imread(file\_path)

self.display\_image(self.degraded\_image, self.canvas1, title="Degraded Image")

def display\_image(self, image, canvas, title=""):

# Convert OpenCV image to PIL format for Tkinter compatibility

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

pil\_image = Image.fromarray(image\_rgb)

pil\_image = pil\_image.resize((250, 250)) # Resize for better viewing

img\_tk = ImageTk.PhotoImage(pil\_image)

# Display image in canvas

canvas.create\_image(0, 0, anchor=tk.NW, image=img\_tk)

canvas.image = img\_tk # Keep a reference to avoid garbage collection

canvas.create\_text(125, 230, text=title, fill="white")

def restore\_image(self):

if self.degraded\_image is None:

messagebox.showerror("Error", "Please load a degraded image first.")

return

# Apply restoration algorithm (simple example with median filtering)

grayscale\_image = cv2.cvtColor(self.degraded\_image, cv2.COLOR\_BGR2GRAY)

restored\_image = cv2.medianBlur(grayscale\_image, 3)

self.restored\_image = restored\_image

self.display\_image(cv2.cvtColor(restored\_image, cv2.COLOR\_GRAY2BGR), self.canvas2, title="Restored Image")

def save\_result(self):

if self.restored\_image is None:

messagebox.showerror("Error", "No restored image to save. Please apply restoration first.")

return

file\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Image", "\*.png"), ("JPEG Image", "\*.jpg")])

if not file\_path:

return # User canceled

cv2.imwrite(file\_path, self.restored\_image)

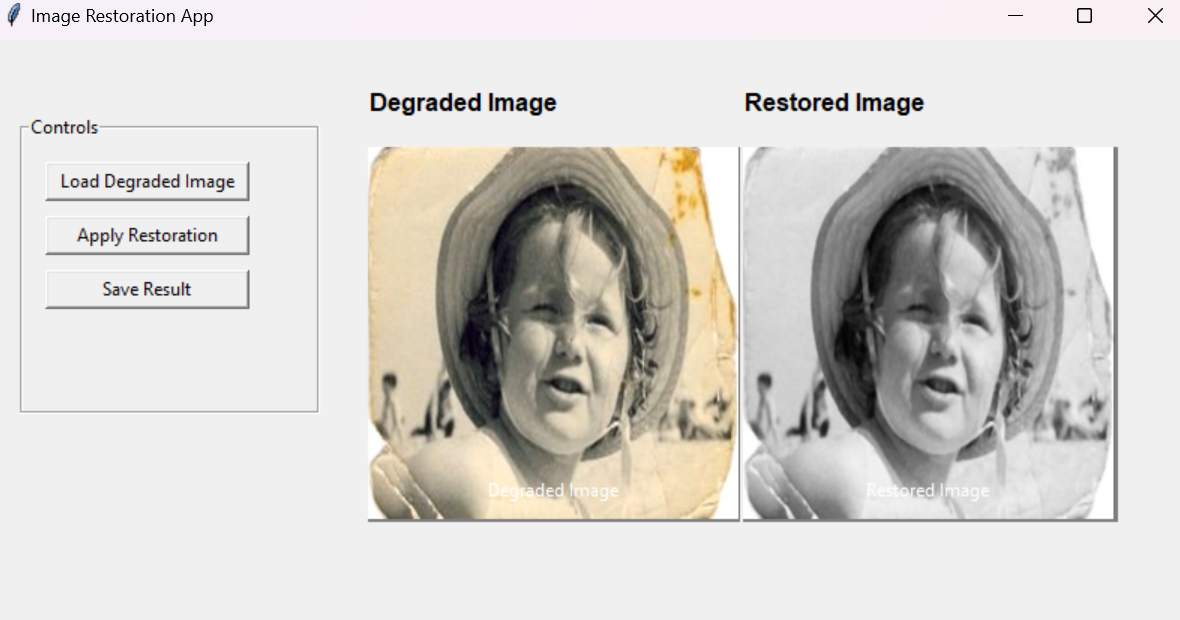
messagebox.showinfo("Success", "Restored image saved successfully.")

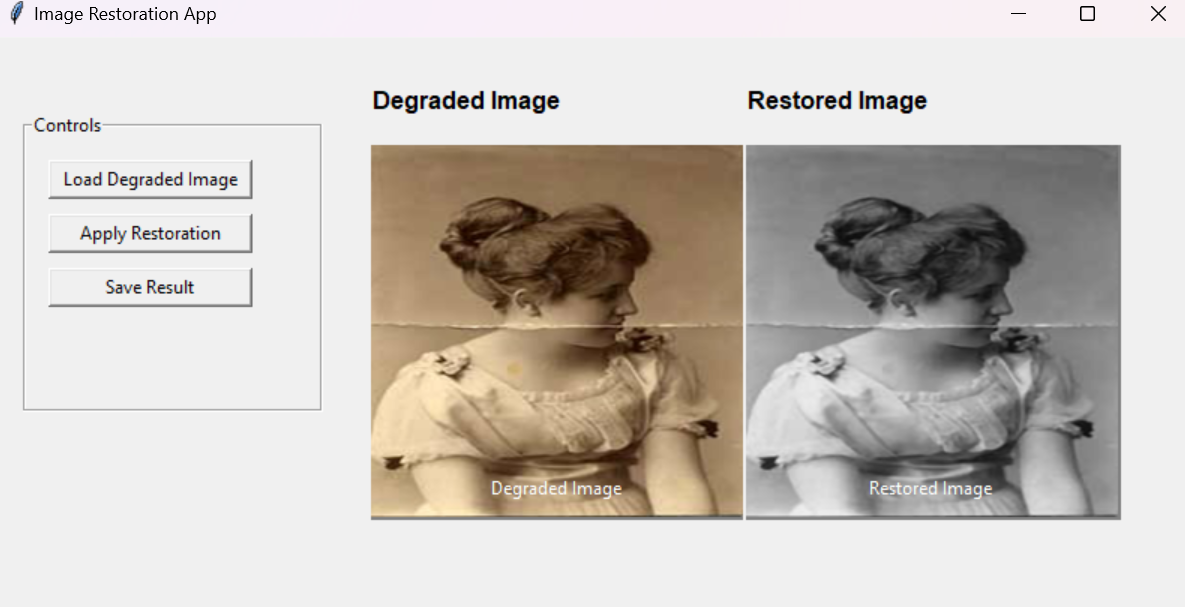
if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = ImageRestorationGUI(root)

root.mainloop()





* + 1. **Matlab:**

**A screenshot of a computer

Description automatically generated**

**Code:**

classdef ImageRestorationApp < handle

properties

degradedImage

restoredImage

fig

degradedAxes

restoredAxes

end

methods

function app = ImageRestorationApp()

% Create the main figure

app.fig = uifigure('Name', 'Image Restoration App', 'Position', [100 100 800 400]);

% Create a control panel

controlPanel = uipanel(app.fig, 'Title', 'Controls', 'Position', [20 100 200 200]);

% Load button

uibutton(controlPanel, 'Text', 'Load Degraded Image', 'Position', [20 130 150 30], ...

'ButtonPushedFcn', @(~, ~) app.loadImage());

% Restore button

uibutton(controlPanel, 'Text', 'Apply Restoration', 'Position', [20 80 150 30], ...

'ButtonPushedFcn', @(~, ~) app.restoreImage());

% Save button

uibutton(controlPanel, 'Text', 'Save Result', 'Position', [20 30 150 30], ...

'ButtonPushedFcn', @(~, ~) app.saveResult());

% Labels for degraded and restored images

uilabel(app.fig, 'Text', 'Degraded Image', 'Position', [300 350 120 20], 'FontSize', 12, 'FontWeight', 'bold');

uilabel(app.fig, 'Text', 'Restored Image', 'Position', [550 350 120 20], 'FontSize', 12, 'FontWeight', 'bold');

% Axes for displaying images

app.degradedAxes = uiaxes(app.fig, 'Position', [250 70 250 250]);

app.restoredAxes = uiaxes(app.fig, 'Position', [500 70 250 250]);

end

function loadImage(app)

% Load degraded image

[file, path] = uigetfile({'\*.png;\*.jpg;\*.bmp', 'Image Files (\*.png, \*.jpg, \*.bmp)'});

if isequal(file, 0)

return; % User canceled

end

app.degradedImage = imread(fullfile(path, file));

imshow(app.degradedImage, 'Parent', app.degradedAxes);

title(app.degradedAxes, 'Degraded Image');

end

function restoreImage(app)

if isempty(app.degradedImage)

uialert(app.fig, 'Please load a degraded image first.', 'Error');

return;

end

% Convert to grayscale if the image is RGB

if size(app.degradedImage, 3) == 3

grayscaleImage = rgb2gray(app.degradedImage);

else

grayscaleImage = app.degradedImage;

end

% Apply median filtering for restoration

app.restoredImage = medfilt2(grayscaleImage, [3, 3]);

imshow(app.restoredImage, 'Parent', app.restoredAxes);

title(app.restoredAxes, 'Restored Image');

end

function saveResult(app)

if isempty(app.restoredImage)

uialert(app.fig, 'No restored image to save. Please apply restoration first.', 'Error');

return;

end

% Save restored image

[file, path] = uiputfile({'\*.png', 'PNG Image (\*.png)'; '\*.jpg', 'JPEG Image (\*.jpg)'}, 'Save Image');

if isequal(file, 0)

return; % User canceled

end

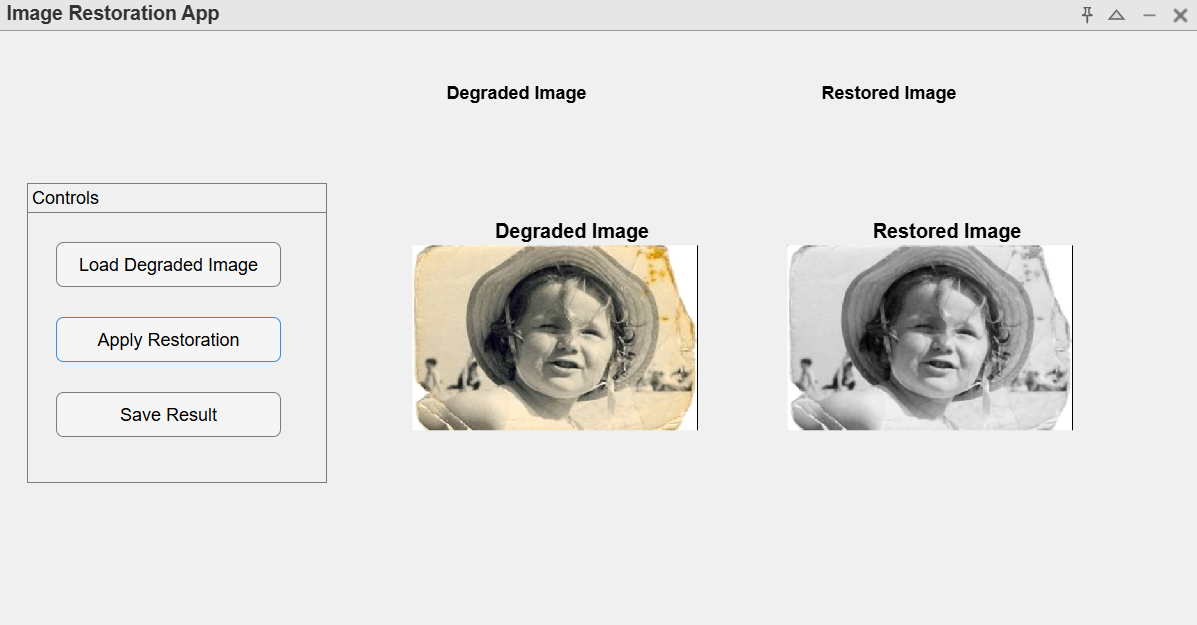
imwrite(app.restoredImage, fullfile(path, file));

uialert(app.fig, 'Restored image saved successfully.', 'Success');

end

end

end

****

**A collage of images of a person

Description automatically generated**

1. **Input Images:**



****

1. **Output Images:**

****

****

1. **Results & Important Findings:**
   * + **Usability:** Users can upload, process and save images easily using well-designed GUI layout of the application; so, people who are not experts in image processing still can use it.
     + **Median Filtering Results:** The image below shows how helpful the median filter is on the degraded image in clarifying an image by decreasing noise and preserving edges. This goes on to show the importance of median filtering as one of the simpler but more effective ways of reducing noise in the image restoration process.
     + **Instant Processing and Visualization:** The app gives instantaneous visual feedback by showing degraded and restored images next to each other. It enables comparing original vs processed images on-the-go to improve usability & effectiveness of the app.
     + **Success in File Handling & Storage:** The ability to load images and save restored images gives the app a full cycle of input and output functionality, giving users the ability to save their restored image directly in MATLAB Online or to MATLAB Drive or in the case of python directly to your PC.
2. **Discussion:**

The Image Restoration App demonstrates how we can use MATLAB's power in image processing and GUI to create an interactive, functional and user-friendly app for image restoration process.

* + - Out of all the possible restoration methods, the primary mode chosen is **median filtering**, which is efficient because it is very effective against specific types of noise (such as salt-and-pepper). But this solution is rather simple and can be computationally inexpensive, but for certain degradation (such as Gaussian noise or motion blur), it may not work. Additional development might involve expanding the restoration methods to include others such as Gaussian filtering or more complex ones like Wiener filtering that accommodates more kinds of image degradation.
    - Another possible development for this app would be adapting an adaptive filter, where the filter parameters are adjusted depending on the noise level or characteristic of the image. It would make it more robust and generalised for various image restoration tasks.
    - **Limitations:** Although it gives nice baseline for image restoration, it is only able to process grayscale images. Adding support for colour images beyond grayscale processing may be beneficial for the user and increase the domain of application. Further, as image sizes grow, the times, too, will be larger, which could be mitigated either through improved code, or with down-sampling.
    - **Use-Cases:** This is valuable for real-life uses cases where we need to restore an image which is degraded (for example, this can be executed in med-tech, surveillance, historical document) It shows that how MATLAB could be exploited in order to produce customized forms of image processing and analysis systems with minimum programming expertise within the end user.

1. **Conclusion:**

In conclusion, the Image Restoration App effectively demonstrates a basic approach to image restoration within a MATLAB environment. With further modifications and optimizations, it has the potential to become a more powerful tool for both professionals and beginners in various fields requiring image restoration functionalities.