NUTAN MAHARASHTRA VIDYA PRASARAK MANDAL'S

Under administrative support of Pimpri Chinchwad Education Trust

Nutan Maharashtra Institute of Engineering and Technology

Talegaon Dabhade, Pune

PROJECT REPORT ON

"PyMat Vision"

DEAPARTMENT OF ELECTRONICS &TELECOMMUNICATION ENGINEERING

(2024-25)

NUTAN MAHARASHTRA VIDYA PRASARAK MANDAL'S

Under administrative support of Pimpri Chinchwad Education Trust

Nutan Maharashtra Institute of Engineering and Technology

Talegaon Dabhade, Pune

CERTIFICATE

This is to certify that, this mini project report entitled: PyMat Vision Using PYTHON and MATLAB is a record of project work carried out in this college.

BY

Name	Roll No.	Seat No.
Soham Rajaram Mane	TEN 36	T400550142

Subject In charge	Head of the Department	Principal
Dr. Ashwini Shinde	Dr. Ashwini Shinde	Dr. S. N. Sapali

ACKNOWLEDGEMENT

With all respect and gratitude, we would like to thank all people who have helped us directly or indirectly for the completion of this dissertation work.

We thank our project guide **Dr. Ashwini Shinde** for helping us to understand the project topic conceptually in every phase of work. He offered us so much advice, patiently supervising and always guiding in right direction. We have learnt a lot from him and he is truly a dedicated mentor. His encouragement and help made us confident to fulfill my desire and overcome every difficulty we encountered.

We express my heartily gratitude towards **Dr. Ashwini Shinde**, Head of department of Electronics and Telecommunication Engineering for guiding us to understand the work conceptually and also for providing necessary information and required resources with his constant encouragement to complete this dissertation work.

With deep sense of gratitude, we thank our Principal **Dr. S. N. Sapali** and Management of the NMIET for providing all necessary facilities and their constant encouragement and support.

Last but not the least, we thank to all the Teaching & Non-teaching staff members of Electronics and Telecommunication Engineering Department for providing necessary information and required resources. We are ending this acknowledgement with deep indebtedness to my friends who have helped us.

INDEX

Abstract: -

PYMAT Vision presents an innovative, interactive digital image processing tool developed by integrating Python and MATLAB functionalities. The solution offers users a seamless experience to perform numerous image processing techniques through a user-friendly graphical interface. Users can execute tasks such as resizing, colour conversions (RGB to GRAYSCALE), noise filtering, contrast enhancements, feature extraction, thresholding, and edge detection. The hybrid implementation leverages the strengths of both Python and MATLAB, ensuring computational efficiency and high accuracy.

Introduction: -

Digital Image Processing (DIP) techniques are vital across various industries, including healthcare diagnostics, security surveillance, entertainment media, and industrial automation. However, due to the inherent complexity and technical knowledge required, these techniques often remain inaccessible to users lacking a strong programming background. To bridge this gap, the present project, "PYMAT Vision," introduces an integrated solution combining the intuitive scripting capabilities of Python with the robust computational power and specialized image processing functionalities of MATLAB. This integration ensures both ease of use and high-performance processing, enabling a broader audience—including students, researchers, and industry professionals—to perform sophisticated image processing tasks efficiently.

Benefits

- **User-Friendly Interface**: Simplifies complex image processing operations for users with minimal programming skills.
- **Enhanced Performance**: Combines Python's ease of scripting and MATLAB's high-speed computational capabilities.
- **Versatile Application**: Suitable for educational, research, and real-world industrial scenarios.
- **Accessible Learning**: Provides practical exposure to advanced image processing techniques, fostering better understanding and skill development.

Technical Architecture Diagram:-

A simple flowchart showing the **system architecture** would add immense clarity and professionalism:

User Input ➔ Python GUI ➔ MATLAB Engine ➔ Processing ➔ Display ➔ Download

Scope: -

The developed application serves educational purposes, research-oriented tasks, and real-world image processing applications. The scope includes:

- Educational demonstrations of DIP concepts.
- Preliminary analysis and processing in academic research.
- Real-world application scenarios like security, healthcare imaging, and heritage site identification.

Requirements: -

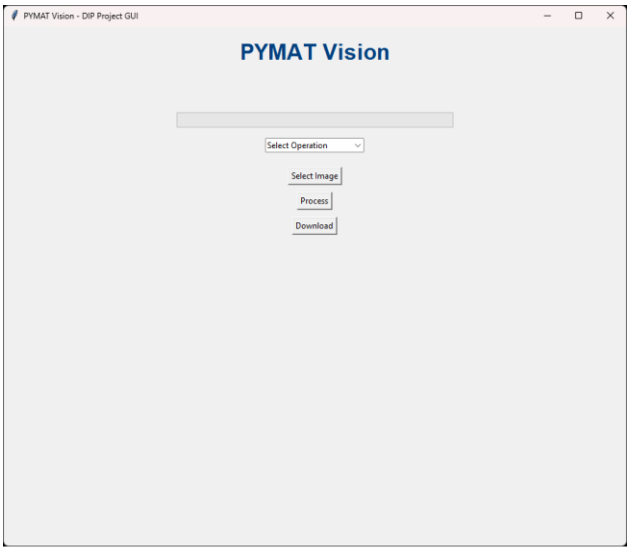
The project implementation requires:

- **Python 3.10+** with standard libraries such as tkinter, NumPy, OpenCV, and SciPy.
- **MATLAB (Prerelease Version 2025a)**, crucial for seamless integration with the latest Python releases.
- MATLAB's Image Processing Toolbox.
- MATLAB Engine API for Python to facilitate inter-language communication.

Graphical User Interface: -

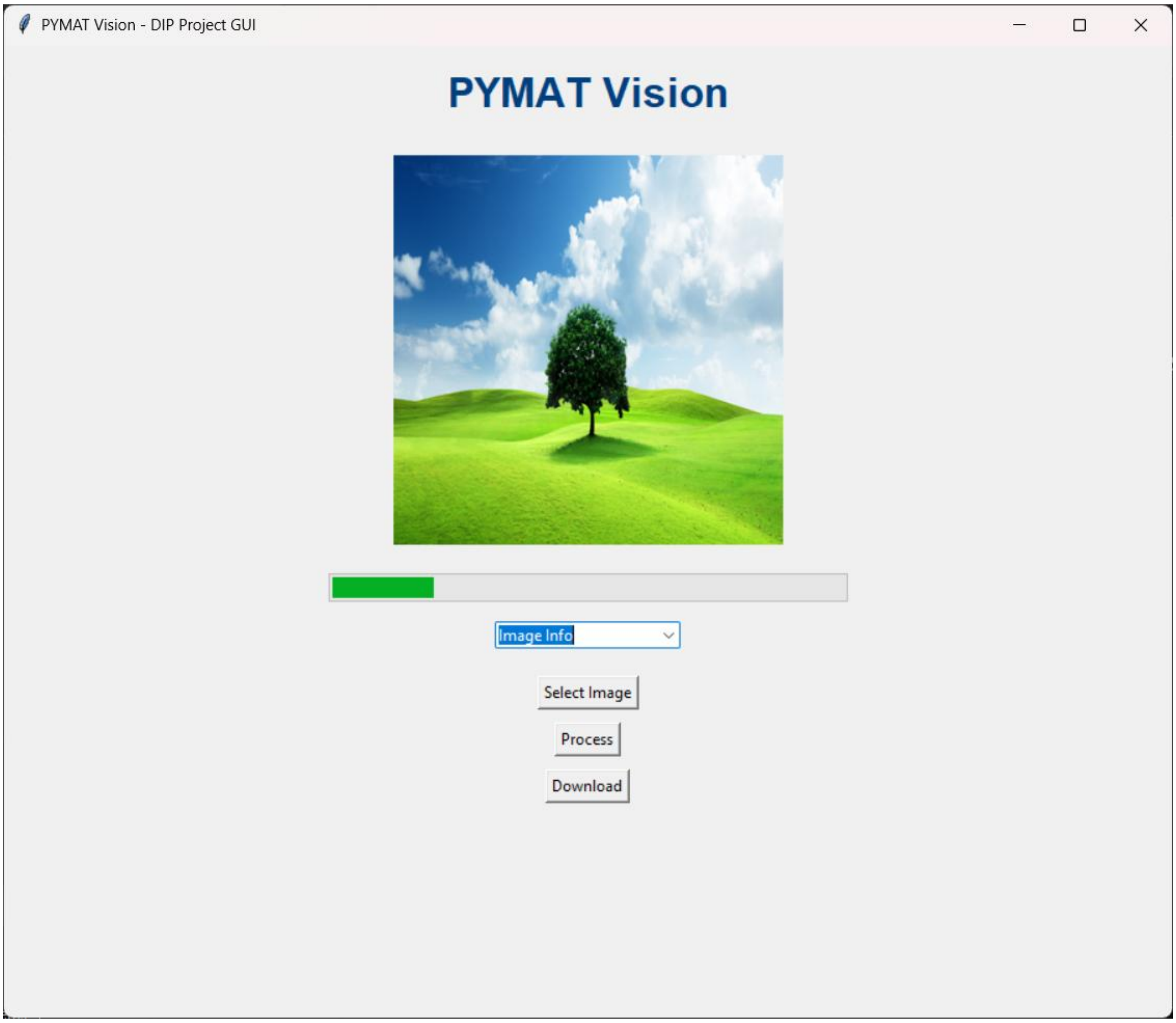
The GUI is structured around a logical workflow comprising:

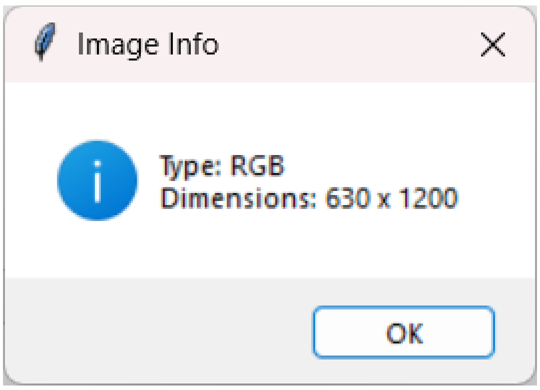
- **Input Stage**: Image uploading with automatic extraction of metadata (image type, dimensions).
- **Processing Stage**: Options including resizing, flipping, negative transformation, RGB to grayscale/binary conversion, histogram operations (generation and equalization), noise filtering (salt & pepper, median, averaging filters), global/local/adaptive thresholding, contrast stretching, bit-plane slicing, gray level slicing, edge detection, and feature extraction.
- **Output Stage**: Interactive visualization of processed images.
- **Download Stage**: Option to download processed images directly.



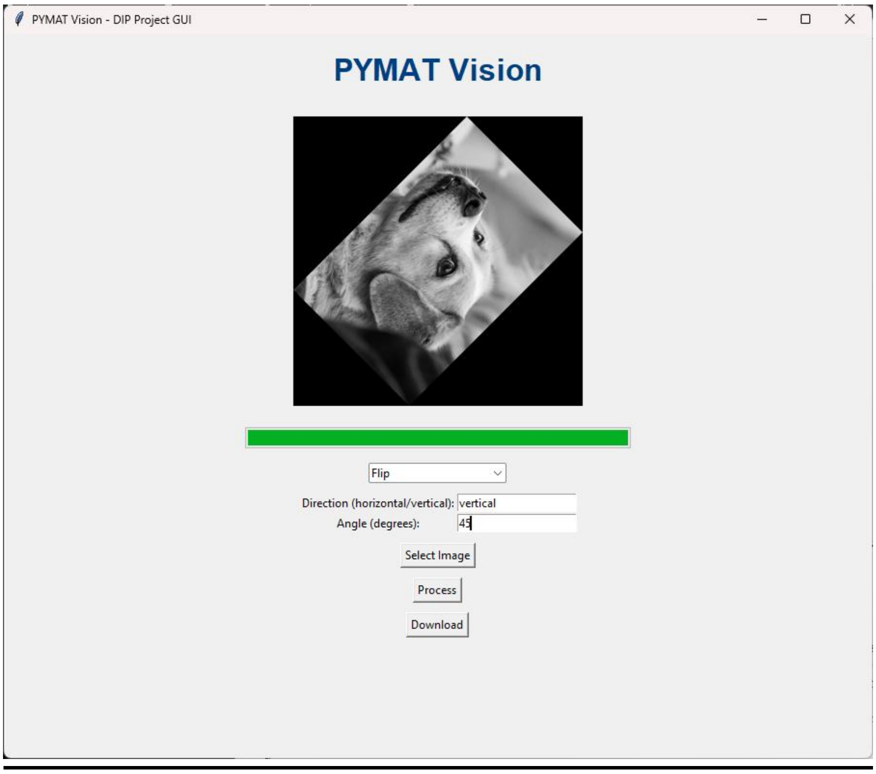
Testing Documents: -

Image Info:

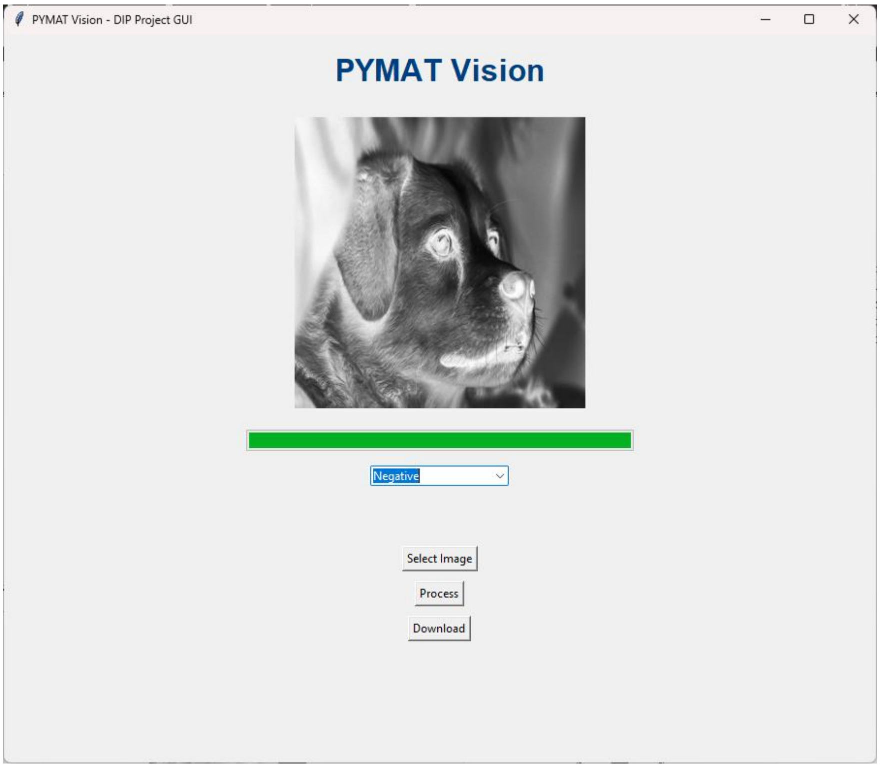




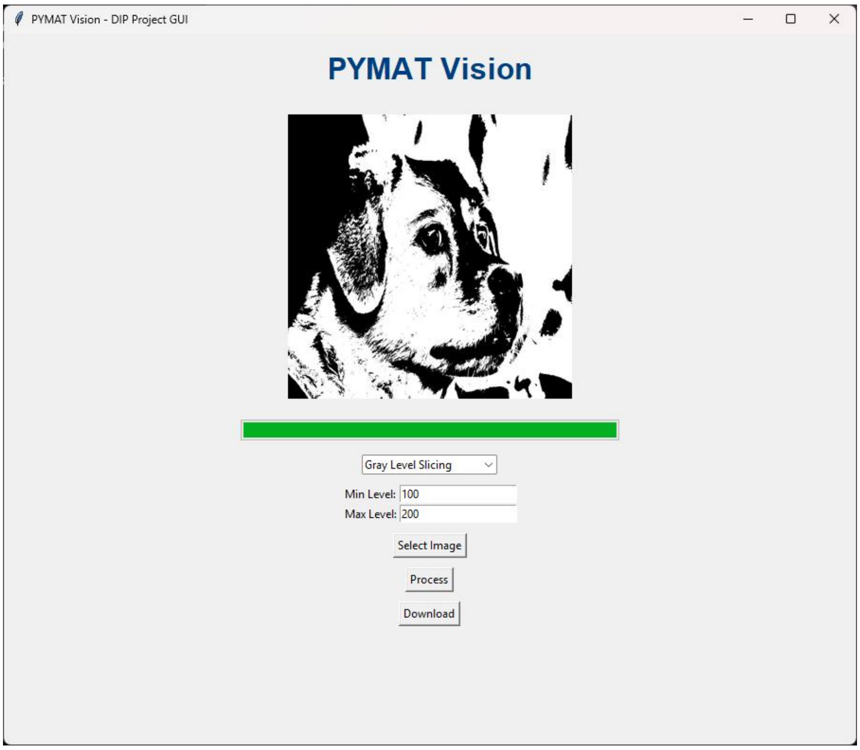
Flip:



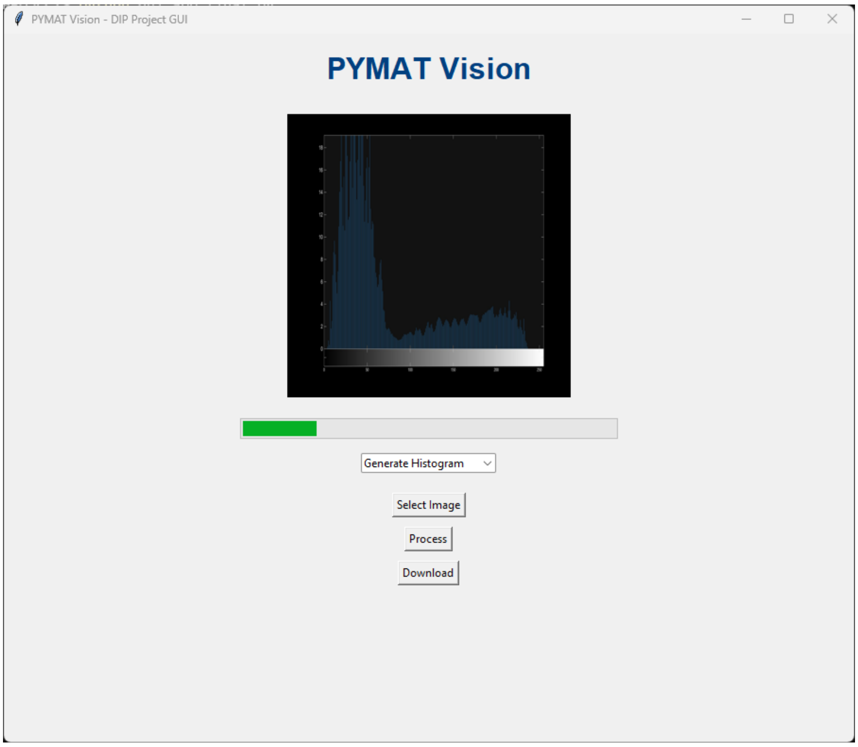
Negative:



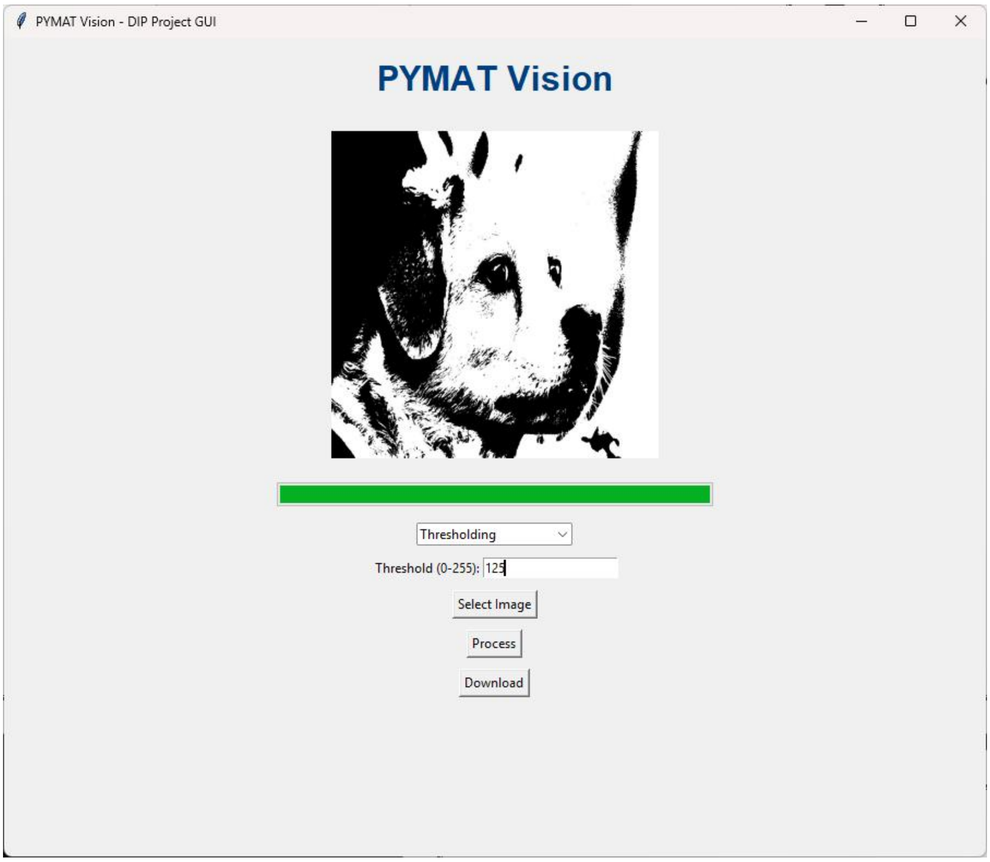
Gray Level Slicing:



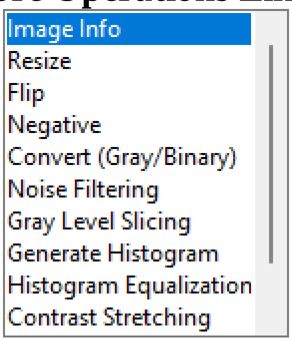
Histogram:

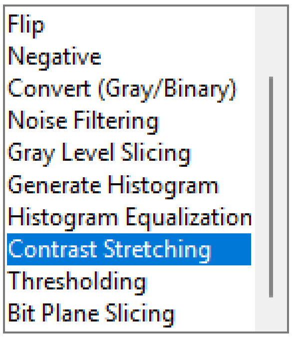


Thresholding:



More Operations Like:





Operations Performed:-

The PYMAT Vision platform enables the following digital image processing operations through an intuitive graphical interface:

- **Image Metadata Extraction:**
  Automatic retrieval and display of image properties such as format, dimensions, and bit depth.
- **Image Resizing and Flipping:**
  Dynamic image scaling and directional flipping (horizontal/vertical) to prepare images for further analysis.
- **Negative Transformation:**
  Generation of negative images to highlight contrast and enhance feature visibility.
- **Colour Space Conversions:**
  - RGB to Grayscale Conversion
  - RGB to Binary (Threshold-based) Conversion
- **Noise Reduction Filters:**
  Application of Median, Averaging, and Salt & Pepper noise removal techniques.
- **Histogram Operations:**
  - Histogram Generation (visualization of pixel intensity distributions)
  - Histogram Equalization (contrast enhancement)
- **Thresholding Techniques:**
  Implementation of global, local, and adaptive thresholding algorithms for image segmentation.
- **Contrast Stretching:**
  Enhancement of image contrast based on user-defined thresholds.
- **Gray Level Slicing:**
  Highlighting specific intensity ranges to focus on targeted image features.
- **Bit Plane Slicing:**
  Extraction and visualization of individual bit planes for in-depth image analysis.
- **Edge Detection and Feature Extraction:**
  Identification of object boundaries using gradient and threshold-based edge detection methods.
- **Processed Image Download:**
  Capability to save the processed image directly from the GUI for future reference or analysis.

Source Code:-

Python:

```python
import tkinter as tk

from tkinter import filedialog, ttk, messagebox

from PIL import Image, ImageTk

import matlab.engine

import threading

import os

# Start MATLAB engine

eng = matlab.engine.start_matlab()

eng.addpath(r'C:\Users\SOHAM\Documents\MATLAB\DIP_Project')  # Update as needed

selected_image_path = ""

output_image_path = "images/temp_imgs/output.jpg"

histogram_image_path = "images/temp_imgs/histogram.jpg"

root = tk.Tk()

root.title("PYMAT Vision - DIP Project GUI")

root.geometry("900x750")

# Project Title Label

title_label = tk.Label(

    root, text="PYMAT Vision", font=("Helvetica", 24, "bold"), fg="#004080"

)

title_label.pack(pady=15)

image_label = tk.Label(root)

image_label.pack(pady=10)

progress = ttk.Progressbar(root, orient='horizontal', length=400, mode='determinate')
```

```
progress.pack(pady=10)
operation = tk.StringVar()
operation.set("Select Operation")
operation_menu = ttk.Combobox(root, textvariable=operation, values=[
    "Image Info", "Resize", "Flip", "Negative", "Convert (Gray/Binary)",
    "Noise Filtering", "Gray Level Slicing", "Generate Histogram",
    "Histogram Equalization", "Contrast Stretching", "Thresholding", "Bit Plane Slicing"
])
operation_menu.pack(pady=5)
input_frame = tk.Frame(root)
input_frame.pack(pady=5)
input_entries = {}
def clear_inputs():
    for widget in input_frame.winfo_children():
        widget.destroy()
    input_entries.clear()
def show_inputs(*args):
    clear_inputs()
    selected = operation.get()
    if selected == "Resize":
        tk.Label(input_frame, text="Width:").grid(row=0, column=0)
        input_entries["width"] = tk.Entry(input_frame)
        input_entries["width"].grid(row=0, column=1)
        tk.Label(input_frame, text="Height:").grid(row=1, column=0)
        input_entries["height"] = tk.Entry(input_frame)
```

```
        input_entries["height"].grid(row=1, column=1)
    elif selected == "Flip":
        tk.Label(input_frame, text="Direction (horizontal/vertical):").grid(row=0, column=0)
        input_entries["direction"] = tk.Entry(input_frame)
        input_entries["direction"].grid(row=0, column=1)
        tk.Label(input_frame, text="Angle (degrees):").grid(row=1, column=0)
        input_entries["angle"] = tk.Entry(input_frame)
        input_entries["angle"].grid(row=1, column=1)
    elif selected == "Noise Filtering":
        tk.Label(input_frame, text="Type (salt_pepper/gaussian):").grid(row=0, column=0)
        input_entries["filter_type"] = tk.Entry(input_frame)
        input_entries["filter_type"].grid(row=0, column=1)
    elif selected == "Gray Level Slicing":
        tk.Label(input_frame, text="Min Level:").grid(row=0, column=0)
        input_entries["min_level"] = tk.Entry(input_frame)
        input_entries["min_level"].grid(row=0, column=1)
        tk.Label(input_frame, text="Max Level:").grid(row=1, column=0)
        input_entries["max_level"] = tk.Entry(input_frame)
        input_entries["max_level"].grid(row=1, column=1)
    elif selected == "Convert (Gray/Binary)":
        tk.Label(input_frame, text="Mode (gray/binary):").grid(row=0, column=0)
        input_entries["mode"] = tk.Entry(input_frame)
        input_entries["mode"].grid(row=0, column=1)
    elif selected == "Contrast Stretching":
        tk.Label(input_frame, text="Min Threshold (0-255):").grid(row=0, column=0)
```

```
        input_entries["min_thresh"] = tk.Entry(input_frame)
        input_entries["min_thresh"].grid(row=0, column=1)
        tk.Label(input_frame, text="Max Threshold (0-255):").grid(row=1, column=0)
        input_entries["max_thresh"] = tk.Entry(input_frame)
        input_entries["max_thresh"].grid(row=1, column=1)
    elif selected == "Thresholding":
        tk.Label(input_frame, text="Threshold (0-255):").grid(row=0, column=0)
        input_entries["threshold"] = tk.Entry(input_frame)
        input_entries["threshold"].grid(row=0, column=1)
    elif selected == "Bit Plane Slicing":
        tk.Label(input_frame, text="Bit Plane (0-7):").grid(row=0, column=0)
        input_entries["bit"] = tk.Entry(input_frame)
        input_entries["bit"].grid(row=0, column=1)
operation.trace('w', show_inputs)
def show_image(img_path):
    img = Image.open(img_path)
    img = img.resize((300, 300))
    tk_img = ImageTk.PhotoImage(img)
    image_label.configure(image=tk_img)
    image_label.image = tk_img
def browse_image():
    global selected_image_path
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg *.png *.jpeg
*.bmp *.tiff")])
    if file_path:
        selected_image_path = file_path
```

```
        show_image(selected_image_path)

def process_image():

    if not selected_image_path:

        messagebox.showwarning("No image", "Please select an image first.")

        return

    def run():

        progress['value'] = 20

        root.update_idletasks()

        try:

            op = operation.get()

            if op == "Resize":

                w = int(input_entries["width"].get())

                h = int(input_entries["height"].get())

                eng.resize_image(selected_image_path, output_image_path, w, h, nargout=0)

            elif op == "Flip":

                d = input_entries["direction"].get()

                a = float(input_entries["angle"].get())

                eng.flip_image(selected_image_path, output_image_path, d, a, nargout=0)

            elif op == "Noise Filtering":

                ftype = input_entries["filter_type"].get()

                eng.filter_noise(selected_image_path, output_image_path, ftype, nargout=0)

            elif op == "Gray Level Slicing":

                min_l = int(input_entries["min_level"].get())

                max_l = int(input_entries["max_level"].get())

                eng.gray_level_slicing(selected_image_path, output_image_path, min_l, max_l,
nargout=0)
```

```
    elif op == "Negative":
        eng.negative_image(selected_image_path, output_image_path, nargout=0)
    elif op == "Convert (Gray/Binary)":
        mode = input_entries["mode"].get()
        eng.convert_image(selected_image_path, output_image_path, mode, nargout=0)
    elif op == "Image Info":
        info = eng.get_image_info(selected_image_path, nargout=1)
        messagebox.showinfo("Image Info", info)
        return
    elif op == "Generate Histogram":
        eng.generate_histogram(selected_image_path, nargout=0)
        show_image(histogram_image_path)
        return
    elif op == "Histogram Equalization":
        eng.histogram_equalize(selected_image_path, output_image_path, nargout=0)
    elif op == "Contrast Stretching":
        min_t = int(input_entries["min_thresh"].get())
        max_t = int(input_entries["max_thresh"].get())
        eng.contrast_stretch(selected_image_path, output_image_path, min_t, max_t,
nargout=0)
    elif op == "Thresholding":
        thresh = int(input_entries["threshold"].get())
        eng.threshold_image(selected_image_path, output_image_path, thresh, nargout=0)
    elif op == "Bit Plane Slicing":
        bit = int(input_entries["bit"].get())
        eng.bit_plane_slicing(selected_image_path, output_image_path, bit, nargout=0)
```

```
        except Exception as e:
            messagebox.showerror("Error", str(e))
            return
        progress['value'] = 100
        root.update_idletasks()
        messagebox.showinfo("Done", "Processing complete.")
        show_image(output_image_path)
    threading.Thread(target=run).start()
def download_output():
    if not os.path.exists(output_image_path):
        messagebox.showwarning("No Output", "Please process an image first.")
        return
    save_path = filedialog.asksaveasfilename(defaultextension=".jpg")
    if save_path:
        img = Image.open(output_image_path)
        img.save(save_path)
        messagebox.showinfo("Saved", f"Saved to: {save_path}")
tk.Button(root, text="Select Image", command=browse_image).pack(pady=5)
tk.Button(root, text="Process", command=process_image).pack(pady=5)
tk.Button(root, text="Download", command=download_output).pack(pady=5)
root.mainloop()
```

MATLAB

bit_plane_slicing:

```
function bit_plane_slicing(input_path, output_path, bit)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    plane = bitget(img, bit + 1);  % MATLAB uses 1-based indexing
    result = uint8(plane) * 255;
    imwrite(result, output_path);
end
```

contrast_stretch:

```
function contrast_stretch(input_path, output_path, min_thresh, max_thresh)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    stretched = imadjust(img, [double(min_thresh)/255, double(max_thresh)/255], []);
    imwrite(stretched, output_path);
end
```

convert_image:

```
function convert_image(input_path, output_path, mode)
    input_path = convertCharsToStrings(input_path);
```

```
    img = imread(input_path);
    if strcmp(mode, 'gray')
        out = rgb2gray(img);
    elseif strcmp(mode, 'binary')
        gray = rgb2gray(img);
        level = graythresh(gray);
        out = imbinarize(gray, level);
    else
        error('Invalid mode. Use "gray" or "binary".');
    end
    imwrite(out, output_path);
end
```

filter_noise:

```
function filter_noise(input_path, output_path, filter_type)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if strcmp(filter_type, 'salt_pepper')
        noisy = imnoise(img, 'salt & pepper', 0.02);
        filtered = medfilt2(rgb2gray(noisy), [3 3]);
    elseif strcmp(filter_type, 'gaussian')
        noisy = imnoise(img, 'gaussian', 0, 0.01);
        h = fspecial('gaussian', [3 3], 0.5);
        filtered = imfilter(rgb2gray(noisy), h);
    else
        error('Unsupported filter type');
```

```
    end
    imwrite(filtered, output_path);
end
```

flip_image:

```
function flip_image(input_path, output_path, direction, angle)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if strcmp(direction, 'horizontal')
        flipped = flip(img, 2);
    elseif strcmp(direction, 'vertical')
        flipped = flip(img, 1);
    else
        error('Invalid direction');
    end
    rotated = imrotate(flipped, angle);
    imwrite(rotated, output_path);
end
```

generate_histogram:

```
function generate_histogram(input_path)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    figure('Visible','off');
```

```
    imhist(img);
    saveas(gcf, 'images/histogram.jpg');  % Save histogram image in output folder
    close;
end
```

get_image_info:

```
function info = get_image_info(input_path)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    dims = size(img);
    if ndims(img) == 2
        type = 'Grayscale';
    elseif ndims(img) == 3 && dims(3) == 3
        type = 'RGB';
    else
        type = 'Unknown';
    end
    info = sprintf('Type: %s\nDimensions: %d x %d', type, dims(1), dims(2));
end
```

gray_level_slicing:

```
function gray_level_slicing(input_path, output_path, min_level, max_level)
    input_path = convertCharsToStrings(input_path);
    img = rgb2gray(imread(input_path));
    sliced = uint8(zeros(size(img)));
    sliced(img >= min_level & img <= max_level) = 255;
    imwrite(sliced, output_path);
```

```
end
```

histogram_equalize:

```
function histogram_equalize(input_path, output_path)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    eq_img = histeq(img);
    imwrite(eq_img, output_path);
end
```

negative_image:

```
function negative_image(input_path, output_path)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    neg = 255 - img;
    imwrite(neg, output_path);
end
```

resize_image:

```
function resize_image(input_path, output_path, new_width, new_height)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    resized = imresize(img, [new_height, new_width]);
    imwrite(resized, output_path);
end
```

threshold_image:

```
function threshold_image(input_path, output_path, thresh)
    input_path = convertCharsToStrings(input_path);
    img = imread(input_path);
    if size(img, 3) == 3
        img = rgb2gray(img);
    end
    bw = imbinarize(img, double(thresh)/255);
    imwrite(bw, output_path);
end
```

Advantages:-

- **Seamless Hybrid Integration:**
  By combining Python's flexibility and MATLAB’s computational robustness, the system delivers superior processing speed and accuracy.
- **User-Centric Design:**
  Designed for ease of use, even for users with minimal programming or technical expertise.
- **Extensive Functional Coverage:**
  Supports a wide array of fundamental and advanced image processing operations within a single platform.
- **High Scalability:**
  Future-proof architecture capable of integrating additional features like cloud deployment and machine learning-based enhancements or even any additional operations.
- **Educational Impact:**
  Serves as a bridge between theoretical knowledge and practical skills, ideal for academic institutions and learners.
- **Cross-Platform Compatibility:**
  Built with consideration for future extensions into mobile, web, and cross-platform environments.

Applications:-

- **Healthcare Diagnostics:**
  Pre-processing of medical images (e.g., X-rays, MRIs) for improved analysis and diagnosis.
- **Security and Surveillance:**
  Image enhancement and feature extraction in CCTV footage for threat detection and forensic applications.
- **Heritage and Cultural Site Preservation:**
  Assists in digital documentation and analysis of historical artifacts and monuments.
- **Educational Tools:**
  Hands-on demonstrations for academic courses in Digital Image Processing, Machine Learning, and Artificial Intelligence.
- **Industrial Automation:**
  Quality control processes via machine vision systems for defect detection and product inspection.
- **Research and Prototyping:**
  Rapid development and validation of image processing algorithms for research projects.
- **Mobile Application Development:**
  Potential extension into mobile platforms for real-time image processing on smartphones and tablets.

Future Enhancements:-

Corporate reports *always* emphasize scalability and innovation potential.
You could include a formal subsection outlining **planned improvements**, like:

- Integration of **Machine Learning models** for automatic image classification.
- Adding **real-time video processing** capabilities alongside static images.
- **Mobile App Version** leveraging lightweight Python-MATLAB servers.
- Incorporating **Cloud-Based Storage** and **Cloud Processing** (e.g., AWS, Azure).
- Adding **Augmented Reality (AR) overlays** for processed images.

Conclusion: -

The successful integration of Python and MATLAB creates an efficient, versatile, and user-friendly platform for digital image processing tasks. This hybrid approach not only simplifies the use of complex DIP operations but also serves as a valuable educational and research tool. Future enhancements could include cloud integration, expanded machine learning capabilities, and mobile compatibility, further extending its applicability and accessibility.