Basic Image Editor

Tushar Nandy
Dept. of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, India
190020125@iitb.ac.in

***Abstract*—This is a report on a basic GUI editor and includes the information on the GUI layout, image processing operations and their effects on various images.**

I. Introduction

Image editing is a procedure in which various image processing operations are applied to an image in order to enhance it. In this assignment, the transformations used were intensity transforms and spatial filtering. Apart from the prescribed set of operations, control and enhancement of brightness and contrast has also been added and shown in the following section.
The layout of the rest of the report is a description of the GUI Design, mathematical details of the image transformation techniques, results of these on some images, and finally conclusion and discussion.

II. GUI Design

Graphical user interfaces have become the standard of user-centered design in software application programming, providing users the capability to intuitively operate computers and other electronic devices through the direct manipulation of graphical icons such as buttons, scroll bars, windows, tabs, menus, cursors, and the mouse pointing device. The GUI for this assignment is designed using tkinter [1] library, which is a part of the python standard library and includes a plethora of widgets such as buttons, text fields, sliders, drop-down boxes, file dialog boxes, etc. Apart from providing a simplistic interface, an image editing GUI provides the advantage of trying and testing several image processing operations to enhance an image. For this assignment, the GUI window was divided into frames, with each frame handling the layout of a different category of widgets.

```
window = tk.Tk() # creating a GUI window

# fetching the screen width and height
screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()

# setting the size of the window to full-screen
window.geometry(f"{screen_width}x{screen_height}")

# Frame 1 handles single-press buttons
Frame1 = tk.Frame(window, height=20, width=200)
Frame1.pack(anchor=tk.N)

# Frame 2 consists of sliders
Frame2 = tk.Frame(window, height=20,
                  width=screen_width)
Frame2.pack(anchor=tk.NW)

# Frame 3 consists of text fields for entry
Frame3 = tk.Frame(window, height=20)
Frame3.pack(anchor=tk.NW)
```

Listing 1. Python example

A. Frame 1

Actions such as 'undo all' do not require any textual/numeric input from the user, while 'import', 'save' and 'close' need inputs via file-dialog boxes. Additionally, image operations such as histogram equalization and log-transform do not involve any parametric specifications from the user. These actions can then be implemented using buttons. For designing buttons in tkinter, it is required to initialize a button object and pass a callback function which handles the operation when a button is clicked. Tkinter involves file dialog-box function which allows the user to import from or save to the desired location. The undo all button simply updates the current image with the original image, thereby erasing all enhancements. The undo action, on the other hand displays the image before the current operation was performed. An advanced application of undo, which spans several steps of previous enhancements could be implemented using stacks. Histogram equalization and log-transform operations are performed similar to the previous actions, with the exception that they update the current-image and previous-image variables after operating on the image. The functions for operation are fetched from a different file and are not written in the callback function, as will be discussed in the next section. Following is an example of how buttons and their corresponding callbacks are written in tkinter.

```
# Import button will be used to open image files for
     editing
importButton = tk.Button(Frame1, text="import",
                         padx=10, pady=5,
                         command=import_callback)


# location of the button
importButton.grid(row=0, column=0)

def import_callback():
   # Callback function corresponidng to the import
    button
    global image_og   # stores original image
    global image_curr    # stores the current image
```

```
    global image_prev    # stores the previous image
    global display_image # stores the display image

    # fetching filename from dialog box
    filename = filedialog.askopenfilename()

    # importing the image in rgb format with pixels
    [0..255]
    image_og = image.imread(filename)

   # png images are usually in RGBA format with
    pixel values in [0..1]
   if filename.split(".")[-1] =="png":
       # checking the extension

       # extracting only the RGB layers
        image_og = image_og[:, :, :3]
        # converting to [0..255]
        image_og = (image_og*255).astype("uint8")

    image_curr = image_og # assigning the current
    image
    image_prev = image_og # assigning the previous
    image

   # rgb cannot be directly passed to tkinter
    objects
    display_image = Image.fromarray(image_curr) #
    coverting RGB to PIL image format
    displayImage(display_image) # displaying the
    image on the GUI



def displayImage(display_image):
   # a wrap-around function for displaying images on
     the GUI
    ImagetoDisplay = display_image.resize((900,600),
     Image.ANTIALIAS) # resizing the image
   # displaying the image on the GUI
    ImagetoDisplay = ImageTk.PhotoImage(
    display_image)
    showWindow.config(image=ImagetoDisplay)
    showWindow.photo_ref = ImagetoDisplay
    showWindow.pack()
```

Listing 2. Buttons code block

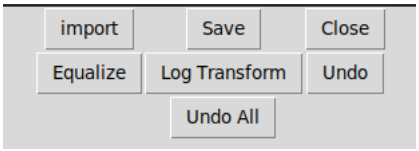

Fig. 1. Layout of buttons

B. Frame 2

This frame consists of sliders for numerical inputs to sharpening and blurring. Sliders are widgets consisting of a cursor which can slide between a specified minimum and maximum for a given resolution. Sliders need to be accompanied by buttons which can implement the operation after the input values have been selected. This involves writing two callback functions. One for reading the value on the slider and another for implementing the action after a user presses the button.

```
# A slider to select the sigma for blurring
blurSlider = tk.Scale(Frame2, label="Select Sigma",
                              from_=0, to=5,
                              orient=tk.HORIZONTAL,
                              length=screen_width/2,
                              resolution=0.1,
                              command=blur_callback)
blurSlider.pack(anchor=tk.N)

# Button for implementing blurring based on the
    selected gamma
blurButton = tk.Button(Frame2, text="Blur", command=
    apply_blur)
blurButton.pack()

def blur_callback(pos):
    # callback for reading the position
    # of the slider
    global sigma
    sigma = float(pos)

def apply_blur():
    global image_og
    global image_curr
    global image_prev
    global display_image
    global sigma

    image_prev = image_curr # updating previous
    image variable
    image_curr = gaussian_blur(image_curr, sigma=
    sigma) # blurring the current image

    # displaying the current image
    display_image = Image.fromarray(image_curr)
    displayImage(display_image)
```

Listing 3. Slider code block

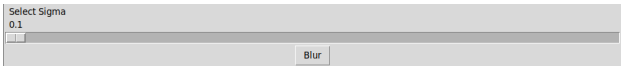

Fig. 2. Layout of slider

C. Frame 3

Frame 3 consists of text fields for entering the values for parameters such as α and β for contrast and brightness, or for entering the value of gamma for gamma correction. The text fields take the value as string and need to be converted to float. Like the sliders, these too need to be accompanied with buttons to implement the action after a value has been entered.

```
# defining a tkinter string for storing gamma
gamma_input = tk.StringVar()

# a label displayed above the text field
gamma_entry = tk.Label(Frame3, text="Enter Gamma")
gamma_entry.grid(row=0, column=0)

# creating a text field to take input
gammaEntry = tk.Entry(Frame3, bd=7,
                      textvariable=gamma_input)
gammaEntry.grid(row=1, column=0)

# creating a button to implement gamma correction
correctButton = tk.Button(Frame3,
                          text="Gamma Correct",
                          padx=10, pady=5,
                          command=gamma_callback)
```

```
correctButton.grid(row=2, column=0)
```

Listing 4. Field code block

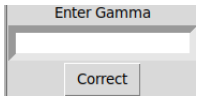

Fig. 3. Layout of text field

III. Image Processing Operations

A. Histogram Equalization

This method [2] is used to increase the global contrast of an image when represented by a narrow range of intensity values. As the name suggests, an ideal implementation of the method should result in a flat histogram (all bars at a constant value) when representing the count of pixels. In practice, parts of the histogram which are dense are rarefied and vice-versa. Mathematically, the method first involves calculating the probability of a pixel to have a certain intensity value 'i' in the respective range.

$$P_i = \frac{n_i}{N}$$

Where $0 \leq i < L$ and L is the maximum intensity level (typically 256). P_i is the probability that a pixel has intensity 'i'. n_i is the number of pixels with intensity 'i' and N is the total number of pixels.
Then, it is required to calculate the cumulative distribution function.

$$cdf_i = \sum_{0 \leq j \leq i} P_j$$

For each pixel with original intensity 'i', the new normalized intensity value is given by cdf_i. To denormalize, the values are multiplied by L and are usually rounded-off to the nearest integer.

An example of histogram equalization is shown in fig 4. and fig 5.

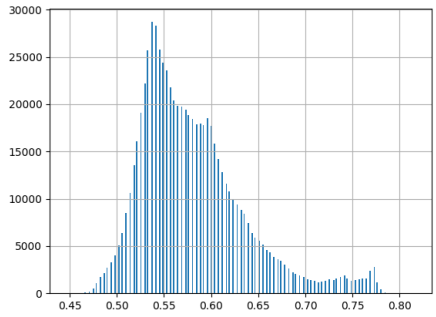

Fig. 4. Unequalized Histogram

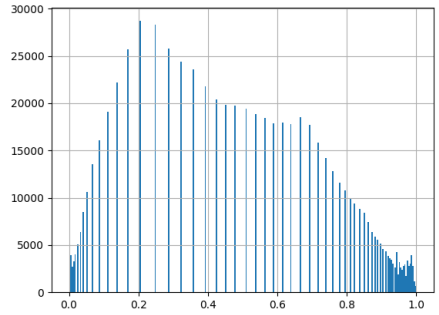

Fig. 5. Equalized Histogram

B. Log Transform

Log transform [3] is another intensity transformation method, using which the pixel values are mapped to higher intensities. The effect is more for pixels at lower intensity values and is reduced for pixels which originally had high intensity values.

$$\text{T(r)} = c\log(1 + \text{r})$$

Where $0 \leq r \leq L$ and c is a normalization constant and is chosen such that pixels with value L remain unchanged. Substituting the value of 'c' would result in:

$$\text{T(r)} = \frac{L}{\log(1+\text{L})} \times \log(1 + \text{r})$$

C. Gamma Correction

This method of transformation [4] is used to map values of pixels to either higher levels, or lower. The method works by raising the current values to an exponent (γ). Depending on whether gamma is greater or lower than 1, current values are mapped to lower or higher intensities, respectively.

$$\text{T(r)} = c \times r^{\gamma}$$

As in the case of log-transform, the value of 'c' is such that pixels at level L remain unchanged.

D. Gaussian Blur

Gaussian filter [5] is a discrete-space low-pass filter which removes high frequency components like edges, noise, scars or wrinkles. This results in blurring of the image. Blurring an image is a kernel operation. For this assignment, zero-padding is used for blurring as well as sharpening an image. Gaussian blur is an odd-sized filter with each element having the value:

$$W(m,n) = \frac{1}{2\pi\sigma^2} \times e^{-\frac{(m-m_o)^2+(n-n_o)^2}{2\sigma^2}}$$

Where m_o, n_o is the center of the grid. For colored images, this is either applied to each channel (R, G and B) or to the 'V' matrix after converting to HSV format. A gaussian filter of size 3 with $\sigma = 5$ is shown as follows:

$$\begin{bmatrix} 0.1096297 & 0.11184436 & 0.1096297 \\ 0.11184436 & 0.11410377 & 0.11184436 \\ 0.1096297 & 0.11184436 & 0.1096297 \end{bmatrix}$$

If the size is predetermined, then σ can be used to control the extent of blurring. Higher values of sigma result in more blurring and vice-versa.
In the code, sigma is selected using a slider with values ranging between 0.1 and 10. The resolution is 0.1. The choice of this range is purely arbitrary.

E. Sharpening

Sharpening is a method of enhancing high-frequency components (such as edges) and suppressing slow-varying intensity regions. The Laplacian operator [6] is one such filter which emphasizes high frequency components by taking the derivative of intensity values. The filter is usually 3×3 in size.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

By applying the filter to an image, the result is a matrix with suppressed continuous regions and highlighted edges. To obtain a sharpened image, this resulting matrix is added to the original image. For RGB images, this operation is applied to the 'V' matrix after conversion.

$$V_{sharpened} = V_{original} + \eta V_{lap}$$

Where V_{lap} is the operated matrix. The factor η controls how much of the enhanced edges should be added back to the original image. The choice of η is arbitrary and is usually selected after trying several values. η can be greater than 1, apart from being a fraction in most cases. This requires saturating the intensity values (at 0 or L, as the case may be) so that the range is not exceeded.

F. Brightness and Contrast

The simplest linear operations that can be applied to an image are *brightening* [7] and increasing its *contrast* [7].
Contrast is achieved by multiplying each of the RGB channels by a factor $\alpha > 0$. This changes the intensity values of all pixels in proportion. Brightness is controlled by adding a bias term $beta$ to the pixel values. Beta can have both +ve and -ve values depending upon whether the image has to be brightened or darkened.

$$\text{T(r)} = \alpha r + \beta$$

In the GUI, however, brightness and contrast are implemented as two different operations, although one can choose to combine them.

IV. Results

Following are some examples of the operations discussed above.

A. Histogram Equalization

Figures 6 and 7 are images of original and equalized image. Figures 4 and 5 show the histogram for this image before and after equalization.

Fig. 6. Unequalized Image

Fig. 7. Equalized image

B. Log-Transform

The effect of log-transform is visible in fig. 8 and fig. 9 Additionally, the effect of gamma correction for γ = 0.6 is shown in fig. 10.

Fig. 8. Original Image

Fig. 9. Log Transform

Fig. 10. Gamma = 0.6

C. Gamma Correction

Gamma correction can also be used on over-exposed images, as shown in fig. 11 and 12.

D. Blurring

Fig. 13 and 14 show the effect of blurring. Blurring is applied to 'V' matrix after converting to HSV, and then transformed back to RGB. For this image, the kernel size is 51 X 51 and sigma = 5.

E. Sharpening

Images in fig 15 and 16 show the effect of sharpening using the laplace operator. The value of η is 5 in this case.

V. Conclusion

The results show the advantage of applying image transform techniques which are capable of highlighting important information and enhancing the image using simple mathematical operations to an image. There is no method set in stone which has to be followed to enhance an image, and the wisdom to know what transformation to apply comes largely from experience.

Next steps in this direction would be to apply histogram specification techniques such as CLAHE to images.

Fig. 11. Over-exposed image

Fig. 12. Gamma = 4

References

[1] https://docs.python.org/3/library/tkinter.html

[2] P. E. Trahanias and A. N. Venetsanopoulos, "Color image enhancement through 3-D histogram equalization," in Proc. 15th IAPR Int. Conf. Pattern Recognition, vol. 1, pp. 545–548, Aug.-Sep. 1992

[3] Gonzalez, R & Woods, R. (1977). Digital Image Processing.

Fig. 13. Original

Fig. 14. Blurred

Fig. 15. Original

[4] Z. Wang, J. J. Cao, L. K. Zhang, and J. Zhang, "Measurement and evaluation for profile on images," Optics and Precision Engineering, vol. 17, pp. 395-401, Feb. 2009.
[5] Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001
[6] Gonzalez, R & Woods, R. (1977). Digital Image Processing.
[7] https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html

Fig. 16. Sharpened