Task-1

Summary

In this task we have to capture an image using Webcam and write python functions for

- Converting the image to the HSV color space
- Applying histogram equalization for contrast enhancement
- Apply binary inversion thresholding
- Reducing the image to 4 gray intensity levels
- Applying Laplacian and Scharr filter
- Applying median filter
- Applying unsharp masking for sharpening
- Converting the image from RGB to LAB color space

Capturing the image using OpenCV

- For me cv2.VideoCapture(0) was working so and the VideoCapture(0) accesses the default webcam
- cap.read() used to capture one frame
- cv2.imwrite() and it saves for further processing

Converting to HSV Color space

cv2.cvtColor(img,cv2.COLOR_BGR2HSV) in this we can see it will convert the image from BGR to HSV.

It is BGR because OpenCV loads images in BGR format but not in RGB

cv2.split(hsv) this one splits the 3-channel HSV image into three separate grayscale images

Channel type	What does it do
Hue	shows dominant colors
Saturation	shows how pure the color is
Value	shows brightness levels

Histogram Equalization

Histogram of a grayscale image shows how many pixels have each intensity value from 0 to 255

Like it will redistribute pixel intensities to cover a wider range (0 to 255)

Like making dark areas darker and bright areas brighter which will the hidden details like it helps in edge detection, feature extraction..

For converting the image to Grayscale **cv2.COLOR_BGR2GRAY**

Applying histogram equalization to enhance contrast by using **cv2.equalizeHist(gray)**

Binary inversion

General form of cv2.threshold is **cv2.threshold(src , thresh, maxval, type)**

Parameter	Description
Src	input image (must be grayscale)
thresh	Threshold value to compare pixel intensity
maxval	Value to assign if the condition is satisfied
type	binary /inverse/truncation..

for us the syntax would be **cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)**

Like in this each pixel has value 0 to 255 and if the pixel intensity is greater than 127 it will be converted to zero and if it is less than or equal to 127 it will be converted to 255 and 255 is the maximum value assigned to pixels that meet the condition in thresholding.

Using this we have benefit of detecting the objects nicely like it highlights the dark regions and this is also used in Text segmentation like text is often dark on light background and this thing inverts it

Posterization

posterized = (gray // 64) * 64

It reduces the number of gray levels to 4

Edge detection using Laplacian and Scharr Filters

Laplacian detects the rapid intensity changes like edges it is a **second order derivative filter** it calculates the second order derivative of the image and it detects the regions of rapid intensity change

And it is isotropic as it responds equally to edges in all directions

I used CV_64F because as the derivative s can result in negative values and uint cannot store negatives so we use a floating point format to preserve the result

used **cv2.Scharr(gray,cv2.CV_64F,1,0)** to detect vertical edges by travelling in the X-direction.

and to detect the horizontal edges i used (0,1) instead of (1,0).

Median Filtering

noisy=gray.copy() i used this to create a copy of my gray scale image so that the original one do not change and added noise to this manually

noisy[::10,::10]=0 selects every 10th row and 10th column and sets the pixel at that to 0(black)

noisy[::15,::15]=255 similarly it selects the 15th row and 15th column and makes the pixel to 255

Unsharp Masking

This is to enhance the edges and fine details and it is done by using unsharp masking which sharpens by subtracting a blurred version from the original

blurred=cv2.GaussianBlur(gray,(5,5),0) less detail and no sharp edges.

here gray is input grayscale image and (5,5) is the kernel size and the kernel size should be odd because it allows the kernel to be centered on a single pixel and this central pixel becomes the focus for calculating the new value

sharpened=cv2.addWeighted(gray,1.5,blurred,-0.5,0)

sharpened=1.5 x original - 0.5 x blurred +0 it is like adding back the detail that was removed by the blur and this helps in sharpening the edges

$$Unsharpmask formula: sharpened = original + amount(original - blurred)$$

LAB Color space

It is powerful for color-based processing like color enhancement , object segmentation.

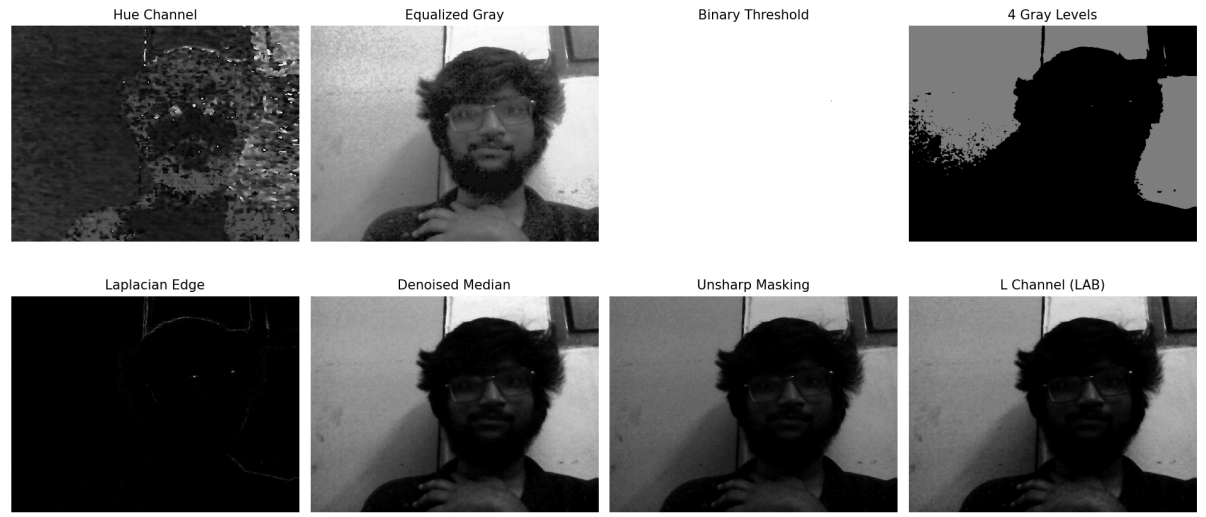
Channel	Meaning	Value range
L	Lightness	0 to 255
A	Green ↔ Red	0 to 255(Centered at 128)
B	Blue ↔ yellow	0 to 255(Centered at 128)

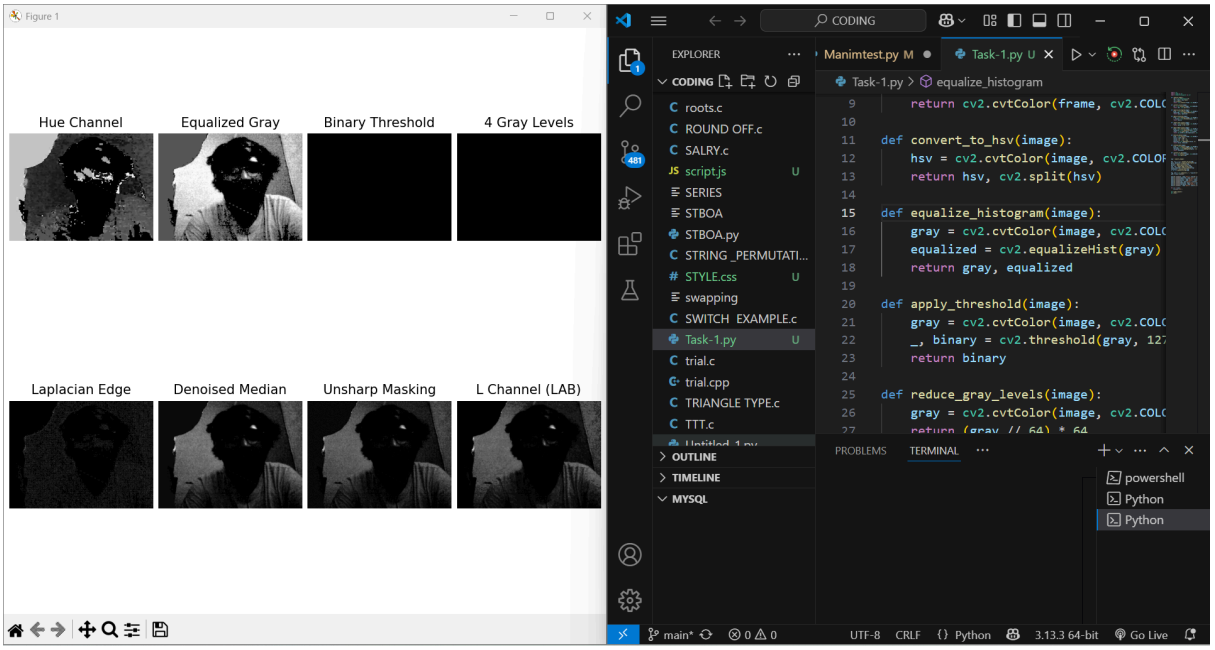
lab=cv2.cvtColor(img,cv2.COLOR_BGR2LAB)

Conversion from BGR to LAB

L,A,B=cv2.split(lab)

This is used to separate the 3 LAB channels





Link for the code

https://github.com/CODER-7777/GanForge/blob/vivekananda_m/Task-1(code)