CARNEGIE MELLON UNIVERSITY

(CMU AFRICA, KIGALI)

Name: Pierre Ntakirutimana
Course: Applied Computer Vision

Assignment:2
Date: February 3, 2024

PART 1

Introduction

In this report, we explore various image processing techniques using Python and OpenCV. We start by loading an RGB image, convert it to grayscale, perform histogram equalization, and apply edge detection using Sobel filters and the Canny algorithm.

1. BGR to RGB Conversion

To convert the BGR image to RGB, we create a function `bgr2rgb` using only NumPy. We display the original RGB image alongside the original BGR image for comparison.

Figure 1: Original RGB and BGR Images

We can see that the RGB image is more visible showing all the colors for our eyes comfort.

2. RGB to Grayscale Conversion

We implement the RGB to grayscale conversion using the average method. The resulting grayscale image is displayed.

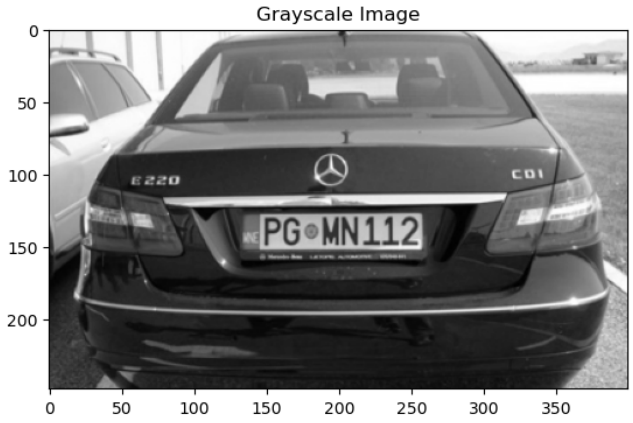

Figure 2: Grayscale Image

This grayscale image has different dimensions and it is simple to do calculations on it.

Histogram Equalization

We compute the histogram of the grayscale image, calculate the cumulative distribution function (CDF), and perform histogram equalization. The original image, its histogram, and the equalized image are displayed in figure 3 below.

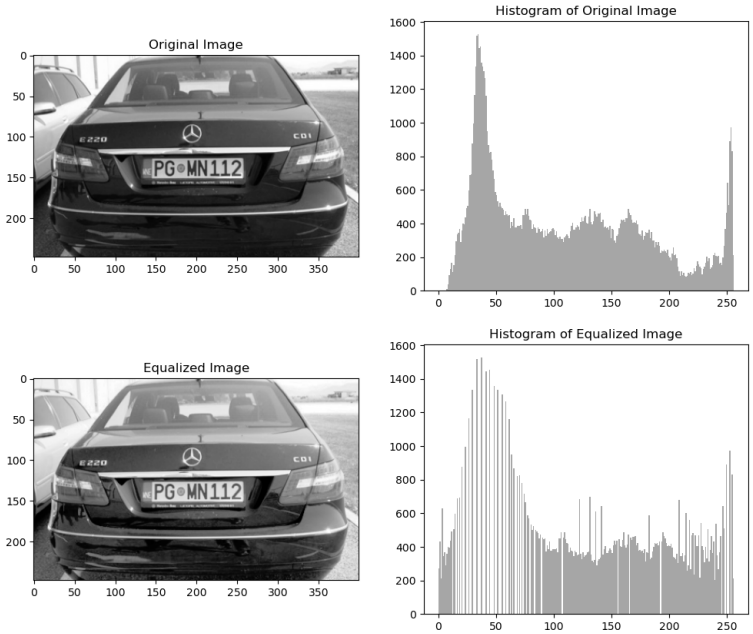

Figure 3: Original Image, Histogram, Equalized Image, and Equalized Histogram

The images have changes in brightness, and colors are highly infersided on the image. The histogram of equalised images appears to be uniform.

Edge Detection

We use Sobel filters to perform edge detection and display the resulting Sobel edges. Additionally, we apply the Canny algorithm for edge detection and compare it with the Sobel edges. In figure 4, a Sobel edge detection image is created, and a canny image is created with a minimum threshold of 100 and a maximum threshold of 255. Both images are shown together to make the comparison easier.

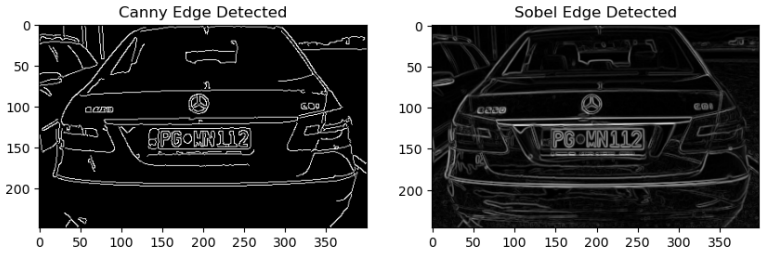

Figure 4: Sobel and Canny Edge Detection

Looking at the images in figure 4, Canny image [left], shows the edges clearly. It doesn't have big lines and it can be adjusted to pick edges with desired thresholds.

Binary Image from Sobel Edges

Finally, we create a binary image from the Sobel edges by applying a threshold manually. The threshold was set to be 120 and we changed the pixel below 120 to 0 and 255 otherwise. The resulting binary image is displayed in figure 5.

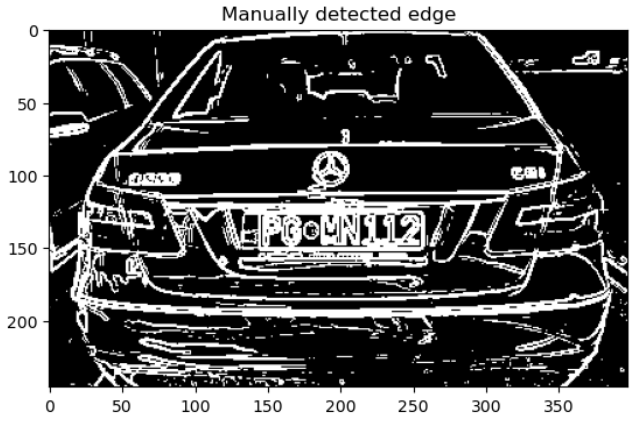

Figure 5: Binary Image from Sobel Edges

The image in figure 5, shows that the image captured noise, which means that the threshold of 120 in Sobel edge detected image is a small number. It would be increased to capture the edges only.

PART 2

1. Introduction

In this part of the assignement, we needed to verify the relationship between the RGB and Y space as stated in the OpenCV documentation [1] that says that an RGB image can be changed to gray scale by using the equation 1 below;

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

With R representing the pixel value of red, G representing green pixel value and B representing the blue pixel value, The primary data structure in this whole part were lists.To do this, we solved a linear system of equations. Since matrices play an important role in image processing and computer vision,we accomplished the task by computing the inverse of the RGB matrix in the equation below:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} R_1 & G_1 & B_1 \\ R_2 & G_2 & B_2 \\ R_3 & G_3 & B_3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

where x_1, x_2, and x_3 are the coefficients of the R, G, and B pixel values as seen in Equation 1; Y_1, Y_2, and Y_3 are grayscale values, and R_i, G_i, and B_i represent the red, green, and blue pixel values of the i-th row in the above equation. Solving for x_1, x_2, and x_3 will require finding the inverse of the matrix below:

$$\begin{bmatrix} R_1 & G_1 & B_1 \\ R_2 & G_2 & B_2 \\ R_3 & G_3 & B_3 \end{bmatrix}$$

using Gauss-Jordan elimination and cofactors, and verifying relationships between grayscale and RGB matrices.

Read Images

Colored, floating-point grayscale, and quantized grayscale images are loaded from JSON files using the `json.load()` method. RGB matrices and grayscale vectors are defined based on the loaded images, providing a foundation for subsequent operations.

a.

The reason why pixel 2 cannot be used along with pixel 1 is because they are the same. One is a multiple of another (factor of 1) and these would cause them to be dependent to each other. If we use both of them together, the matrix will be singular, and we wont have a way to calculate the inverse. The determinant of this matrix will be zero.

b. Finding the Inverse of a 3×3 Matrix

The `inv_gauss_jordan` function is implemented to find the inverse of a 3×3 matrix using Gauss-Jordan elimination. A `CofactorInv3` class is introduced for finding the inverse using cofactors, including methods for transposing a matrix, getting the minor of a matrix, calculating

the determinant, calculating the cofactor, and finding the inverse. The codes are shared along with the submission.

c. Verification and Relationship Analysis

Verification steps are included to calculate the check if solving for values of x we will get a vector $[0.299\ 0.587\ 0.114]^T$ from the inverse we got and the quantized image values. For quantized grayscale values, both inverses gave us very closed vectors, where the difference comes after 10 significant digits. We concluded that they are the same, and the vectors are [0.5942028985507477, 0.20289855072462615, 0.10144927536231307]. For unquantized images, we also got pretty similar vectors, which are [0.11400792218637434, 0.5869932312896253, 0.2989974091018581] and this was in BGR fromat as images were read in openCV and it used BGR format [2]. This mean that we solved the probelm and both methods are accurate.

e. Conversion to Grayscale

We used the findings to convert an RGB image to grayscale. The resulting grayscale values are saved to a JSON file to be opened by the codes we are provided with the questions. The real image and the grayscale images are shown in figure 6.

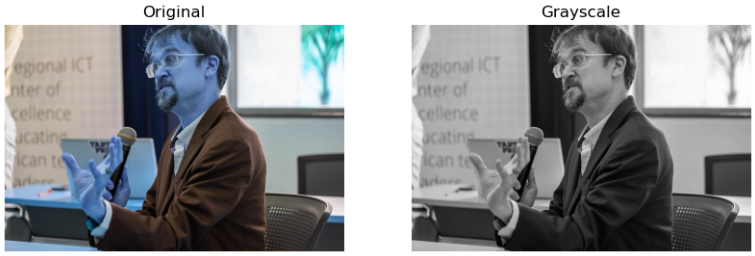

Figure 6: Comparison of CV2 grayscale image and our Algorithm

It is seen that the image is grayscale, like the image we were given is assignment questions. We conclude that the target has been well achieved.

2.

For a 4x4 matrix, both methods are feasible, but the Gauss Jordan method is generally preferred due to its speed [3]. The cofactor method directly computes the inverse without the need for extensive row operations but it uses several calculations. The Gaus Jordan methid does not involve calculating the determinants.

3. Image Quality

The grayscale in figure 6 appears to be like **Figure 2** in the question paper. We conclude that the target has been well achieved. The provided code serves as a comprehensive tool for matrix-related calculations and image processing. With the completion of TODOs and potential improvements, it can be concluded that this is the actual algorithm that openCv use.

References

[1] "OpenCV: Color conversions." [Online]. Available: https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

[2] "Why OpenCV Using BGR Colour Space Instead of RGB - OpenCV Q&A Forum." [Online]. Available: https://answers.opencv.org/question/6515/why-opencv-using-bgr-colour-space-instead-of-rgb/

[3] "Gaus -jordan elimination compared to cofactor method." [Online]. Available: https://www.physicsforums.com/threads/gaus-jordan-elimination-compared-to-cofactor-method.356383/