CS 5330 Project 1

Kevin Heleodoro

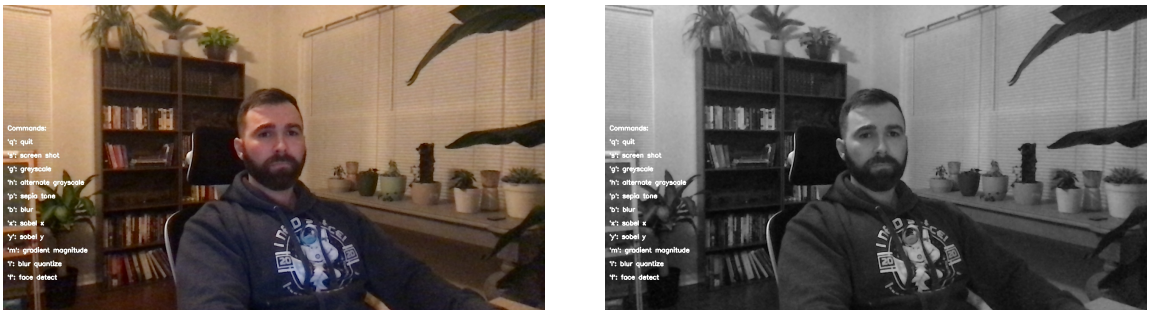
Northeastern University

January 26, 2024

Abstract

The focus of this assignment was to use OpenCV to capture video and apply various filters with an emphasis on speed and real-time responsiveness to inputs. The filters were a way to understand the various alterations that can be applied to an image by accessing individual pixels and making certain adjustments. The filters used within this assignment were greyscale, sepia tone, blurring, gradient magnitude, face detection, negative, and embossing. The project consists of an Image Display program and a Video Display program both of which make use of the various filters.

1 Greyscale live image



(a) Original image (b) Greyscale image

Figure 1: Comparison of greyscale

2 Alternate Greyscale

(a) Inverted red channel (b) Inverted green channel

Figure 2: Alternate greyscale - overwriting all channels with the value of a single inverted channel

In the first greyscale image I used openCV's *cvtColor* function to perform the conversion from the original image. With the alternate greyscale I attempted to follow the project hint and invert the red

channel. I also experimented with inverting the blue and green channels and realized that they all returned the same visuals. Since we are replacing the values for the three color channels there is no noticeable difference between using the red or green values.

The main difference between the method I used to achieve a "greyscale" conversion and openCV's method is in how the values for each pixel are calculated. According to openCV's documentation the pixel values are calculated by applying mixed ratios of the different color channels. [1] The actual formula they used is $0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$.

3 Sepia filter

Figure 3: Sepia filter - applying coefficient values to each channel

I was able to ensure the use of the original RGB values in my sepia filter by using the original pixel values in order to determine the new value after the computation of the coefficients was applied to each channel. Each color channel makes use of its original value within the calculation of the new value. This ensures that the coloring maintains similar features to the original image while applying a "vintage" feel to it.

4 5x5 Blur

The first blur filter was a naive implementation that consisted of a nested loop that would traverse the entire Gaussian kernel at each pixel. The results of the timing function for are:

- Time per image (1): 0.1512 seconds
- Total time (1): 1.5118 seconds

The second implementation of the blur filter, while not the most efficient, resulted in twice the speed. The main change was how I was accessing the values in the kernel and applying the weights to the color channels. The size of the kernel was reduced to a 1x5 instead of the previous 5x5 kernel. Even though two separate passes were made, it resulted in a much faster algorithm.

- Time per image (2): 0.0805 seconds
- Total time (2): 0.8055 seconds

Continuing to improve on the algorithm I was able to cut the time from the 2^{nd} implementation roughly in half by the 5^{th} iteration. The biggest changes were using the *.ptr* function and iterating through the channels instead of the 1x5 kernel. Each iteration calculated the total sum of weights applied to the pixel as well as its neighbors.

- Time per image (5): 0.0405 seconds
- Total time (5): 0.4050 seconds

(a) Original image

(b) Blurred image

Figure 4: Gaussian Blur - Original compared to 5^{th} iteration blur function

5 Gradient Magnitude

(a) Original image

(b) Gradient Magnitude

Figure 5: Gradient Magnitude - Sobel X and Sobel Y filters applied to image

6 Blur Quantize

(a) Original image

(b) Blur Quantize

Figure 6: Blur Quantize - Image blurred and then quantized into 10 levels

7 Face Detect

Figure 7: Face Detect

8 Image embossing

I implemented image embossing as one of the new effects by taking the product of Sobel X and Sobel Y with a direction value of 0.7071. The first attempt at this resulted in an image that was too dark to make out any distinct features. Further research [2] suggested that adding an offset to the product would return a grey-toned image that provides more detail.

(a) Embossed image with offset of 0

(b) Embossed image with offset of 128

9 Adjusting Brightness

To implement a brightness level I started with a base value of $brightness = 1.0$ applied to each pixel. When the user selects "+" or "-", the brightness moves up and down accordingly. The *adjustBrightness* function applies the value of *brightness* to every pixel in the image.

(a) Brightness adjusted to 0.2

(b) Brightness adjusted to 2.5

(c) Normal brightness

Figure 9: Brightness adjustments

10 Negative Filter

A negative filter can be implemented by subtracting the value of a color channel from 255 and thereby inverting each of the colors.

(a) Original image

(b) Negative of Original

11 Image Display Filters

I implemented a few of the filters within the original *imgDisplay.cpp* program and created a more user-friendly GUI. Users will be able to apply various filters to the image they selected without having to restart the program.

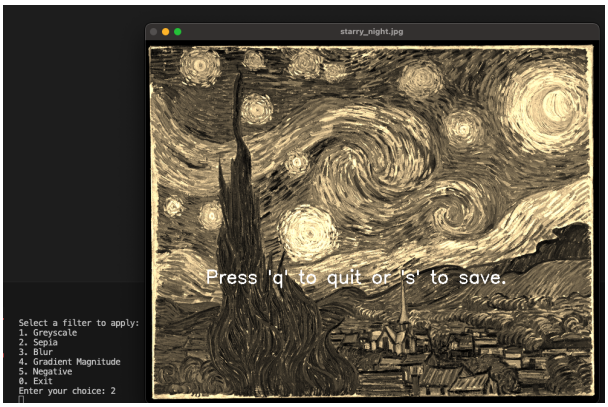


Figure 11: Image filters with CLI menu

12 Video Display Command Menu

I modified the original menu I made for the *vidDisplay.cpp* program to improve on the UI/UX for users. OpenCV does not have a lot of built in GUI functionality so I tried to find a way to at least make the menu more attractive for users. I was able to do so my creating another $cv :: Mat$ which contained the commands for all of the various filters. The menu will also highlight the currently selected command.

Figure 12: vidDisplay menu

13 Reflection

There were many challenging aspects to this project, but after some time I was able to get a deeper understanding of how to navigate through an image and make alterations. My favorite filter was the SobelX and SobelY filters which combined to create the gradient magnitude. There are many use cases for such an application from detecting abnormalities in health care, identifying pedestrians in a self-driving car, or even recognizing threats in secure environments. Overall I was able to walk away with a better understanding of what an image is composed of and the various mechanisms we can use to shape the image to a desired end state.

Link to Repo

References

[1] *Color Conversions - OpenCV 3.4 Documentation*. https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html. Accessed: January 18 2024. 2023.

[2] *Image embossing - Wikipedia*. Accessed: January 23 2024. URL: https://en.wikipedia.org/wiki/Image_embossing.