CS5330 - Project 1, Spring 2023
Arun Madhusudhanan

1. Project Description

The project utilizes the OpenCV library in C++ to perform various tasks related to image and video processing. These tasks include reading and displaying an image from a file, displaying live video from a webcam, and applying special effects such as grayscale conversion, blurring and Sobel filtering, gradient magnitude calculation, color quantization, cartoonization, pixelation, and creating multiple sketch versions of the video. Users can choose which effects to apply by using different keystrokes and also have the ability to save an image with the applied special effects from the live video. The project mainly uses custom functions for applying the special effects, with the exception of the built-in functions for generating sketch versions of the video.

2. Tasks Performed

2.1 Read an image from a file and display it.

"imgDisplay.cpp" file reads and displays an image in a window, and allows the user to quit by pressing 'q'.

2.2 Display live video

"vidDisplay.cpp" file opens a video channel, creates a window, captures and displays new frames in a loop, and quits when the user types 'q'. Tasks below are implemented on the "vidDisplay.cpp" file.

2.3 Display grayscale live video

Uses openCV cvtColor function to display the greyscale version of video if the user types ‘g’. OpencV cvtColor function uses equation 1 for conversion of RGB image to grayscale image [1].

$$\text{RGB to Gray:} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (1)$$

The formula for converting RGB to Grayscale is the same as the formula to convert RGB channel to Y in the YUV color encoding system. The higher weightage given to the green channel in the RGB to grayscale conversion formula is based on the fact that the human eye is more sensitive to variations in the green channel compared to the red and blue channels.

Image 1: Original and cvtColor version of the greyscale image

2.4 Display alternative greyscale live video

Uses a custom function to display an alternative grayscale version of video if the user types ‘h’. Following steps are used to create an alternative grayscale version.

- Create a function called "greyscale" that takes in a color video frame as a parameter.
- Using a loop, iterate through each pixel of the video frame and calculate the average of the R, G, and B color channels.
- Assign the calculated average value to the grayscale output video frame and return it to the main function.

Image 2: Customized grayscale image. The average method is straightforward but does not produce the desired results. This is because human eyes perceive RGB colors differently. The eye is most sensitive to green light, less sensitive to red light and least sensitive to blue light. As a result, the three colors should be given different weights in the conversion process.

2.5 Implement a 5x5 Gaussian filter as separable 1x5 filters.

Uses a custom function to implement a 5x5 Gaussian filter as separable 1x5 filters ([1 2 4 2 1] vertical and horizontal) if the user types 'b'. This will display a blurred version of the video. Following steps are done to implement a gaussian filter.

- Create a function called "blur5x5" that takes in a color video frame as a parameter and returns a blurred colored frame as output.
- Applied horizontal filter [1 2 4 2 1] first on the image and stored results to a temporary image. The values were normalized using a value of '10'.
- Padding by mirror method is used for the outer 2 columns. For example, while calculating the filtered value for the pixels in the first column, it is assumed that there exist 2 columns to the left which have values the same as the first and second column respectively.
- Applied vertical filter [1 2 4 2 1]' on the horizontally filtered image to obtain the output frame. The values were normalized using a value of '10'.Padding by mirror method is used for the outer 2 rows.

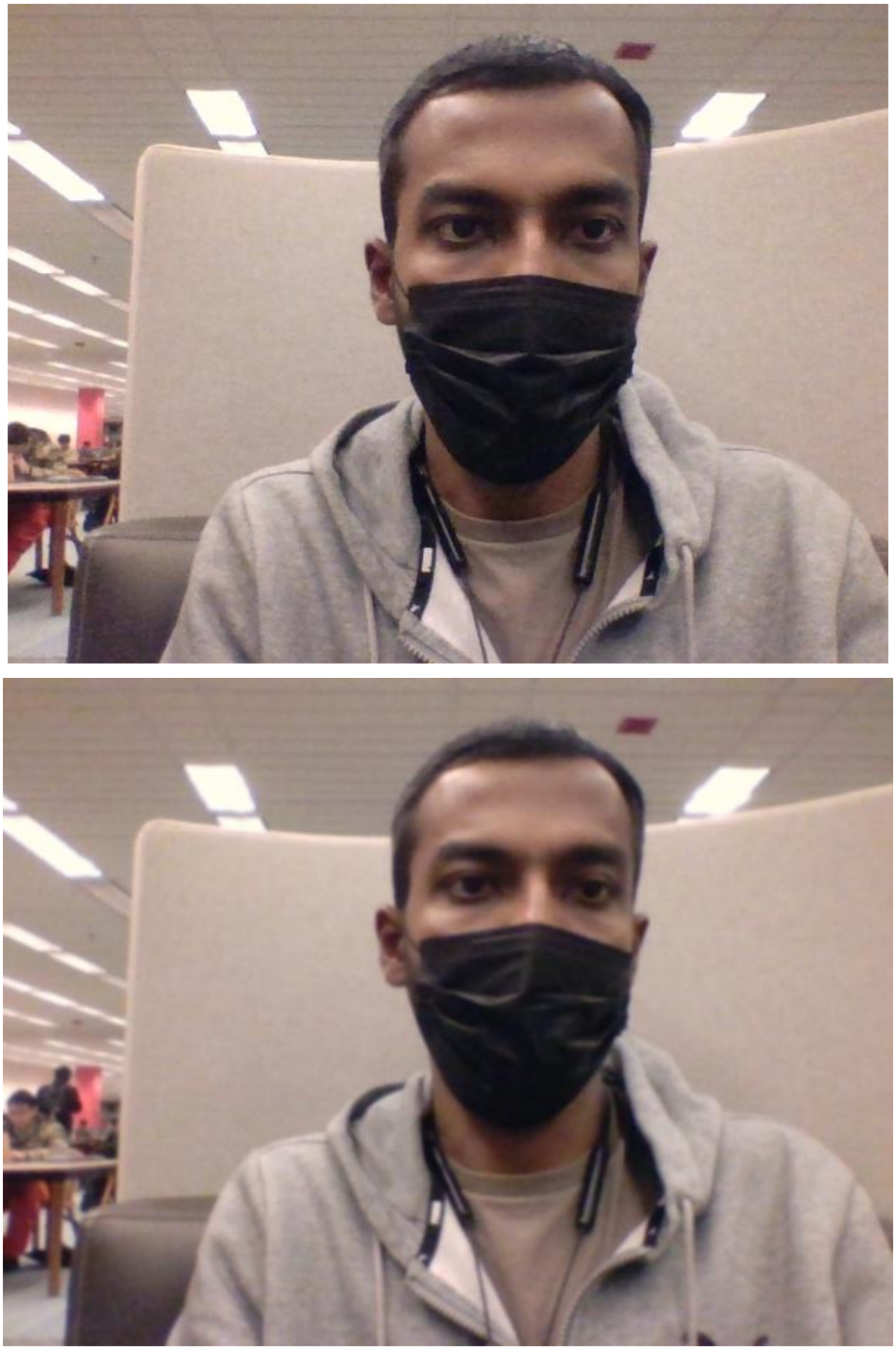

Image 3: Original and blurred image. The degree of blur on the original image is not substantial due to the small size of the gaussian filter kernel (5) in relation to the frame size.

2.6 Implement a 3x3 Sobel X and Sobel Y as separable 1x3 filters.

2.6.1 SobelX

Uses a custom function to implement a 3x3 Gaussian filter as separable 1x3 filters if the user types 'x'. This will display the video with vertical edges. Following steps are done to implement a Sobel X filter.

- Create a function called "sobelX3x3" that takes in a color video frame as a parameter and returns a frame with vertical edges as output.
- Applied horizontal filter [-1 0 1] first on the image and stored results to a temporary image.
- Padding by mirror method is used for the outer columns. For example, while calculating the filtered value for the pixels in the first column, it is assumed that there exists 1 column to the left which has values the same as the first column .
- Applied weightage values of [1 2 1]' on the horizontally filtered image to obtain the output frame. The values were normalized using a value of '4'.Padding by mirror method is used for the outer rows.

2.6.2 SobelY

Uses a custom function to implement a 3x3 Gaussian filter as separable 1x3 filters if the user types 'y'. This will display the video with horizontal edges. Following steps are done to implement a Sobel Y filter.

- Create a function called "sobelY3x3" that takes in a color video frame as a parameter and returns a frame with vertical edges as output.
- Applied vertical filter [1 0 -1]' first on the image and stored results to a temporary image.
- Padding by mirror method is used for the outer columns. For example, while calculating the filtered value for the pixels in the first row, it is assumed that there exists 1 row above which has values the same as the first row .
- Applied weightage values of [1 2 1] on the vertically filtered image to obtain the output frame. The values were normalized using a value of '4'.Padding by mirror method is used for the outer columns.

2.7 Implement a function that generates a gradient magnitude image from the X and Y Sobel images.

Uses a custom function "magnitude" that generates a gradient magnitude image using Euclidean distance for magnitude: I = sqrt(sx*sx + sy*sy), if the user types 'm'. This will display the video with edges detected.

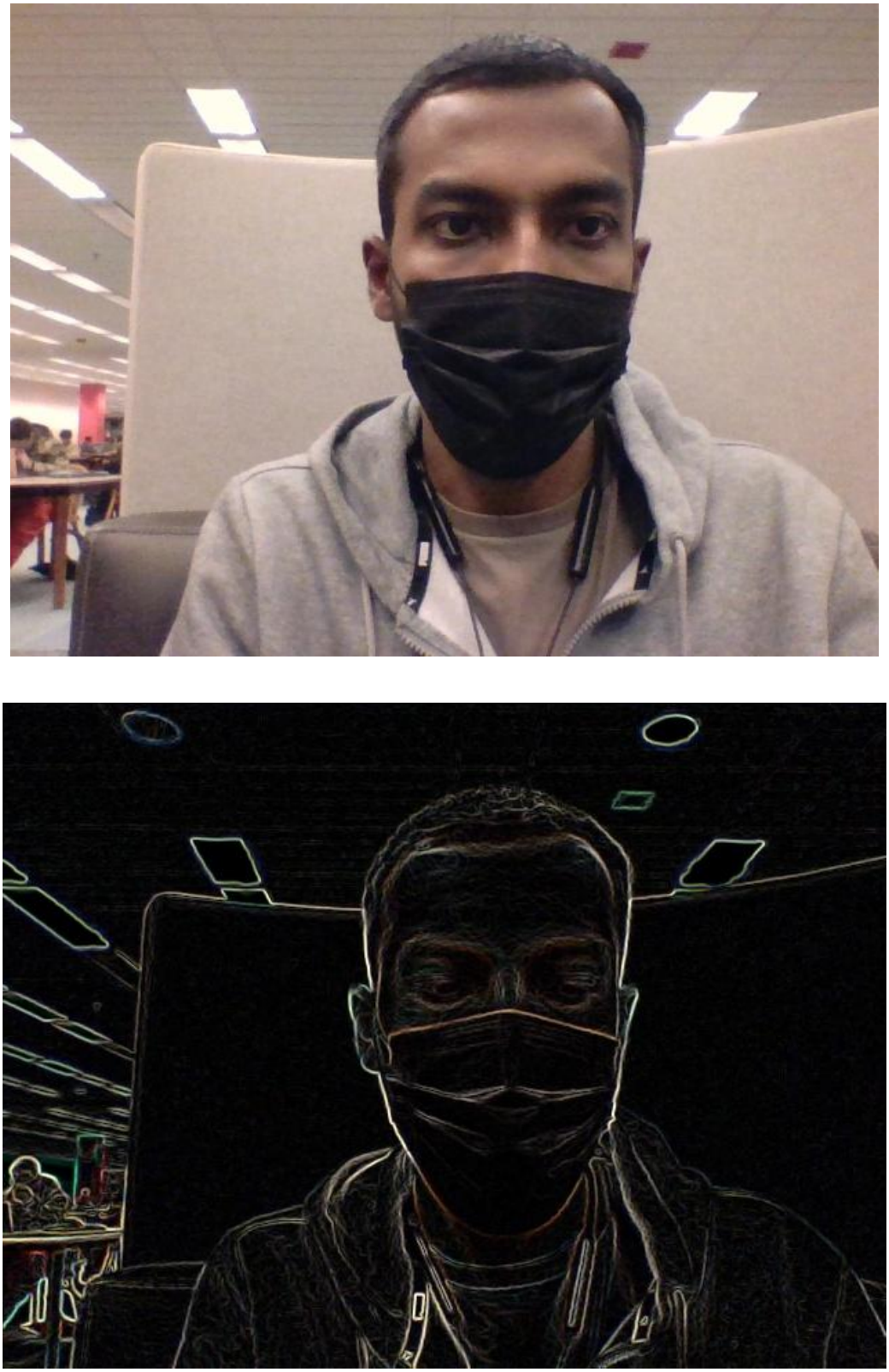

Image 4: Original and gradient magnitude image. The edges and features are detected in the gradient magnitude image

2.8 Implement a function that blurs and quantizes a color image

Uses a custom function that generates a blur and quantized image if the user types ‘l’. Following steps are done.

- Create a function called "blurQuantize" that takes in a color video frame and level value as a parameter and returns a blur and quantized frame as output.
- Color video frame was first blurred using "blur5x5" with two separable 1*5 filters per task 2.5.
- Each color channel of the blurred frame was quantized into 15 values i.e a level parameter of 15 is used.

Image 5: Original and blurred/quantized image. When the image is blurred, the small details and edges are smoothed out, making the image less sharp. When the image is then quantized, the number of levels of intensity is decreased, which leads to a loss of detail and an increase in visible artifacts. This resulted in an image that appears less realistic and less true to the original.

2.9 Implement a live video cartoonization function using the gradient magnitude and blur/quantize filters.

Uses a custom function that cartoonizes a color image if the user types ‘c’. Following steps are done.

- Create a function called "cartoon" that takes in a color video frame , level value and threshold value as a parameter and returns a cartoonized frame as output.
- Calculate the gradient magnitude of the color video frame first using the “magnitude” function per task 2.7.
- Color video frame was blurred and quantized using the "blurQuantize" function. Each color channel of the blurred frame was quantized into 15 values i.e a level parameter of 15 is used.
- Pixel values with a magnitude threshold greater than a given threshold value is set to black. A threshold value of 18 is used.

Image 6: Original and cartoonized image. The image appears to be more stylized and simplified, with less detail and a more cartoon-like appearance.

2.10 Adjust the brightness and contrast of video

Point process multiplication and addition with a constant is used for changing the brightness and contrast.

$$g(x) = af(x) + b, \quad (2)$$

f(x) is the source image pixels and g(x) is the output image pixels. The parameters a > 0 and b are often called the gain and bias parameters;these parameters are said to control contrast and brightness[2]. The change in gain changes the difference between pixel values, causing the contrast to change. The change in bias changes the pixel values by an offset, causing the brightness to change.

Created a custom function “brightness_contrast” to change the brightness and contrast based on the user input as shown below.

- The value of a is incremented by 0.1 when the user types ‘r’ which will increase the contrast.
- The value of a is decremented by 0.1 when the user types ‘i’ which will decrease the contrast.
- The value of b is incremented by 5 when the user types ‘u’ which will increase the brightness.
- The value of b is decremented by 5 when the user types ‘d’ which will decrease the brightness.

- Any values going above 255 is set us white and values going below 0 is set us black.

Image 7: a) Actual image. b) Image with increased brightness. c)Image with decreased brightness. d) Image with increased contrast. e) Image with decreased contrast.

3 Extensions

3.1 Pixelation

Uses a custom function “pixelate” that takes in an image and applies a pixelation effect to it, if the user types ‘p’. The pixelation effect is achieved by dividing the image into square regions of a given size (specified by the "pixel_size" parameter, a value of 25 is used) and then averaging the color values of all pixels within each region. The resulting average color value is then applied to all pixels within the region.

The function starts by initializing the "dst" cv::Mat object as an array of zeros with the same dimensions as "src". It then loops through the rows and columns of the image. Within each iteration of the outer loop, it takes the average of the color values of all pixels within a region defined by the pixel size and the current loop variables, and then applies that average color value to all pixels within that region in the "dst" image.

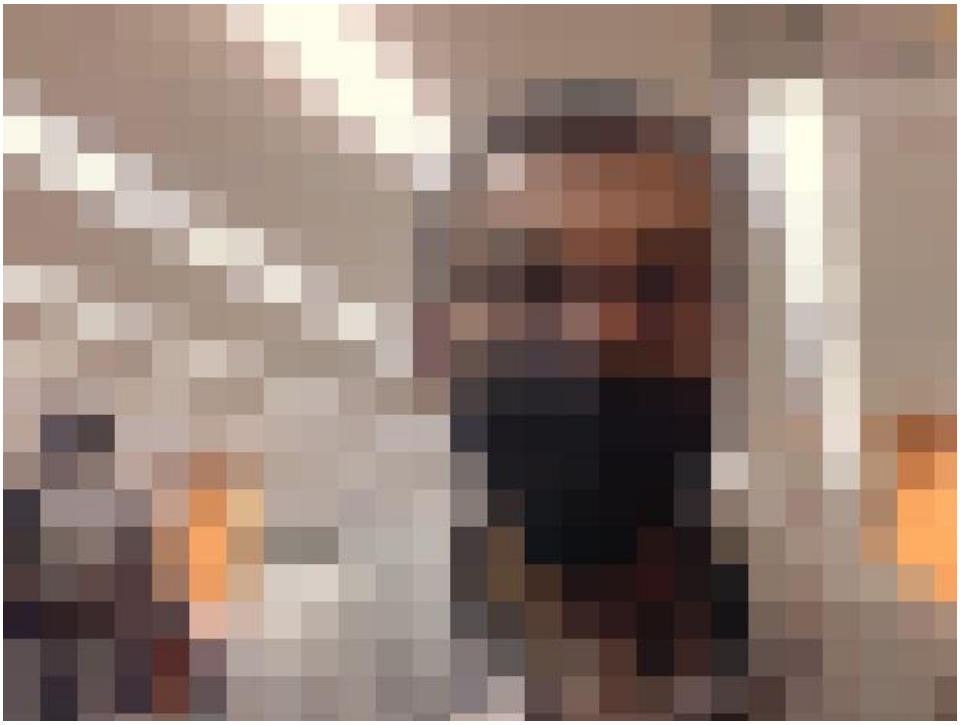

Image 8: Original and Pixelated image

3.2 Additional effects[3]

3.2.1 Edge preserving filter

Uses an OpenCV function cv::edgePreservingFilter(src,dst,flags) to apply an edge-preserving smoothing filter to an image, if user types ‘1’. The function smoothes an image by reducing noise and slight variations in pixel intensity, while preserving the edges of the image. This can be useful for removing noise from an image without losing important details and edges.It takes in three arguments:

1. src: the source image that the filter is applied to
2. dst: the destination image where the filtered result is stored
3. flags: the type of filter that is used. 1 stands for RECURS_FILTER and 2 stands for NORMCONV_FILTER. Using the RECURS_FILTER option is about 3.5x faster than NORMCONV_FILTER, hence flag = 1 is used.

Image 9: Original and Edge preserving filtered image. The final result is a smoothed image where noise is decreased while the edges remain clearly visible.

3.2.2 Detail enhancement filter

Uses an OpenCV function cv::detailEnhance(src,dst) to enhance the fine details in an image if user types ’2’. It takes in two arguments:

1. src: the source image that the filter is applied to
2. dst: the destination image where the filtered result is stored The function enhances the details in an image by amplifying the high-frequency components of the image, which are responsible for the fine details and textures. The function can be used to make an image look sharper and more detailed.

Image 10: Original and Detail enhanced image. The output image is much sharper than the original image.

3.2.3 Pencil sketch

Uses an OpenCV function cv::pencilSketch(Mat src, Mat dst_gray, Mat dst_color, float sigma_s, float sigma_r, float shade_factor) function that applies a pencil sketch effect to an image. Function returns a grayscale pencil sketch if user types ‘3’ and returns a colored pencil sketch if user types ‘4’.

- It takes three Mat arguments, src, dst_gray and dst_color, representing the source image, the destination grayscale image and the destination color image respectively.
- The function also takes three optional parameters, sigma_s, sigma_r and shade_factor which are used to control the level of the sketch effect. sigma_s controls the spatial range of the effect, sigma_r controls the range of color variation and shade_factor controls the level of shading.
- The function applies a pencil sketch effect to the src image and saves the result in both dst_gray and dst_color Mat objects.

Image 11: a) Original sketch. b) gray pencil colored sketch c) color pencil colored sketch. The results aren’t great as expected though.

3.2.4 Stylization(watercolor sketch)

Uses an OpenCV function cv:: stylization(Mat src, Mat dst, float sigma_s, float sigma_r) that produces an output that looks like the image was painted using watercolor, if the user types ‘5’.

- The function takes in two main arguments: the source image (src), and the destination image (dst) where the stylized output will be stored.
- The function also takes in two optional parameters: sigma_s and sigma_r. sigma_s controls the spatial extent of the smoothing effect and sigma_r controls the range of colors that will be affected.

Image 12: Original and stylized sketch. The output image looks like that the image was painted using watercolor

3.3 Save image with special effects

If the user types ‘s’, an image with current special effects will be saved to the current working directory by the name “save.jpg”**.**

4. Reflection of lessons learnt

The exercises provided an in-depth understanding of different fundamental filters that are commonly used in image processing. By constructing these functions from scratch, it was possible to gain a deeper understanding of how these filters are implemented, the data types of pixel values that are stored, as well as the methods for extracting these values (such as the difference between using “ptr” and the "at" method). Furthermore, the process of breaking down filters into separate components in order to increase computational speed was also explored. Additionally, the extension tasks familiarized me with various built-in functions of OpenCV and its wide range of capabilities for processing images and videos.

5. References

[1] Color conversions.
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray

[2] Szeliski, R. (2010). Computer vision: Algorithms and applications. Springer.
http://szeliski.org/Book/download/1TN5RZRLUADA/Szeliski_CVAABook_2ndEd.pdf

[3]Mallick, S. (n.d.). Non-Photorealistic Rendering using OpenCV and Python.
https://learnopencv.com/non-photorealistic-rendering-using-opencv-python-c/