Leonardo DaGraca
CS5330
Project 1 Report

Project Description

This project was an introduction to both C++ and the OpenCV library. It required getting familiarized with both frameworks and then implementing several tasks that were related to class lectures and readings. More specifically, the goal of this project was to first understand how to display images and videos using OpenCV. The next step was figuring out how to access pixels from a Mat object and then applying certain changes to these pixels that resulted in a filter. Several filter functions were needed to be implemented to change an original image to a desired filtered image. The program designed for this project allows for special effects on a live video.

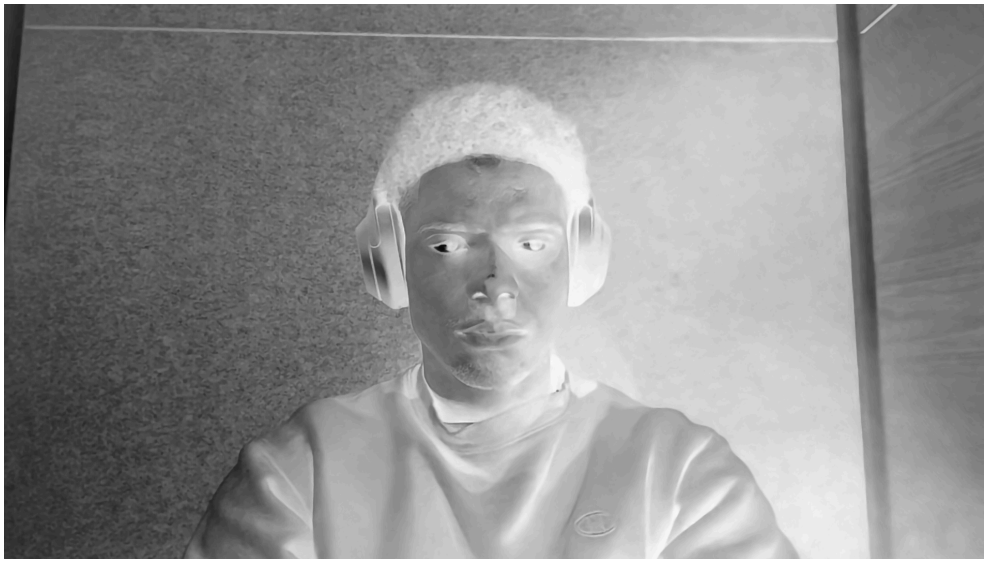
Required Images

Image 1:

Original

Greyscale

Image 2:



Alternative greyscale

This greyscale's implementation differs from OpenCV's greyscale as the blue color channel was subtracted and that change was added to each color channel. Here the intensity is based on the inverse of the blue channel. You can also notice that lumination has increased significantly with this change and turns the darker colors white and the whiter colors black.

Image 3:

For my sepia tone filter, I accessed the pixels using a nested loop and two uchar pointers, one for the src image and the other for the dst image. The row loop(i) was where I declared my pointers for each row in the image. For my column loop, I accessed each color channel by multiplying the current j column pixel I was at and multiplied it by 3. Due to the channels being accessed in a BGR order. The blue channel was set to the current row[current col * 3] and was of type float (float blue = srcptr[3 * j]). If I were to access pixel (0, 2), my BGR values would access index cols 6, 7, and 8 correctly. The green and red channels were set up similarly but were incremented by 1 and 2 respectively to access their channels correctly. I then set up 3 variables to store the values of the new color channel calculations. These calculations used the original color channel values that I set to be accessed at each respective pixel. I then clamped the new values after their calculations and stored them accordingly in the dst image. The full code block can be seen below:

```cpp
for(int i = 0; i < src.rows; i++){
  uchar* srcPtr = src.ptr<uchar>(i); //stores a ptr to each row in memory
  uchar* dstPtr = dst.ptr<uchar>(i);

  for(int j = 0; j < src.cols; j++){

    //access each color channel
    float blue = srcPtr[3 * j]; //mult by 3 at each col to get exact index for colo
    float green = srcPtr[3 * j + 1];//move by 1 to access green channel
    float red = srcPtr[3 * j + 2]; //move by 2 to access red channel

    //apply sepia coeffecient change to get new color values
    float newRed = (0.393 * red) + (0.769 * green) + (0.189 * blue);
    float newGreen = (0.349 * red) + (0.686 * green) + (0.168 * blue);
    float newBlue = (0.272 * red) + (0.534 * green) + (0.131 * blue);

    //keep the values of the pixels within the range of 0 - 255 using the saturate_
    newRed = saturate_cast<uchar>(newRed);
    newGreen = saturate_cast<uchar>(newGreen);
    newBlue = saturate_cast<uchar>(newBlue);

    //apply sepia color change to the respective color channel in dst image
    dstPtr[3 * j] = newBlue;
    dstPtr[3 * j + 1] = newGreen;
    dstPtr[3 * j + 2] = newRed;
  }
```

Image 4:

Original

Blur

Timing Information:

```
(base) leodagraca@MacBook-Pro-3 build % ./timeBlur cathedral.jpeg
Time per image (1): 0.0604 seconds
Time per image (2): 0.0277 seconds
Terminating
```

Here we can see that the first image using the blur_1 function took longer to compute than the blur_2 function. The second blur function was implemented with the use of pointers instead of the at method. It also used separable 1x5 filters to make the changes to the destination image.

Image 5:

Original

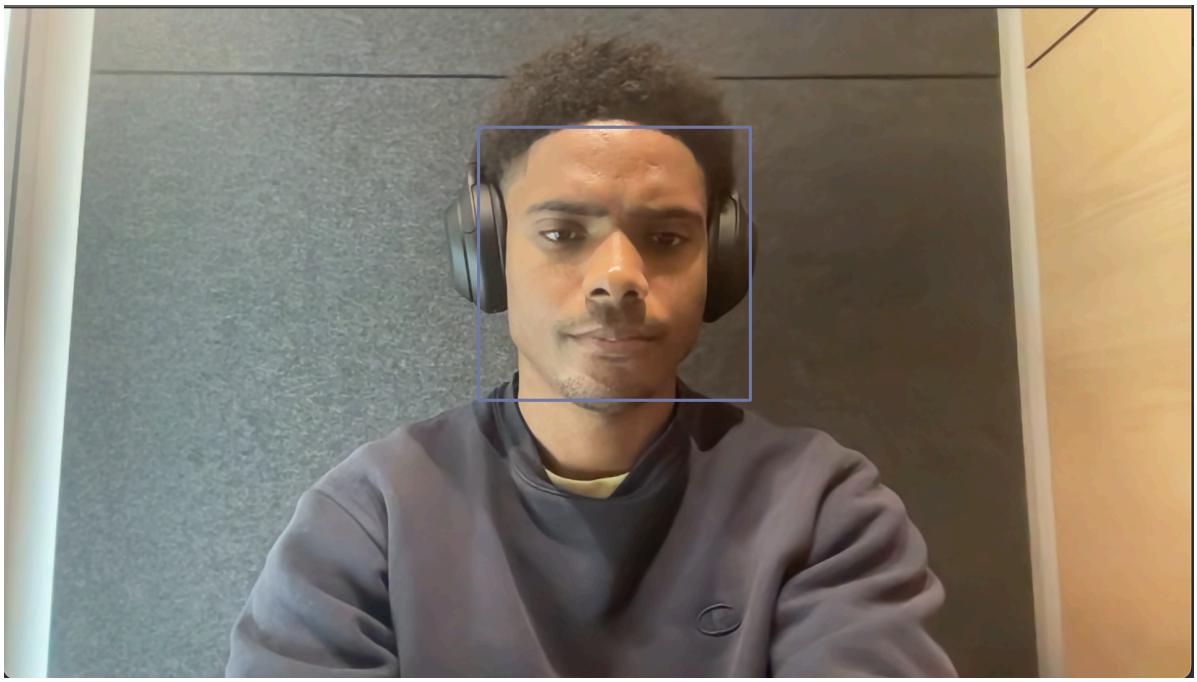
Gradient Magnitude

Image 6:

Original

blurred/quantized

Image 7:



Face detected

Image 8 (Allow user to adjust brightness or contrast):

Original

Contrast adjusted with alpha of 1.5 and brightness adjusted with beta of 50

Image 9 (Make an embossing effect):

Original

Embossing effect

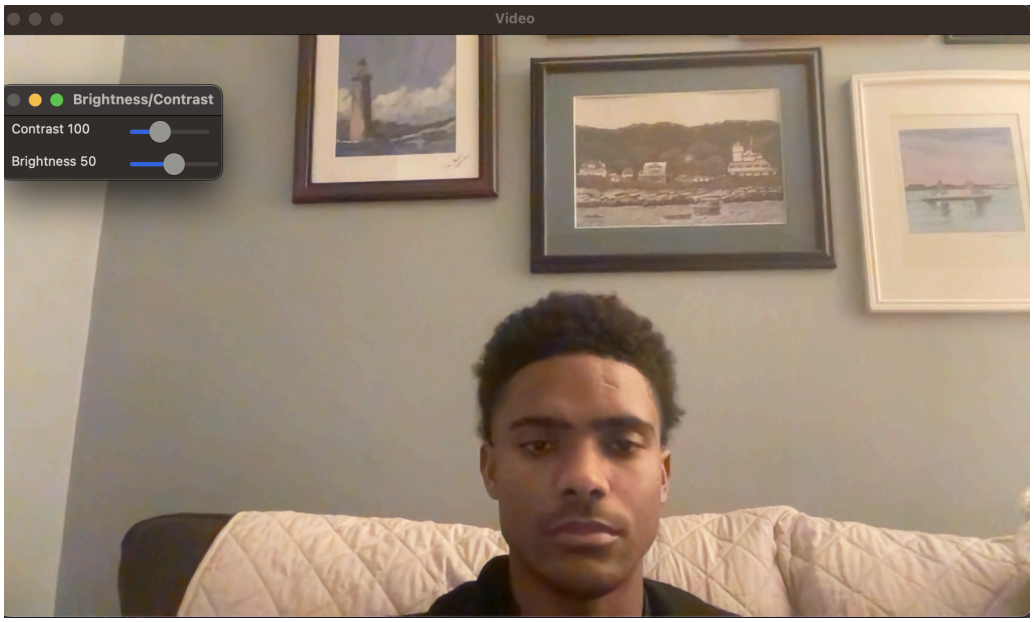
Image 10 (Blur the outside of found faces):

Original

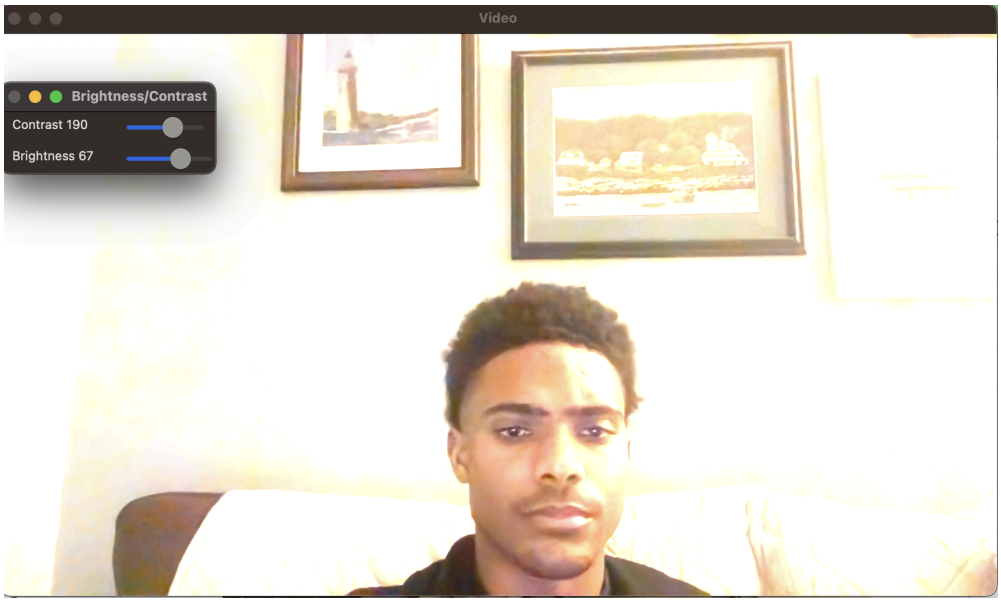
Output image with blur outside of a found face

Extensions

For an extension, I added a trackbar that allows the user to adjust both the brightness and contrast of the video. Instead of the user interacting with the command line to adjust the ranges they can use the much more responsive trackbar. The trackbar clamps the ranges accurately and allows for on-the-spot adjustment.



Original



Filter applied after the sliders were altered by the user

Reflection

I would say this project took a lot of patience from me but in a good way. It was my first time using these tools which required extensive research. While learning these tools, it was also important to understand the theoretical concepts behind each filter effect. With that in mind, I initially focused on ways to access the pixels and their respective color channels. I learned that the type the pointer is set to plays a crucial role in how pixels are accessed and manipulated. From there I looked at each function and looked for what needed to be accessed and how exactly it should be manipulated in order to get the right effect on the output image. Some functions were easier to implement than others but in the end I learned a great deal from this project.

Acknowledgments

https://docs.opencv.org/4.x/d3/dc1/tutorial_basic_linear_transform.html

https://docs.opencv.org/4.x/da/d6a/tutorial_trackbar.html

Professor Maxwell's tutorials