Zhimin Liang

Spring 2024

CS5330 Project report

1. A short description of the overall project in your own words. (200 words or less)

This project spans from basic image and video display functionalities to advanced image processing techniques using OpenCV.

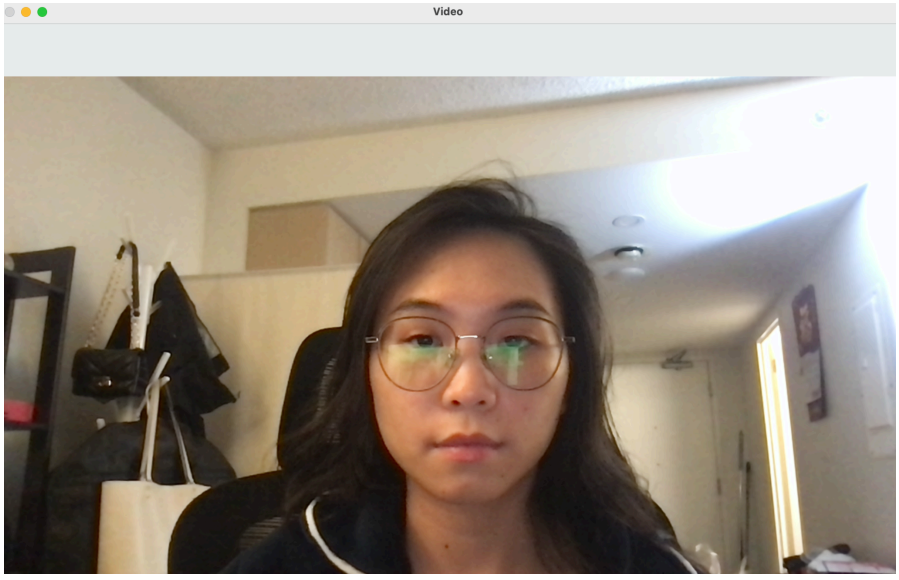
It starts with fundamental tasks like reading and displaying images, then evolves into captivating features such as live video capture. Then, the project takes on an artistic twist. It starts with greyscale live video displays and gets creative by implementing customized greyscale transformations. The journey continues with vintage sepia tone filters that give images that classic, aged-camera aesthetic. Also, the project dive into edge detection with Sobel filters, face detection for a touch of human interaction, and even add personalized effects like blurring outside detected faces.

This project is more than just coding; it's an interactive guide to understanding computer vision concepts and creating a real-time video application.

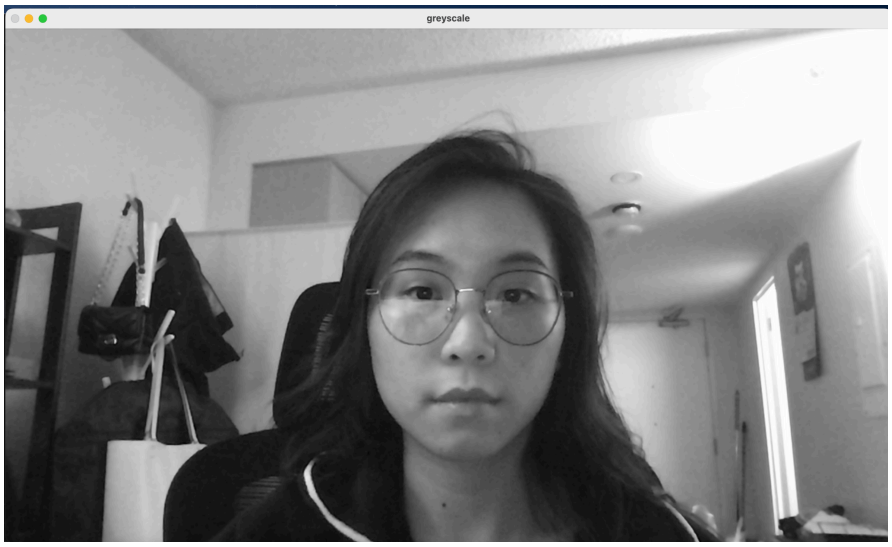
2. Any required images or text along with a short description of the meaning of each image or diagram.

Task3: Display greyscale live video

Required Image 1: *show the original and cvtColor version of the greyscale image in your report.*



The above image is the original video image



The above image is the greyscale image , which use OpenCV's built-in cvtColor function to convert the input color image (frame) to a greyscale image.The specified conversion code cv::COLOR_BGR2GRAY indicates that the input image is in BGR color format, and the function should convert it to greyscale.

Task4: Display alternative greyscale live video

Required image 2: *show your customized grayscale image in your report, explain how you decided to generate your greyscale version, and highlight the differences with the default grayscale image.*

In the custom greyscale transformation, I choose a distinctive approach by first inverting the colors of the input image. Specifically, I subtracted each color channel value from 255. This inversion process is a creative choice, introducing a unique visual effect to the greyscale conversion.

The main difference is in the inversion step. While the default cvtColor method calculates the luminance based on standard color channel weights, the custom method introduces an inverted color scheme before converting to greyscale.

Task 5:Implement a Sepia tone filter

Required image 3: *show your sepia tone filter in your report and explain how you ensured using the original RGB values in the computation.*

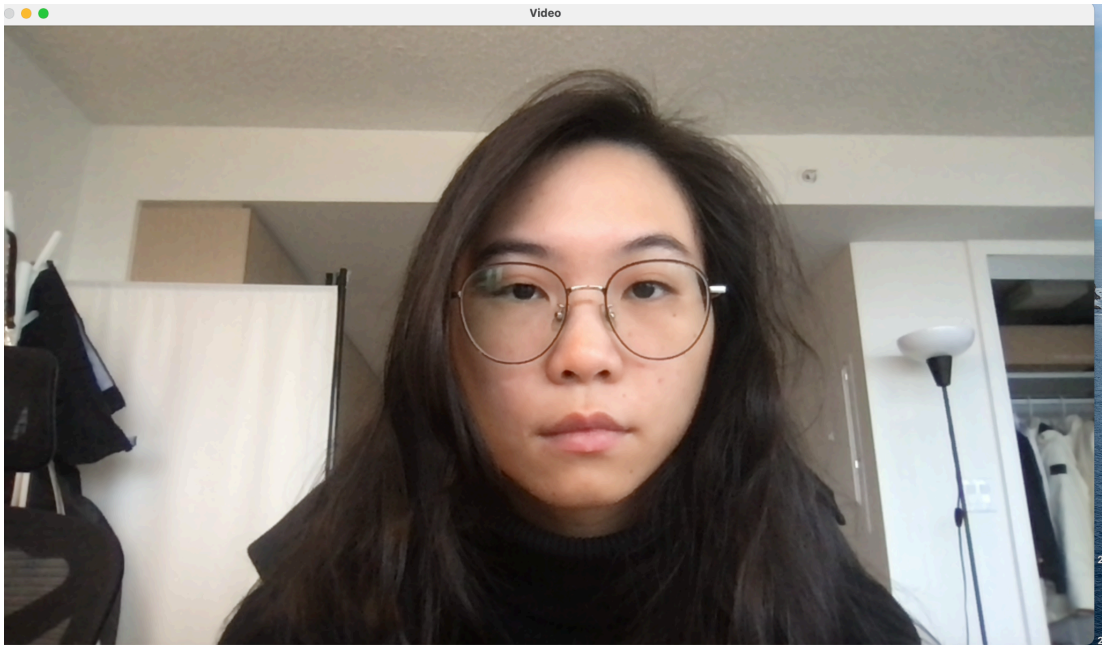
The sepia matrix defines the coefficients used to calculate the new values for each color channel in the sepia filter. The matrix multiplication is performed on the original RGB values of each pixel.

After computing the new color values, the function ensures that they fall within the valid range [0, 255] by using std::min and std::max.

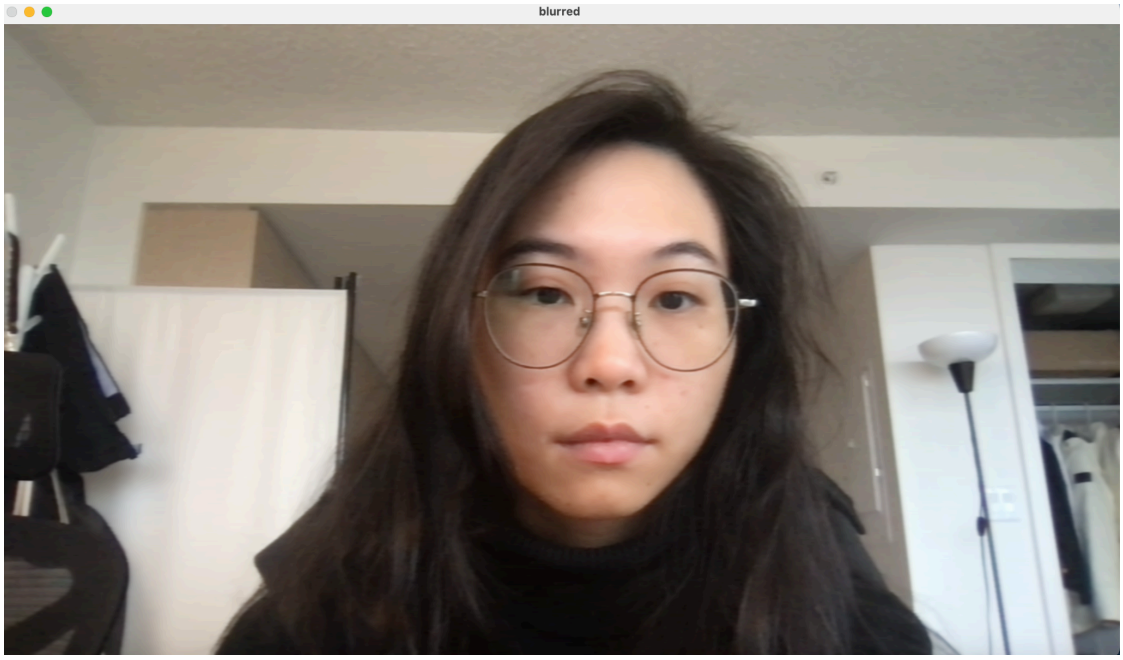
Finally, the new color values are assigned to the corresponding pixel in the destination image. This ensures that the sepia tone filter is applied using the original RGB values, maintaining the integrity of the input image during the transformation.

Task 6: Implement a 5x5 blur filter

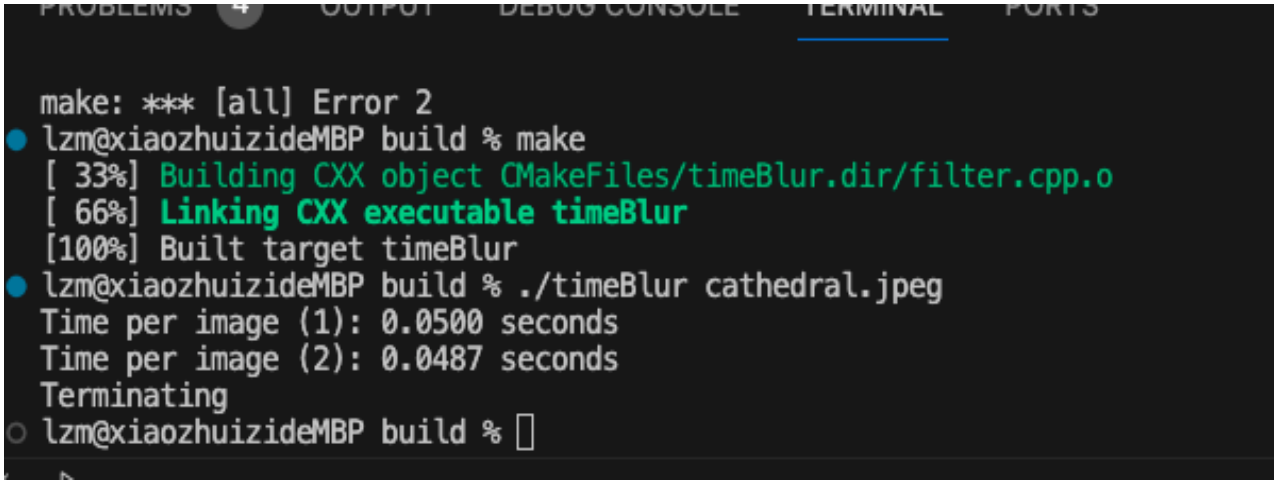
Required Image 4: *show the original and the blurred image in your report along with the timing information.*



The above image is the original video image.



This is the blur image using the second function blur5x5_2.

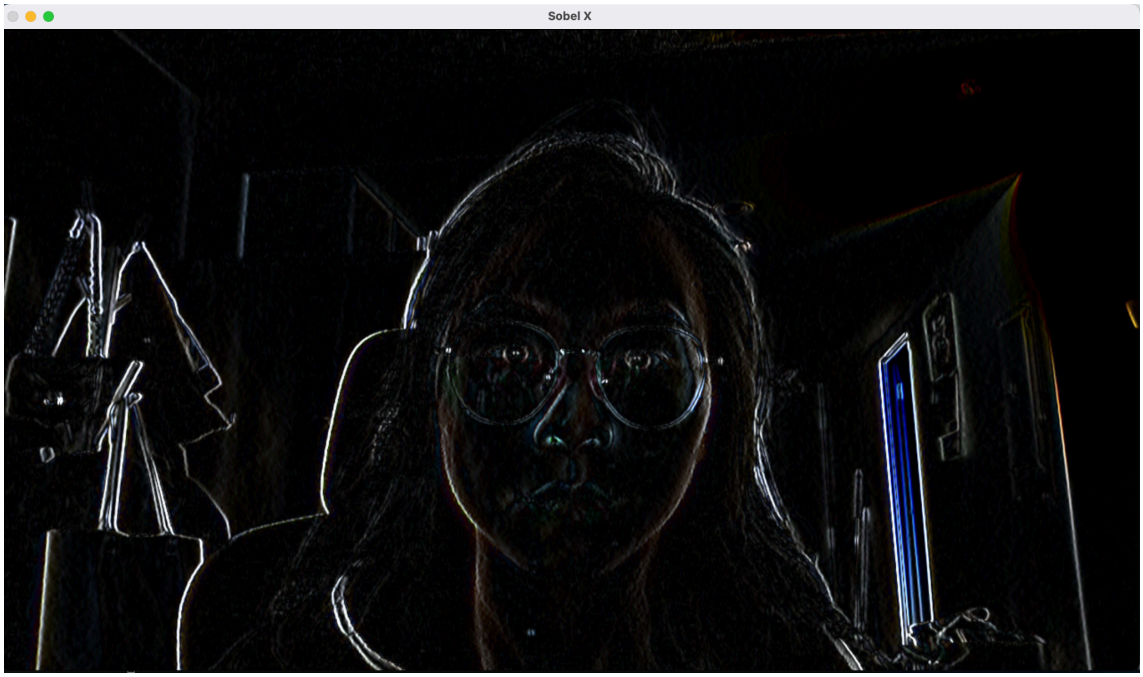


From the time information, the first function uses 0.5000s, and the second function uses 0.0487s, which is faster than the first function.

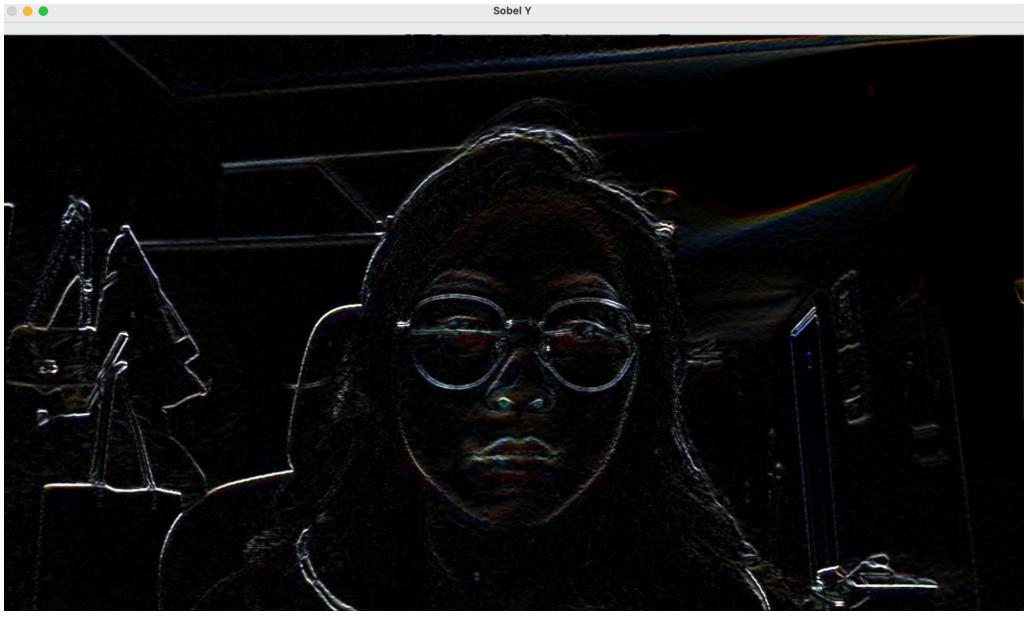
Because the second function avoids using the at<> method, which involves additional bounds checking. Instead, it uses pointers to directly access pixel values, which can be more efficient.

And the second function uses separable 1x5 filters both horizontally and vertically. Separable filters require fewer computations compared to a full 5x5 filter, making the algorithm more efficient.

Task 7: Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters



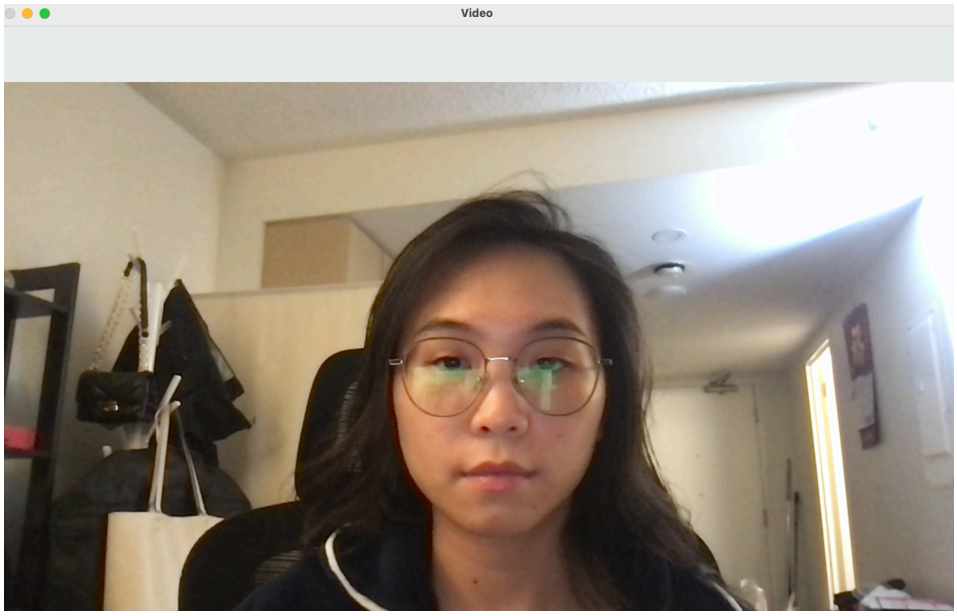
For sobelX3x3' function, the algorithm calculates the gradient along the X-axis for each color channel, resulting in a new image that highlights horizontal edges.



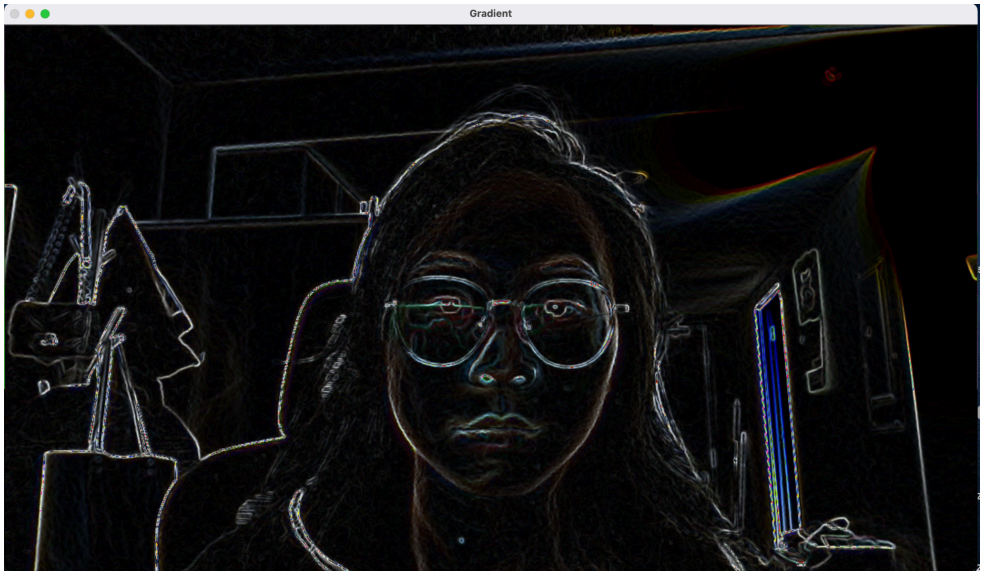
For sobelY3x3' function, the algorithm focuses on calculating the gradient along the Y-axis for each color channel, emphasizing vertical edges in the image.

Task 8:Implement a function that generates a gradient magnitude image from the X and Y Sobel images.

Required Image 5: *show the original and the gradient magnitude image in your report.*



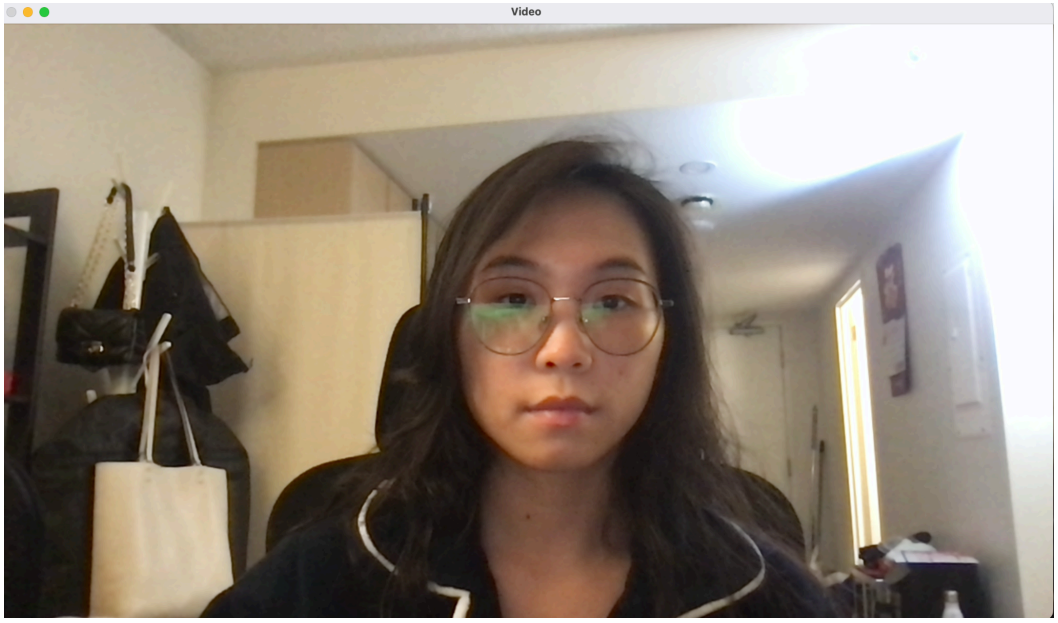
The above is the original image from video capture.



The 'magnitude' function calculates the gradient magnitude of an input image by combining the results of Sobel X and Sobel Y filters. The resulting image ('dst') highlights the intensity of edges in both horizontal and vertical directions. This operation is crucial for edge detection and can be used to emphasize prominent features in the image.

Task 9: Implement a function that blurs and quantizes a color image.

Required Image 6: *show the original and the blurred/quantized image in your report.*

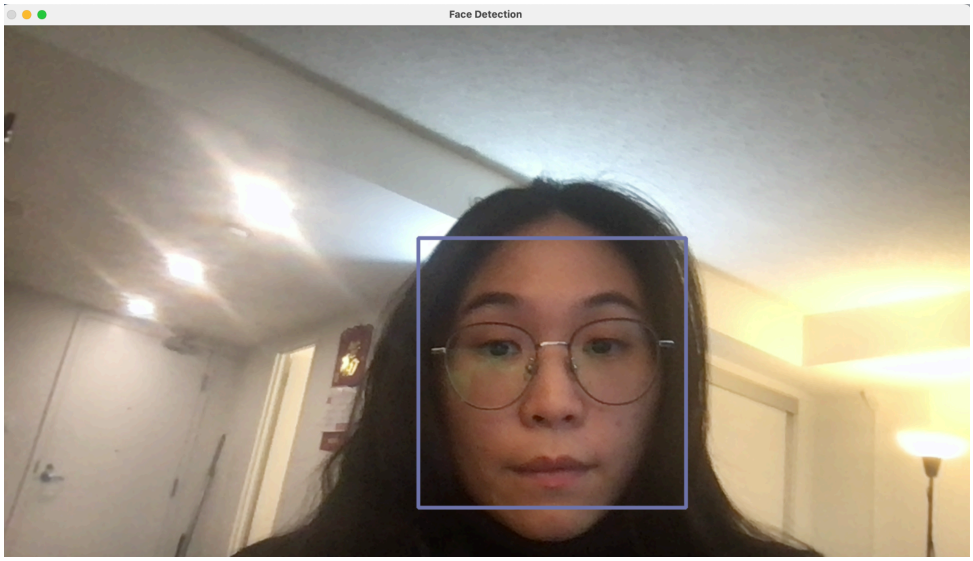


The above is the original image from video capture.

The 'blurQuantize' function performs color quantization on an input image, reducing the number of distinct color levels to enhance visual simplicity or reduce data size. It divides the color range into buckets and assigns each pixel to the nearest bucket, resulting in a quantized image ('dst'). This operation is commonly used for image compression or stylized visual effects.

Task 10: Detect faces in an image.

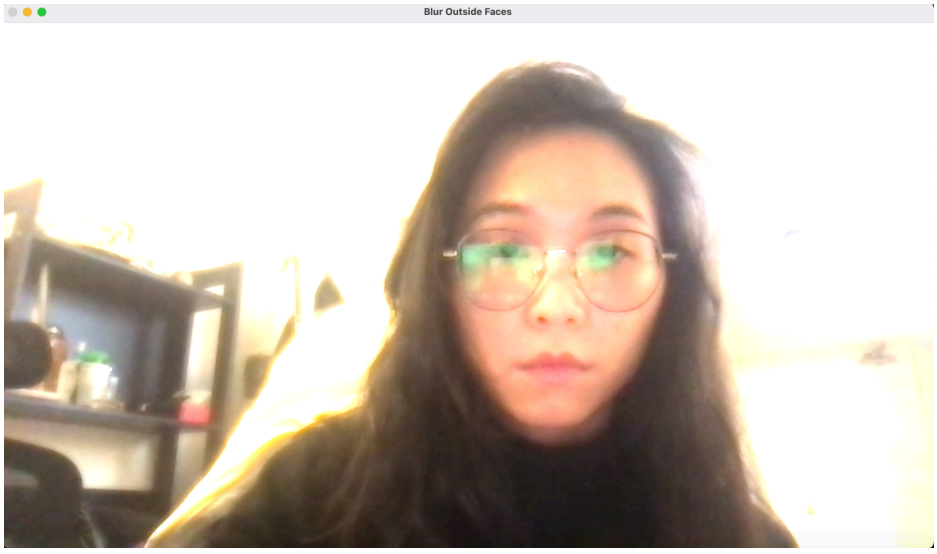
Required Image 6: *show a face being detected in your video stream.*



The above image show that the face has been detected using the provided file faceDetect.cpp

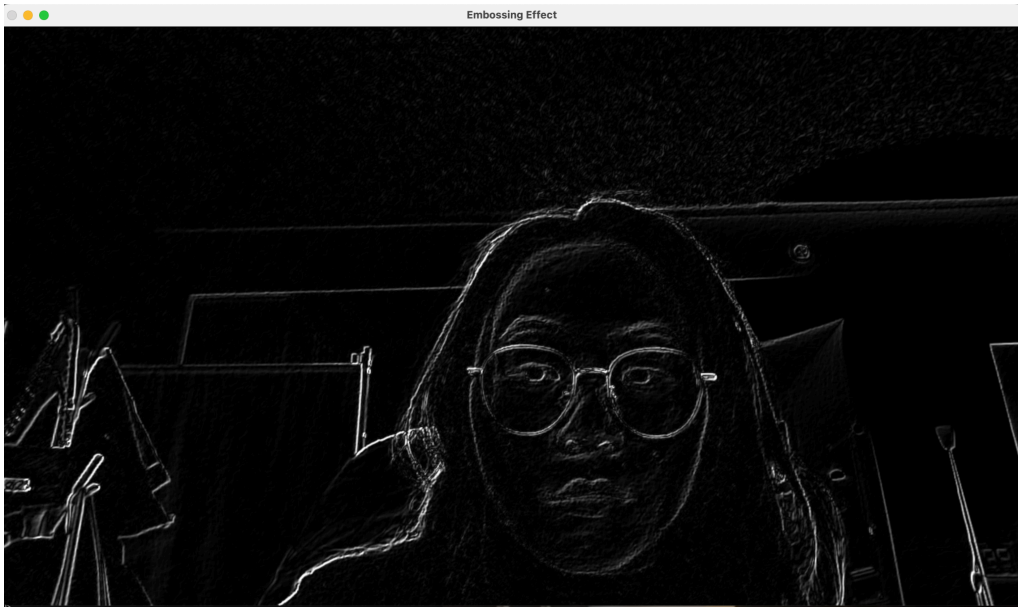
Task 11: Implement three more effects on your video

The above is the original image for effect 1 and 2.



Effect 1: Blur the image outside of found faces.

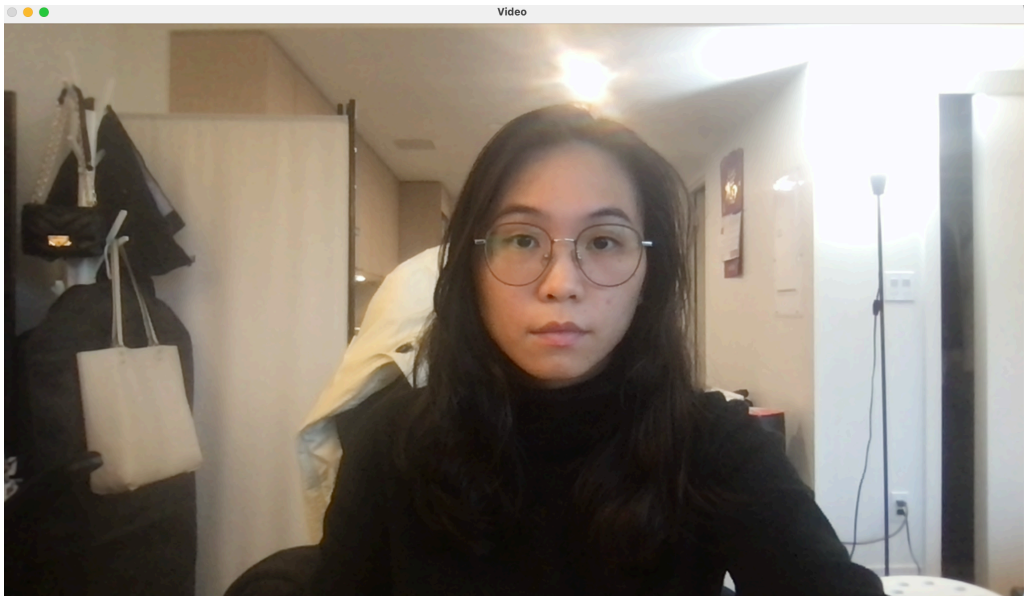
This function essentially blurs the background of the input image outside the detected face regions, creating an effect where faces remain sharp while the background is blurred. Adjustments to the kernel size in the blur operation can be made based on the desired level of blurring.

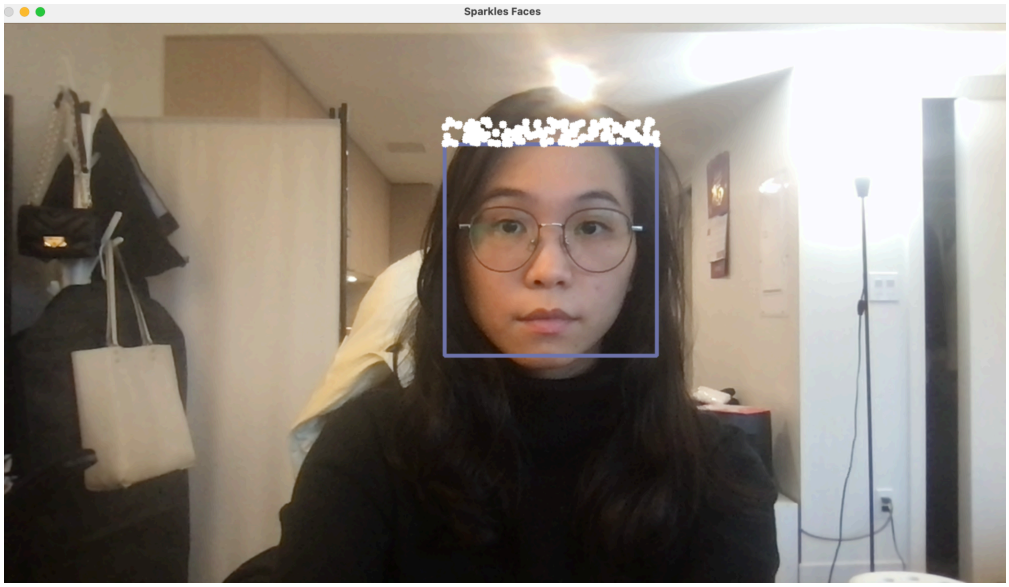


Effect 2: Make an embossing effect

The function applies an embossing effect to the input image by combining the Sobel gradients in both directions. This effect enhances the edges and creates a 3D-like appearance in the image, simulating the effect of embossing on a surface.

Effect 3: Put sparkles in a halo above the face.



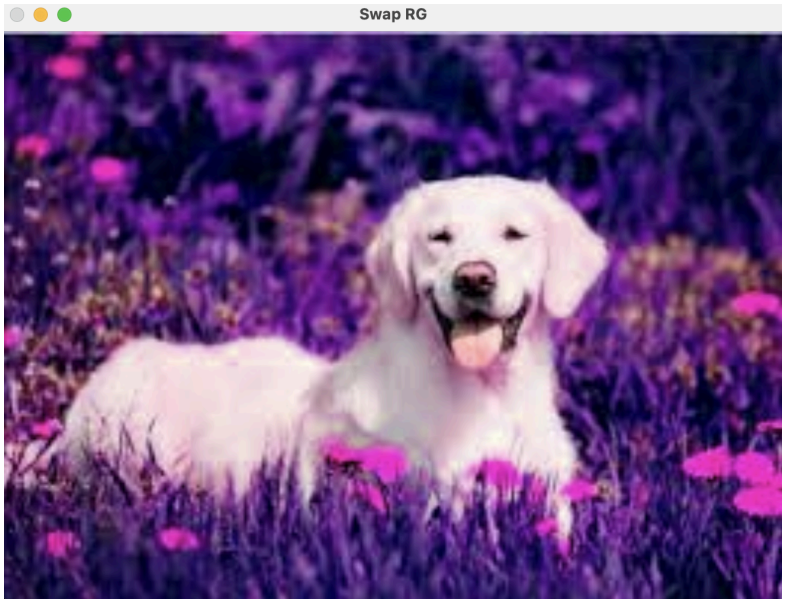


This effect adds sparkles above the detected faces in the input frame. For each detected face, a halo region is created above it, and sparkles of varying positions are generated within this halo. The sparkles are represented as filled white circles, providing a whimsical and decorative touch to the facial regions in the frame.

3.A description of and example images for any extensions.

Extension 1: modify the image (swap the red and green channels)

This is the original image I display.



The extention uses nested loops to iterate through each pixel of the image.For each pixel, it swaps the values of the red (index 0) and green (index 1) channels using a temporary variable.This operation effectively modifies the image by exchanging the red and green color components for each pixel.

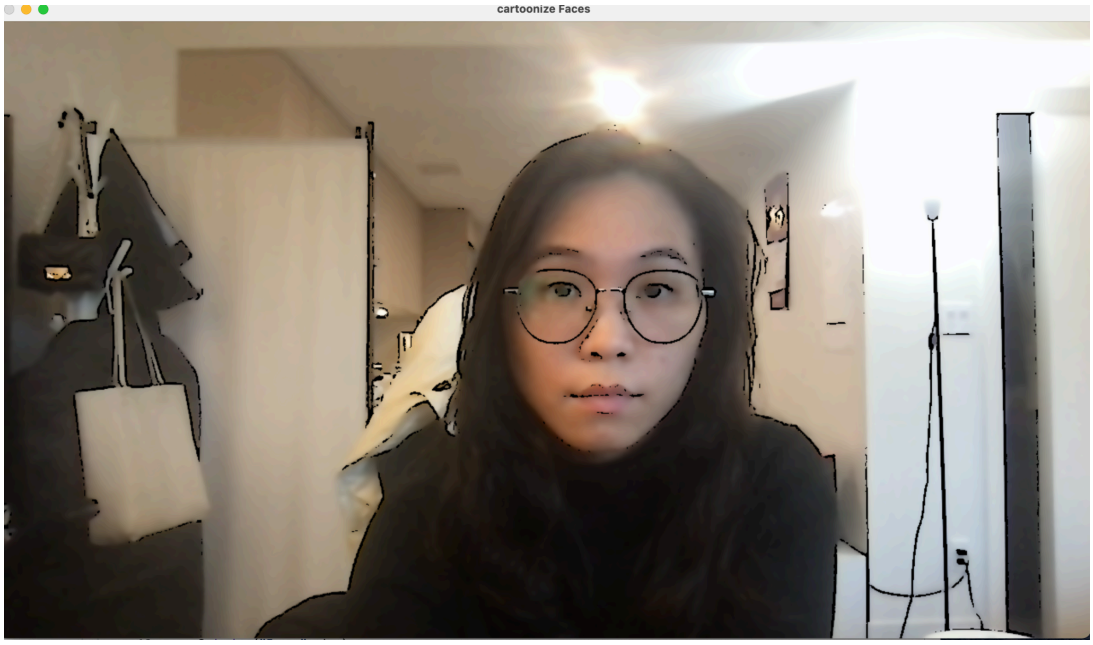
Extension 2: Additional Vignetting Effect to Sepia tone filter

The above image is use Sepia tone filter without Vignetting

An extension to task 5, is to add a vignetting effect to the Sepia tone filter. Vignetting makes the image darker towards the edges. The intensity of vignetting can be controlled, using a vignetting formula to calculate the vignette intensity based on the pixel's distance from the center of the image.

Extension 3: Cartoonization of the video stream

The above image is the original video image



This cartoon effect transforms the input image into a cartoon-like representation by combining smoothing and edge-detection techniques. The image is first converted to grayscale, followed by the application of a bilateral filter for smoothing while preserving edges. Edge detection is then performed using adaptive thresholding. Finally, the smoothed image is combined with the detected edges, resulting in a stylized cartoon effect.

4.A short reflection of what you learned.

In this project, I have learned tasks related to basic image processing, and I have more understanding techniques such as bilateral filtering, edge detection, and adaptive thresholding for achieving specific visual effects.

This project is a great opportunity to learn form OpenCV library for various image processing operations, and it also give us a chance to make good practice for documenting functions with task descriptions

The most interesting thing in this project is that we are free to add extensions, which make me more involved in what I am interested in, and so that lets me learn better.

5.Acknowledgement of any materials or people you consulted for the assignment.

TAs:

Sasank Potluri, Zhizhou Gu, Poorna Chandra Vemula

OpenCV Tutorials:

https://docs.opencv.org/4.5.1/d9/df8/tutorial_root.html

YouTube:

Computer Vision Lab, “OPENCV & C++ TUTORIAL”,https://www.youtube.com/playlist?list=PLUTbi0GOQwghR9db9p6yHqwvzc989q_mu [access, Jan,26,2024]

Coding With Evan, “Cartoon Effect on Image using Python & OpenCV”, https://www.youtube.com/watch?v=v1dx6Q6k8k0 [access, Jan,26,2024]