# Project 1: Video-Special Effects

Samara Holmes
Northeastern University

## I. Introduction

The objective is to work with the OpenCV package and the mechanics of opening, capturing, manipulating, and writing images. This report displays the results of running several effects on a video stream, such as blurs, color filters, and more. This report aims to provide an overview of these filtering techniques and discussing their real-time video processing capabilities.

## II. Methodology

For this project we have the following tasks:

1) Read an image from a file and display it
2) Display [RGB] live video
3) Implement greyscale video stream
4) Implement alternative greyscale video stream
5) Implement a Sepia tone filter
6) Implement a 5x5 blur filter
    a) Native implementation
    b) Faster implementation
7) Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters
8) Implement a function that generates a gradient magnitude image from the X and Y Sobel images
9) Implement a function that blurs and quantizes a color image
10) Detect faces in an image
11) Use the Depth Anything V2 network to get depth estimates on your video feed
    a) Note: Unable to get Onnxruntime library running, implemented another video effect
12) Implement three more effects on your video
13) Build a GUI for this using Qt

## III. Results

Figure 1. Comparison between original and greyscale images

Figure 2. Customized greyscale image

Figure 3. Sepia tone filter

Figure 4. Comparison between original and 5x5 blur filter

Figure 5. Comparison between original and the gradient magnitude image

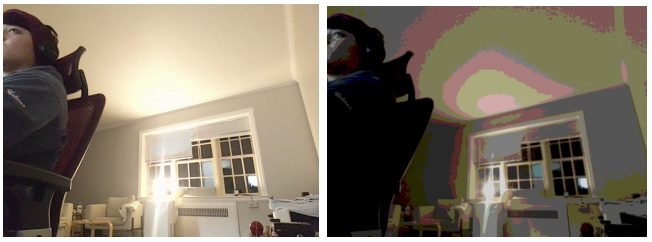

Figure 6. Comparison between the original and blurred/quantized image

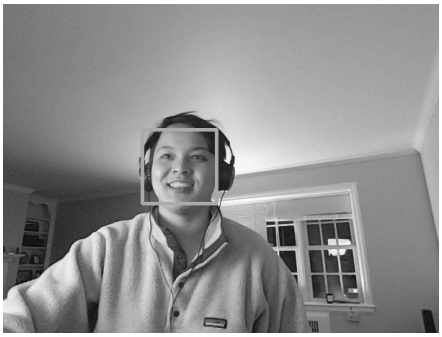

Figure 7. Face being detected in video stream

Figure 8. Image with vignetting

Figure 9. Image with face in color, background greyscale (extra effect 1)

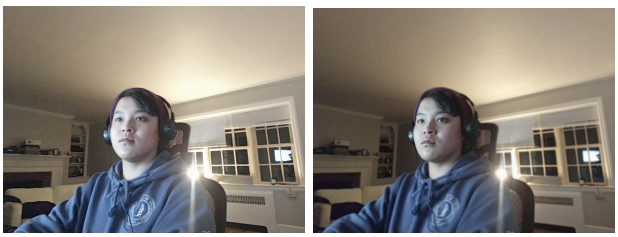

Figure 10.Comparison of no filter live capture and live capture, background blurred (extra effect 2)

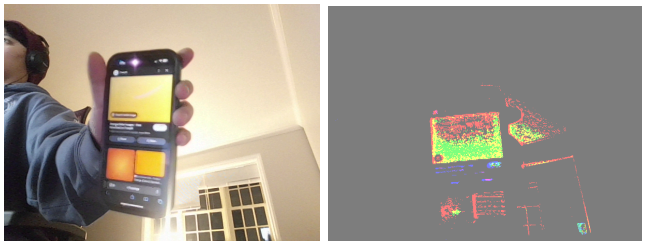

Figure 11. Comparison of original image and keeping one color but greying the rest (extra effect 3)

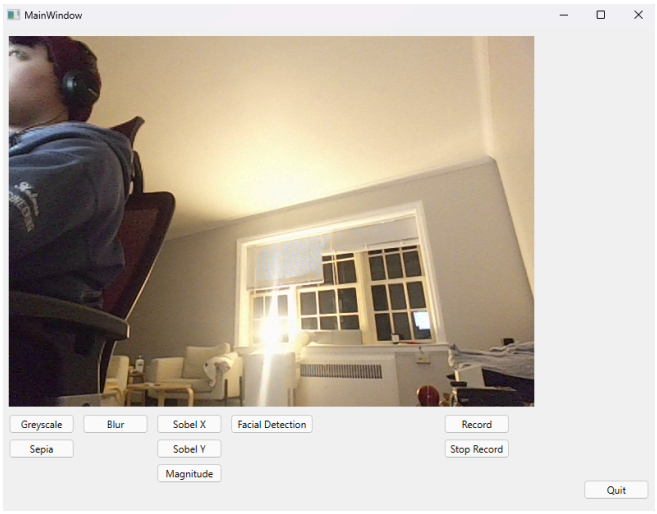


Figure 12. Custom GUI

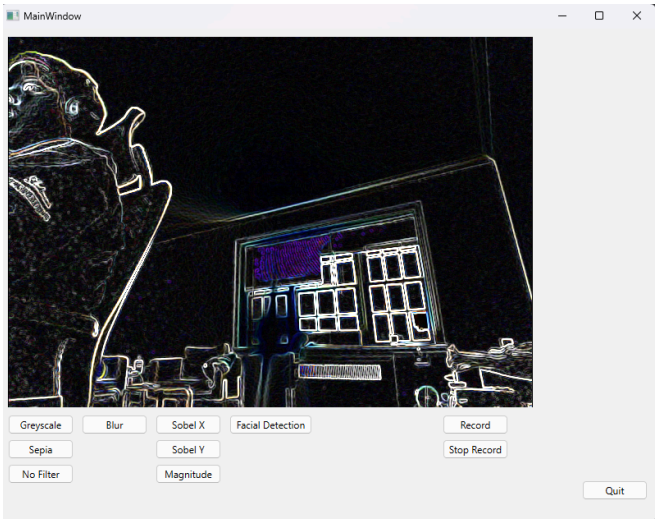


Figure 13. Custom GUI w/ magnitude filter running

## IV. Discussion

Figure 1 is a comparison of the original image coming out of my laptop and the cvtColor version of the greyscale image. For Figure 2, a greyscale image is also shown, however, this was done using the following:

```
Greyscale formula = 0.299R + 0.587G + 0.114B
```

To generate this version of the greyscale image, I first split the source image into its 3 channels (RGB). Then, using those channels, I was able to put them into the formula to get the desired result.

Figure 3, resulting in a sepia tone, was similar in that, I first needed to split the source image, but then I also needed to define a vector for the new channels. From there, I was able to use the following color coefficients:

```
0.272, 0.534, 0.131    // Blue
0.349, 0.686, 0.168    // Green
0.393, 0.769, 0.189    // Red
```

Using these coefficients, I did the following to get the new colors for the sepia tone where R, G, and B are the source values:

```
Sepia formula (for each new R,G,and B)
= coefficient*R + coefficient*G +
coefficient*B
```

Figure 4 is my implementations of the 5x5 blur filter. When I implemented the 5x5 blur filter using a for-loop, there was a clear latency. However, when I used separable filters, which ran a vertical blur, then a horizontal blur, the calculation was 10x faster. See table below.

Table 1. Comparison of blur filter calculation times

| Filter | 5x5 Blur | 5x5 Blur (separable filters) |
|---|---|---|
| Time per image (seconds) | 0.2331 | 0.0222 |

Figure 5 uses the Sobel X and Sobel Y filter functions, and calculates the gradient magnitude of the image. The gradient magnitude combines the edges and shows the strengths from each filter, giving a clear edge visualization.

Figure 6 shows a blurring and quantization of an image. It uses the blur filter generated previously to blur the video feed. Then, it uses a parameter to determine how each color channel should be quantized into.

Figure 7 demonstrates an implementation of facial detection. It classifies the faces detected based on the Haar cascade file. Using these detections, a box can be drawn around and applied to the video feed.

Figures 8-11 represent the extra effects that I decided to do for this project. I was unable to get the OnnxRuntime library running (required for DepthAnything) and kept getting the following error:

*The requested API version [20] is not available, only API versions [1, 17] are supported in this build. Current ORT Version is: 1.17.1*

Instead, as seen in Figure 8, I implemented a vignetting filter. This vignetting filter was based on the edges of the video steam, then darkens the corners.

Figure 9 demonstrates the ability to color a detected face while keeping the background in greyscale. To achieve this, the algorithm first converts the image to greyscale and detects faces. The detected face areas are then copied from the original colored image and overlaid onto the grayscale image.

Figure 10 demonstrates the ability to detect a face while blurring the background. This is done similarly to the algorithm in Figure 9. The algorithm first converts the image to greyscale and detects faces. The detected face areas are then copied from the original colored image and overlaid onto the blurred image.The image is blurred using the 5x5 blur algorithm.

Figure 11 demonstrates the ability to detect one color and keep it in the display, while greying out the rest. The color I chose is orange (since it's my favorite color). What I did was go to an RGB color creator, and defined the upper and lower bounds of the color I wanted to detect. From there, I obtained a mask, copied that mask to the destination image, then copied a grey matrix to the destination image.

Lastly, I generated a GUI to handle some of these filters (see Figures 12 and 13. It was initially difficult to link Qt6 into VSCode and into the same project folder I was using for the majority of the code, so I decided to create a new project using Qt Creator.

In Qt Creator, I was able to add in buttons and the video display into an application window. From there, I connected the buttons to functions I had previously created.

## V. Conclusion

In conclusion, by using the OpenCV library, we can effectively manipulate and analyze visual data. From opening and capturing images, to manipulating and writing them, OpenCV provides the necessary tools for various computer vision tasks.

The GUI creation extension using Qt was a new realm for me and displayed a creative way to demonstrate the OpenCV code I had written for this project.

## VI. References


https://github.com/DepthAnything/Depth-Anything-V2

https://youtu.be/yVeXXuGnnxs?si=VtucNgRLloLtSwrx

https://github.com/microsoft/onnxruntime/releases