Project 1: Real-Time Filtering

CS 5330: Pattern Recognition and Computer Vision

Arjun Rajeev Warrier

Abstract

Project 1 explores and demonstrates the use of the open-source OpenCV library for image filtering applications. From using pre-defined functions to defining custom functions in a custom library, we explore some functionalities that include blurring, gradient magnitude, quantization and layering of these functions. Accessing individual pixel values from Mat objects, reading from and writing to different locations and pointers were excessively used in this project. In addition, some extra functions were also included as a result of exploring the OpenCV documentation.

Experiment

The functionalities that were programmed as part of this project has been described below. This is a brief overview with short explanations and images that demonstrate the effect of these codes.

1. Read an image from a file and display it

A program, *imgDisplay.cpp* was coded to read an image, *project_img.jpg* stored at the local directory (project folder). A loop was also included to check for a user input of key 'q'; which would close the open window and terminate the program.

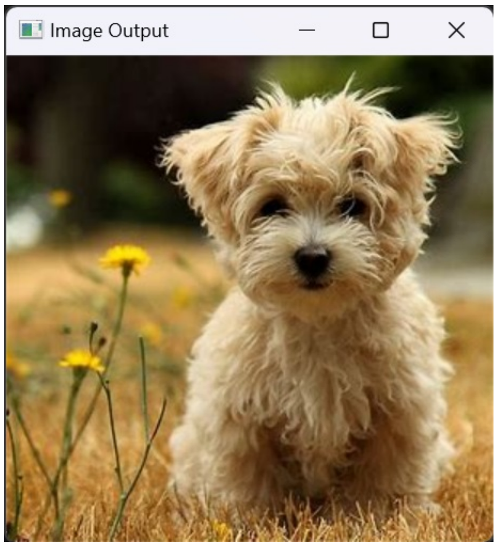
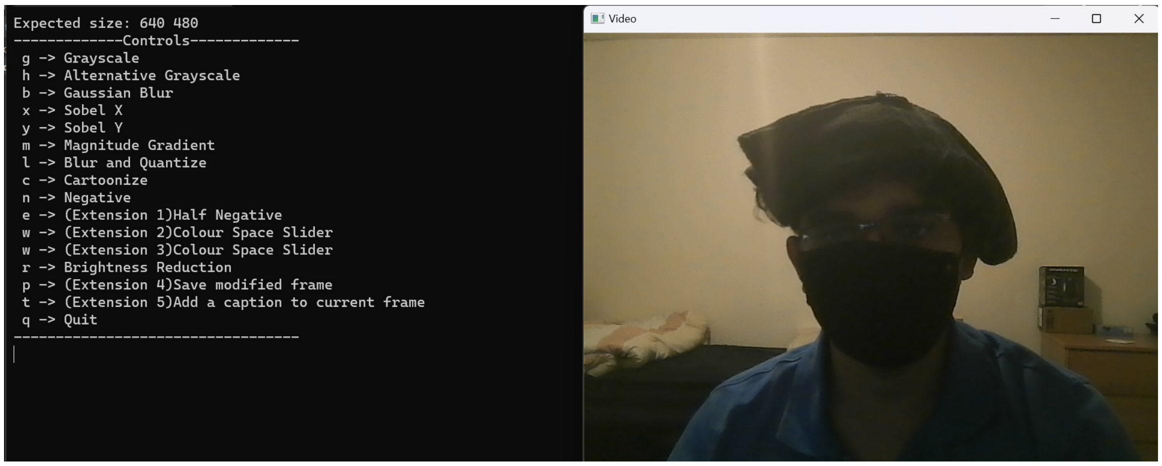


Figure 1: Image display by imgDisplay.cpp

2. Display live video

A program, *vidDisplay.cpp* was built to stream the webcam footage. This program functions by opening a video channel, creating a named window, capturing each frame from the channel/stream and then displaying those captured frame through a loop. Additionally, the functionalities of quitting (press $'q'$) and saving the current frame in the loop (press $'s'$) were included.



```
Expected size: 640 480
-------------Controls-------------
 g -> Grayscale
 h -> Alternative Grayscale
 b -> Gaussian Blur
 x -> Sobel X
 y -> Sobel Y
 m -> Magnitude Gradient
 l -> Blur and Quantize
 c -> Cartoonize
 n -> Negative
 e -> (Extension 1)Half Negative
 w -> (Extension 2)Colour Space Slider
 w -> (Extension 3)Colour Space Slider
 r -> Brightness Reduction
 p -> (Extension 4)Save modified frame
 t -> (Extension 5)Add a caption to current frame
 q -> Quit
----------------------------------
```

Figure 2: Sample real-time webcam frame

3. Display greyscale live video

The inbuilt openCV function of *cvtColor* was used to greyscale the real-time video capture. For this function, the $'g'$ key was assigned as the option for call. From the documentation, the following color weights were recorded :

$$BGR\ to\ GREY : Y \leftarrow 0.299 * R + 0.587 * G + 0.114 * B$$

This is recorded as the luminosity method[1], which is a more sophisticated version of the average method. It forms a weighted average of the color channels to account for human perception. Human eyes are more perceptive to green, so green is weighed more, then comes red followed by blue.

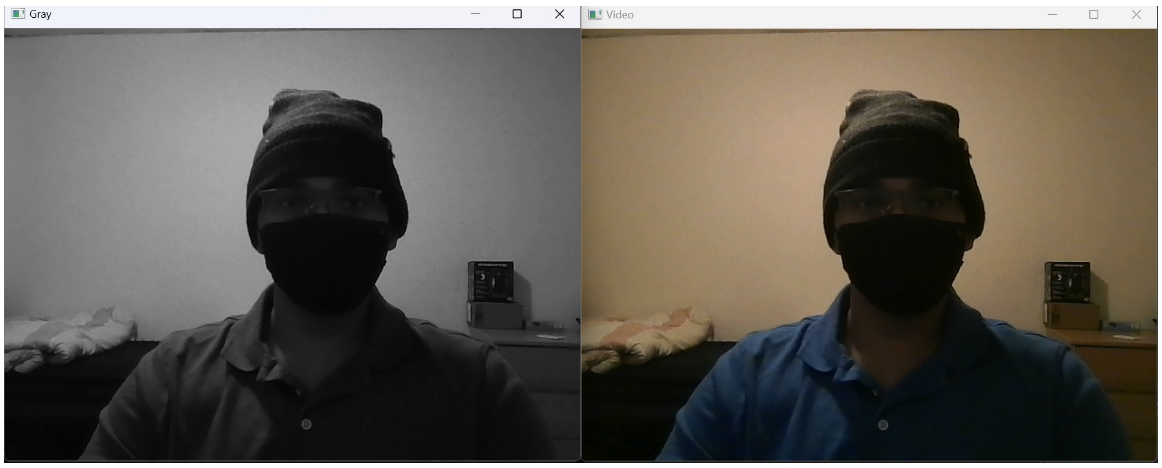


Figure 3: Greyscale version vs Original webcam Frame

4. Display alternative greyscale live video

For this function, the channel magnitudes were replaced by the average of the color channels. Unlike, the luminosity method followed by cvtColor, here all channels have equal weights for the average. There is a slight difference as can be seen between Fig.3 and Fig.4. The alternative greyscale version seems to be less brighter. The function call was assigned to $'h'$ key.

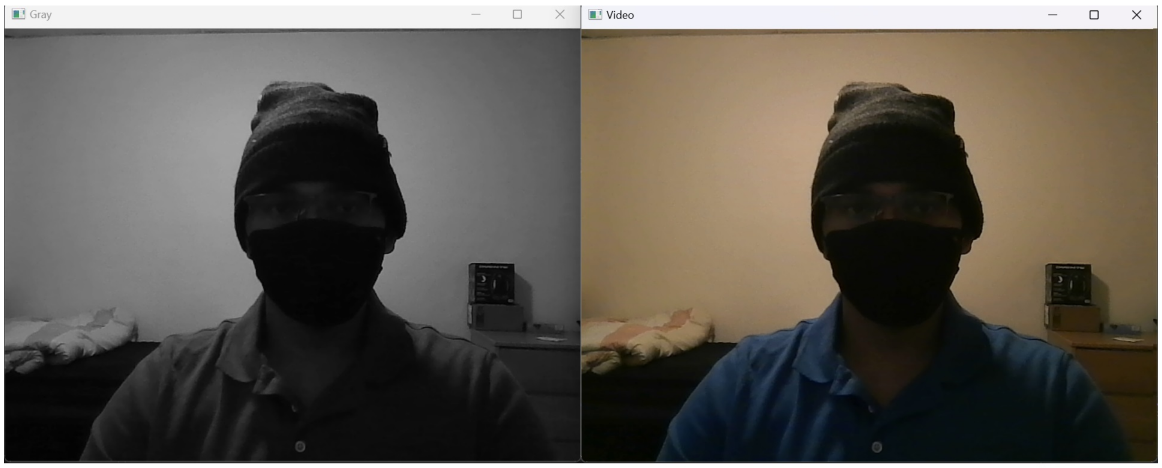


Figure 4: Alternative greyscale version vs Original webcam Frame

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

The Gaussian filter was implemented as a convolution with two 1x5 filters as opposed to a single 5x5 filter. This speeds up the computation by drastically reducing the number of calculations required. The input and output are colored images. The filter used for convolution was $[1;2;5;2;1]$, both vertically and horizontally. This function was mapped to $'b'$ and results in a blurred image.To handle the edge cases, mirroring the rows and columns next to the edges would result in images not having a black outline.

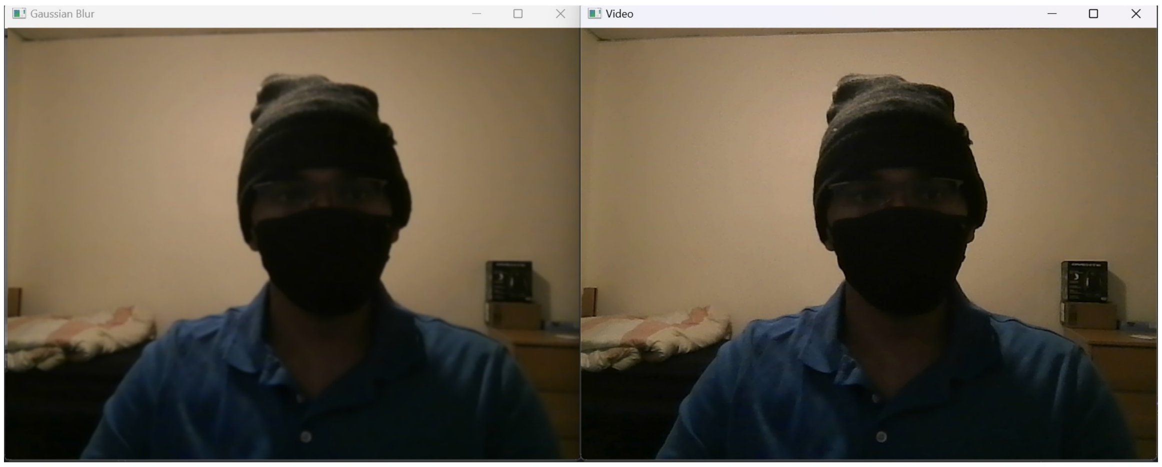


Figure 5: Alternative Gaussian Blurred output vs Original webcam Frame

6. Implementing 3x3 Sobel X Filter

The Sobel X filter is used to find the horizontal gradient, in other words by passing an image through a sobel x filter, the vertical edges in that image are obtained. Here, the source image (webcam frame) has been convolved with two 1x3 filters instead of a 3x3 filter to increase efficiency. The output of both are the same however. The filters were, first vertical $[1; 2; 1]$ and then horizontal $[-1; 0; 1]$. To handle the edge cases, mirroring the rows and columns next to the edges would result in images not having a black outline. This function was mapped to the $'x'$ key. Fig. 6 shows the vertical edges pronounced clearly while the horizontal are not visible.

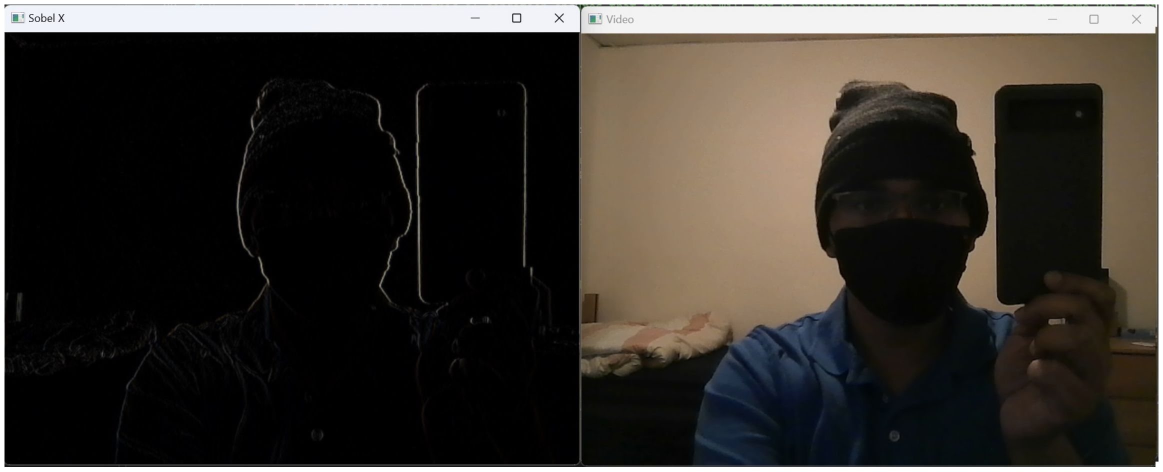


Figure 6: Sobel X filter output vs Original webcam Frame

7. Implementing 3x3 Sobel Y Filter

Sobel y filter is used to find the vertical gradient, in other words the horizontal edges are obtained. Here too, the source image (webcam frame) has been convolved with two 1x3 filters instead of a 3x3 filter to increase efficiency. The filters were, first horizontal $[1; 0; -1]$ and then vertical $[1; 2; 1]$. To handle the edge cases, mirroring the rows and columns next to the edges would result in images not having a black outline. This function was mapped to the $'y'$ key. Fig. 7 shows the horizontal edges pronounced clearly while the vertical are not visible.

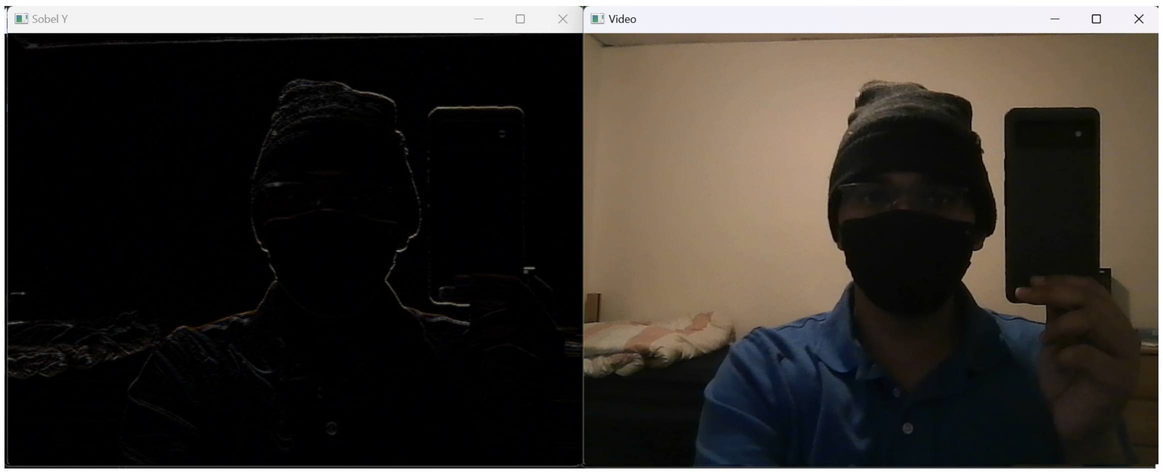


Figure 7: Sobel Y filter vs Original Frame

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

The gradient magnitude image was obtained by using the sobel X and Y results together. The values for each corresponding pixel were used in a Euclidean distance equation $\sqrt{sx^2 + sy^2}$. This generates an image with all edges pronounced clearly as seen in Fig. 8. This function was mapped to the $'m'$ key. The previously defined sobel X and sobel Y functions were called for the sx and sy values in this function.

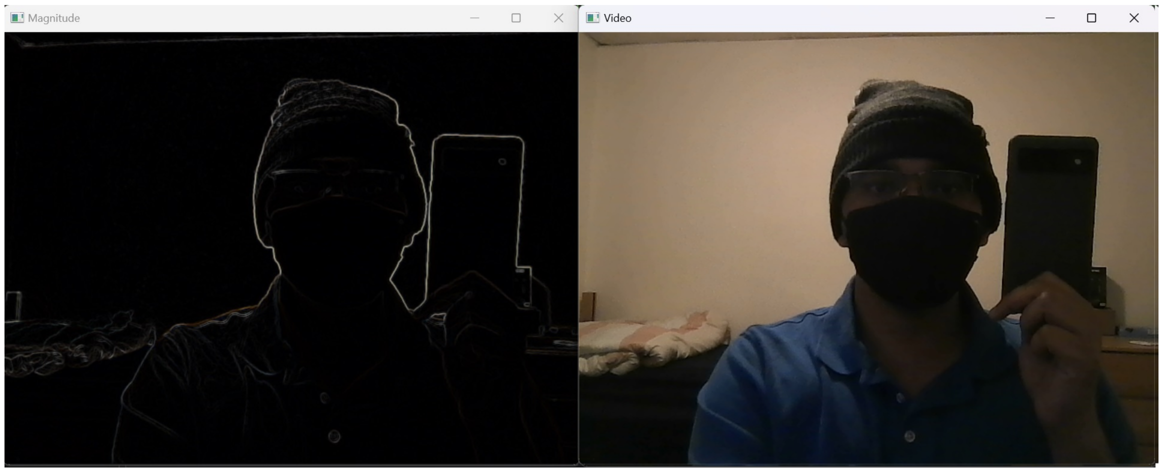


Figure 8: Gradient Magnitude Frame vs Original Frame

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

Here, the frame was first blurred using the earlier coded Gaussian function. The blurred output is then quantized into a fixed number of levels. These levels determine the total number of magnitudes visible in the output. The formula b = $\frac{255}{levels}$ gives the number of buckets, according to which each pixel value is modified using the equations, $x_t = \frac{x}{b}$ and $x_f = x_t$*b. This gives a blurred and quantized output as seen in Fig.9. This function was mapped to the $'l'$ key.

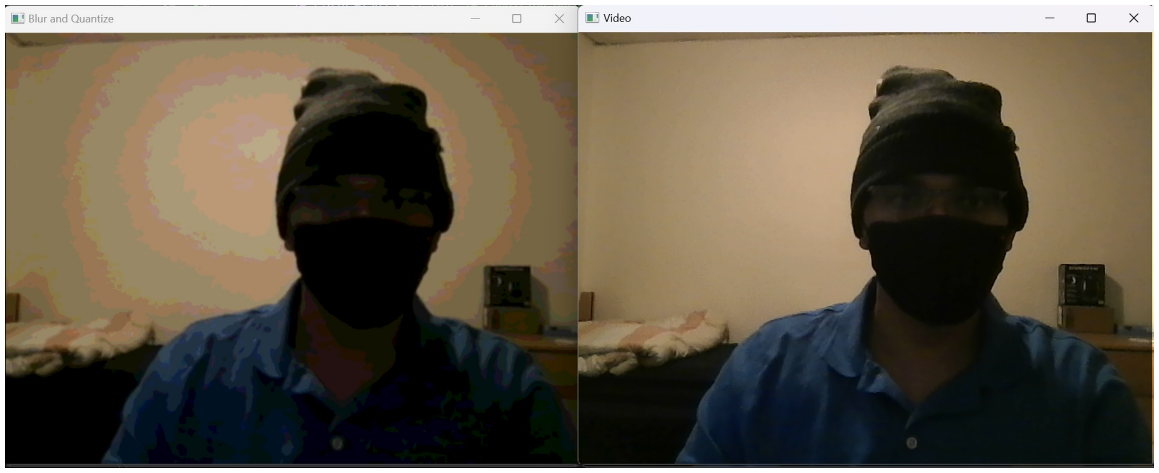


Figure 9: Blurred and Quantized Output vs Original webcam Frame

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

This functionality has been implemented by first passing the captured frame through the blur and quantize function defined earlier. This result is then modified by changing the pixels at strong edges to black. These pixels are decided by comparing each pixels gradient magnitude (from the gradient magnitude function) against a threshold. This cartoonization function has been mapped to the $'c'$ key.

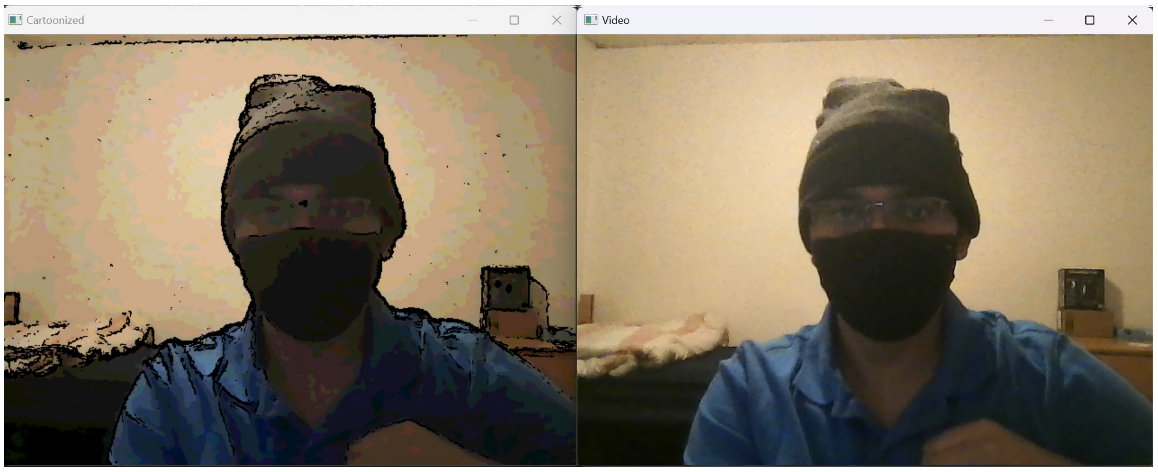


Figure 10: Cartoonized Frame vs Original Frame

11. Pick another effect to implement on your video

A negative filter was coded for this part. All the pixel values are accessed and their color channel values are inverted by subtracting them from 255 (max magnitude). In other words, their complement has been stored. This function was mapped to the 'n' key.

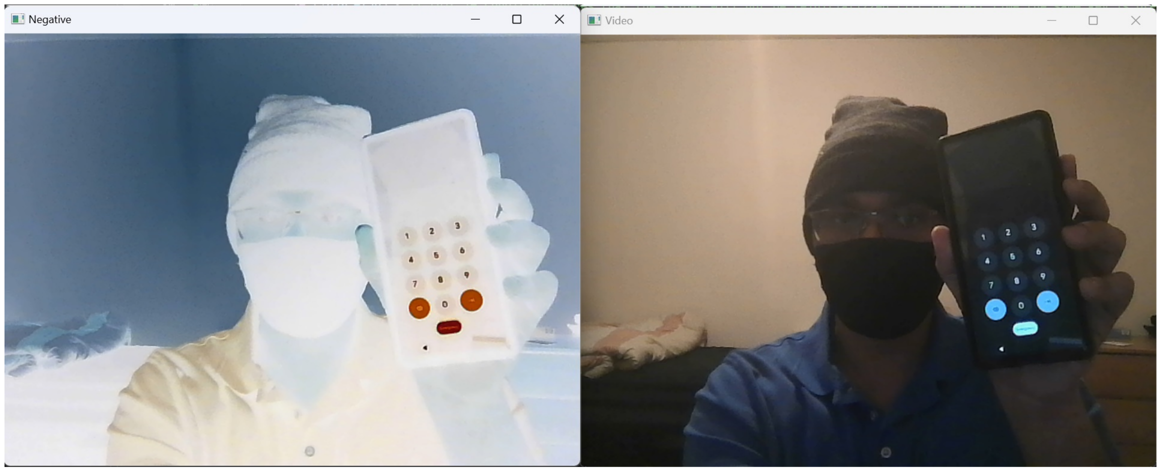


Figure 11: Negative output vs Original Frame

12. Extensions

Five extra functionalities have been added to this project.

1. Half screen negative : This function returns an output with half the screen complemented. This function was implemented to show an easier comparison between a filtered and non-filtered output. Fig.12 shows the half screen.

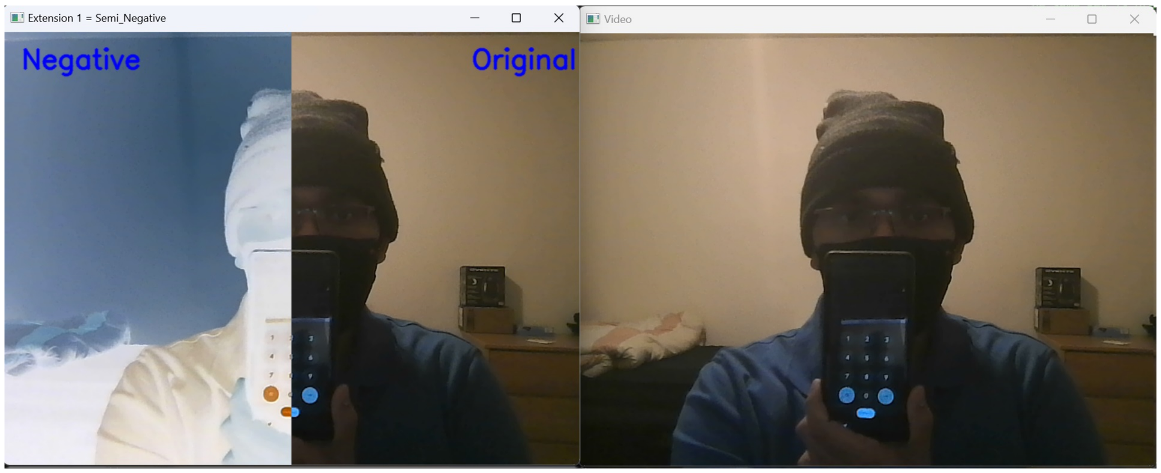


Figure 12: Half screen negative Output

2. Colour Space slider : This function gives a slider that can be used to decide which color space to view the image in. In this, RGB, HSV, Lab and greyscale have been implemented using the cvtColor function. Fig.13 shows an output of this function.

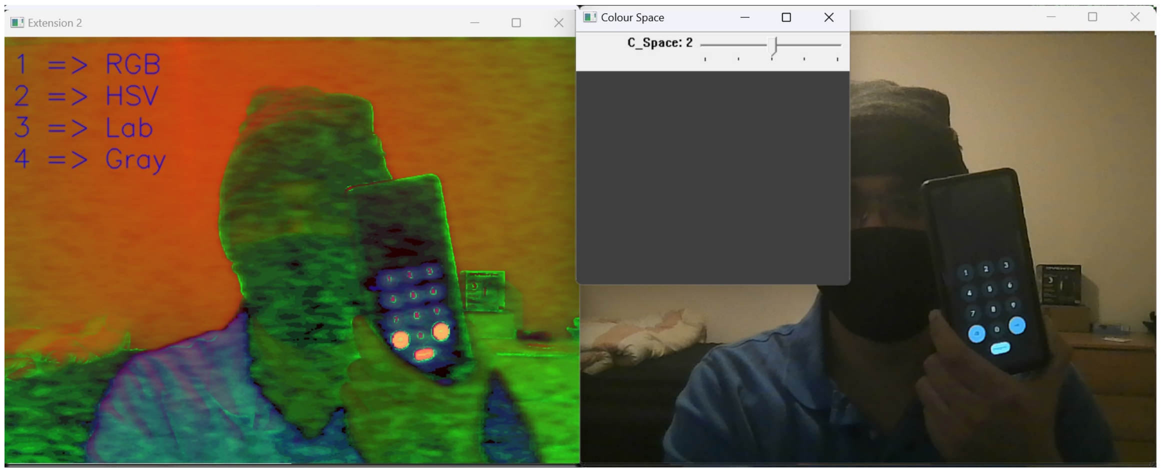


Figure 13: Different color spaces

3. Light reduction: By reducing the magnitudes of the color channels, a dimming effect was obtained. The level to which the dimming takes place is decided by a slider. Fig.14 shows an image dimmed by a factor of 5.

4. Save modified frames: This lets the user save the modified/filtered outputs from all the previously implemented functions to the local directory.

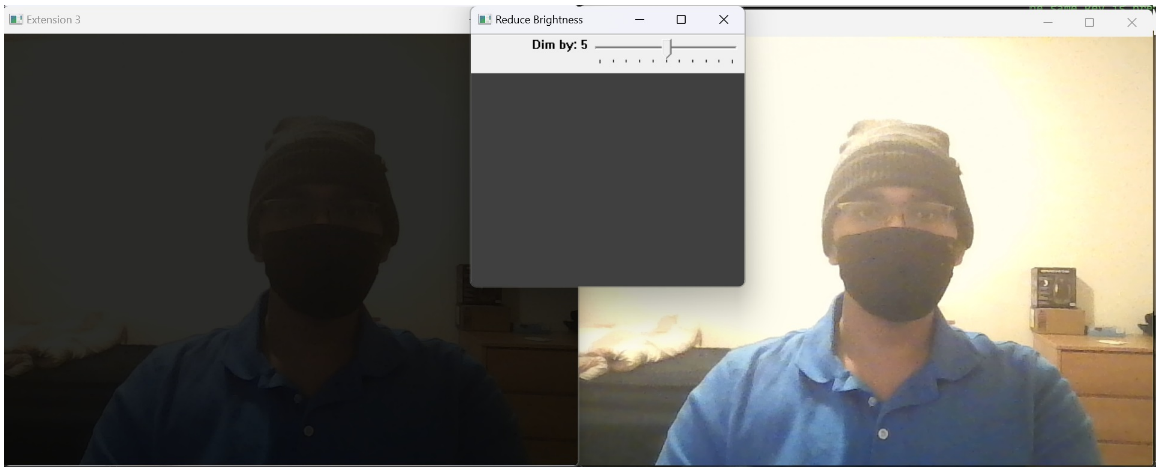
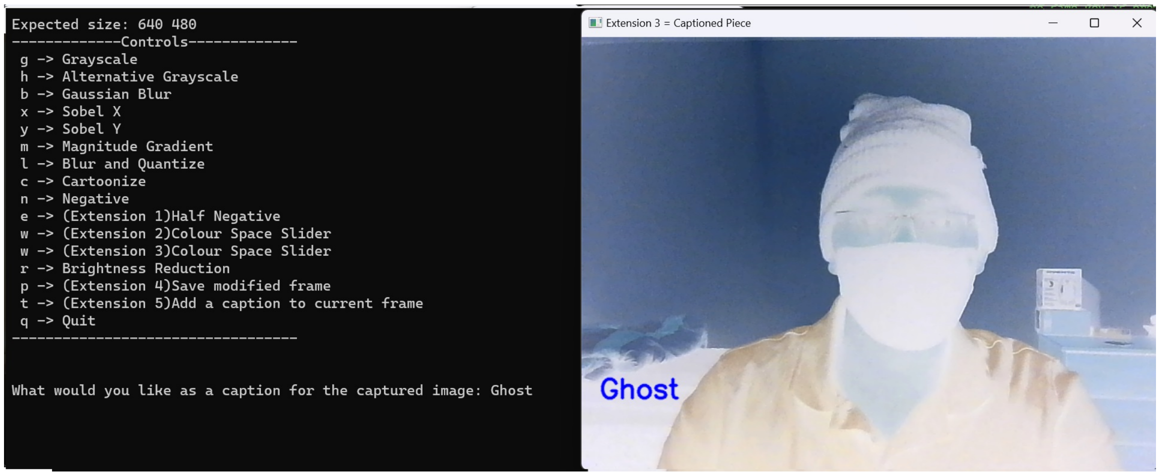


Figure 14: Light reduction

5. Captioning: This lets the user add a caption to the output of any of the previous functions. This captioned image can then be stored using the previous function that saves the modified output. Fig. 15 shows a caption added to the output of the negative function.



```
Expected size: 640 480
------------Controls------------
 g -> Grayscale
 h -> Alternative Grayscale
 b -> Gaussian Blur
 x -> Sobel X
 y -> Sobel Y
 m -> Magnitude Gradient
 l -> Blur and Quantize
 c -> Cartoonize
 n -> Negative
 e -> (Extension 1)Half Negative
 w -> (Extension 2)Colour Space Slider
 w -> (Extension 3)Colour Space Slider
 r -> Brightness Reduction
 p -> (Extension 4)Save modified frame
 t -> (Extension 5)Add a caption to current frame
 q -> Quit
--------------------------------


What would you like as a caption for the captured image: Ghost
```

Figure 15: Captioning

Reflections

Though this project, basic usage of OpenCV functions for image access and modification were covered in C++. Some key takeaways for this project were:

- The usage of separable filters to save time and obtain exactly same outputs from convolution. The concept of how blurring happens was also explored. Applying a 5x5 gaussian filter and 3x3 sobel filters as 1x5 and 1x3 filters help speed up the computation by more than half the time.

- The fact that most modern day social media filters can be obtained by layering classical filters was really surprising. Especially, how easy it was to obtain a cartoonization effect from just blurring, quantising and blackening some edges.

- The ease of exchangeable custom coded libraries was also new. This improves and simplifies understanding moodularity as a concept of object oriented programming. The library developed in this program can now be used in other projects.

References

[1] https://docs.opencv.org/

[2] Winnemöller, H., Olsen, S.C. and Gooch, B., 2006. Real-time video abstraction. ACM Transactions On Graphics (TOG), 25(3), pp.1221-1226.

[3] https://www.baeldung.com/cs/convert-rgb-to-grayscale