NORTHEASTERN UNIVERSITY

CS 5330: Pattern Recognition and Computer Vision (Spring 2022)

PROJECT - 1

REAL-TIME FILTERING

Submitted By:
TARUN SAXENA (002979327)

I. Description

Real-time filtering project involves multiple challenging tasks related to filtering images/frames from live video. This project's goal is to get users acquainted with C/C++, the OpenCV package, and the operations of reading, extracting, processing, and writing images. The tasks are as follows:

1. Read an image from a file and display it
2. Display live video
3. Display greyscale live video
4. Display alternative greyscale live video
5. Implement a 5x5 Gaussian filter as a separable 1x5 filter
6. Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters
7. Implement a function that generates a gradient magnitude image from the X and Y Sobel images
8. Implement a function that blurs and quantizes a color image
9. Implement a live video cartoonization function using the gradient magnitude and blur/quantize filters
10. Pick another effect to implement in the video

For each task different key presses were assigned. The above processing tasks are performed using OpenCV libraries in C++ programming language in Xcode IDE *(Version 14.2)* on MacBook Pro (M1 Chip).

II. Tasks

1. Greyscale using OpenCV cvtColor function:

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>Fig.1 Original Frame</td><td>Fig.2 Greyscaled Frame</td></tr>
</table>

The original frame (Fig.1) is converted to a greyscale frame (Fig.2) using OpenCV cvtColor function, **COLOR_RGBA2GRAY**. This conversion is done using the function:

$$Y \leftarrow 0.299xR + 0.587xG + 0.114xB$$

Individual weights are applied to RGB color channels of the original frame to get a grayscale frame. This conversion is done when the user enters the ‘g’ key.

2. Alternative greyscale:

	
Fig.3 Original Frame	**Fig.4 Alternate Greyscaled Frame**

The original frame (Fig.3) is converted to a customized greyscale frame (Fig.4) by setting each pixel value of the destination frame equal to the average of the RGB channel values of that particular pixel from source frame. This conversion is done using the function:

$$Y \leftarrow (R + G + B)/3$$

This conversion is done when the user enters the ‘h’ key.

3. 5x5 Gaussian filter:

	
Fig.5 Original Frame	**Fig.6 Blurred Frame**

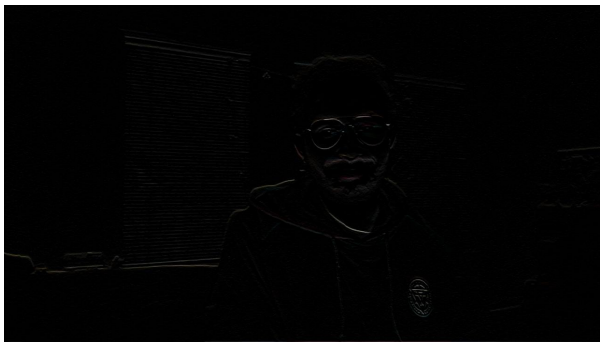
The original frame (Fig.5) is converted to a blurred frame (Fig.6) using a 5x5 Gaussian filter. A 5x5 Gaussian filter is implemented as two separable 1x5 filters, one [1 2 4 2 1] vertical filter and another [1 2 4 2 1] horizontal filter. This conversion is done when the user enters the ‘b’ key.

4. 3x3 Sobel X filter:

	
Fig.7 Original Frame	**Fig.8 Sobel X Frame**

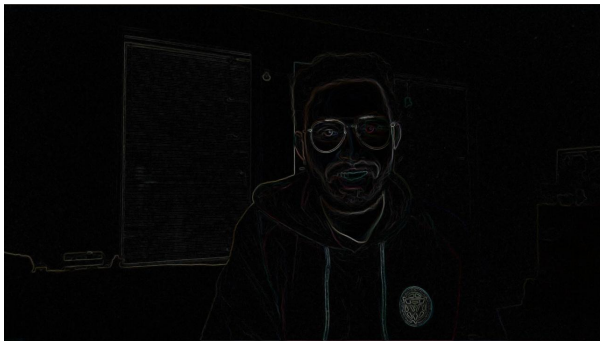
The original frame (Fig.7) is converted to a filtered frame (Fig.8) using a 3x3 Sobel filter. A 3x3 Sobel X filter is implemented as two separable 1x3 filters. This filter displays areas where the color rapidly changes as we go horizontally. This conversion is done when the user enters the ‘x’ key.

5. 3x3 Sobel Y filter:

	
Fig.9 Original Frame	**Fig.10 Sobel Y Frame**

The original frame (Fig.9) is converted to a filtered frame (Fig.10) using a 3x3 Sobel filter. A 3x3 Sobel Y filter is implemented as two separable 1x3 filters. This filter displays areas where the color rapidly changes as we go vertically. This conversion is done when the user enters the ‘y’ key.

6. Gradient magnitude filter:

	
Fig.11 Original Frame	**Fig.12 Gradient Magnitude Frame**

The original frame (Fig.11) is converted to a filtered frame (Fig.12) by a function that generates a gradient magnitude image based on Euclidean distance for magnitude. This conversion is done using the function:

$$L \leftarrow \text{sqrt}(sx*sx + sy*sy)$$

, where sx and sy are outputs from Sobel X and Sobel Y. This conversion is done when the user enters the ‘m’ key.

7. Blurred and Quantized filter:

	
Fig.13 Original Frame	**Fig.14 Blurred and Quantized Frame**

The original frame (Fig.13) is converted to a filtered frame (Fig.14) by a function that generates a blurred and quantized image into a specific number of levels.

Firstly the frame is blurred. The size of the Bucket will be, b = 255/levels. Then, given a color channel value x, execute xt = x / b, followed by xf = xt * b.
This is done first by using a blur filter and then This conversion is done when the user enters the 'l' key.

8. Cartoonization filter:

	
Fig.15 Original Frame	**Fig.16 Cartoonized Frame**

The original frame (Fig.15) is converted to a cartoonized frame (Fig.16) by the following steps:

1. Calculate the gradient magnitude of the frame.
2. Blur the frame.

3. Quantize the frame.
4. Modify the generated frame by converting any pixels with a gradient magnitude greater than a threshold to black, i.e. {0,0,0}.

This conversion is done when the user enters the ‘c’ key.

9. Some other Effects:

9.1. Increase/Decrease brightness:

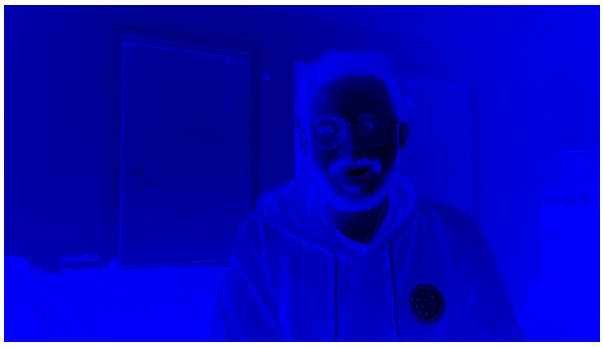
	
Fig.17 Original Frame	**Fig.18 Bright Frame**

The original frame (Fig.17) is converted to a bright frame (Fig.18) by adding a custom value to the function. This conversion is done when the user enters the ‘1’ numerical key.

	
Fig.19 Original Frame	**Fig.20 Dim Frame**

The original frame (Fig.19) is converted to a dim frame (Fig.20) by subtracting a custom value to the function. This conversion is done when the user enters the ‘2’ numerical key.

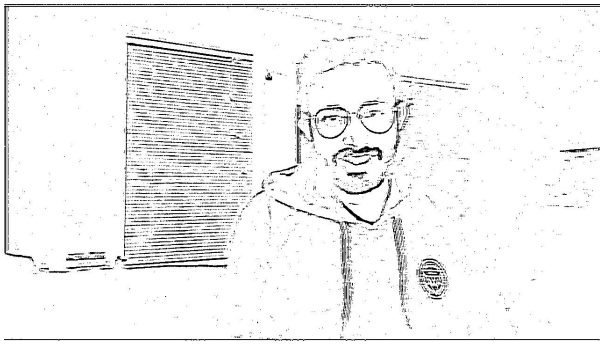
9.2. Make the image a negative of itself:

	
Fig.21 Original Frame	**Fig.22 Negative Frame**

The original frame (Fig.21) is converted to a negative frame (Fig.22) by subtracting the source frame pixel values from 255. This conversion is done when the user enters the ‘n’ numerical key.

10. Extensions:

10.1. Pencil Sketching:

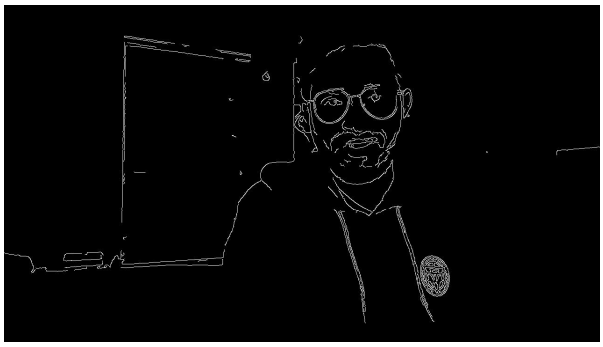
	
Fig.23 Original Frame	**Fig.24 Sketch Frame**

The original frame (Fig.23) is converted to a negative frame (Fig.24) by the following steps:

1. Converting to greyscale
2. Blurring the frame
3. Using the Laplacian operator to find edges
4. Inverting the image

This conversion is done when the user enters the ‘p’ numerical key.

10.2. Canny Edge Filter:

	
Fig.25 Original Frame	**Fig.26 Canny Edge Frame**

The original frame (Fig.25) is converted to a Canny Edge frame (Fig.26) by the OpenCV Canny function, **Canny**. This conversion is done when the user enters the ‘e’ numerical key.

10.3 Text on Frame:

	
Fig.27 Original Frame	**Fig.28 Text on Frame**

On the original frame (Fig.27) text is projected and (Fig.28) is generated. The user is prompted for a text and using the putText function we specify the Point, font, and text color to be added on the frame. This conversion is done when the user enters the ‘t’ numerical key.

10.4 Sepia Filter:

	
Fig.29 Original Frame	**Fig.30 Sepia Frame**

The original frame (Fig.29) is converted to a Sepia frame (Fig.30). The Sepia filter gives a frame a warm reddish-brown tint. This conversion is done when the user enters the 'o' key.

III. Learning Outcomes

This project on Real Time Filtering which focuses on filtering real time images was both interesting and challenging to work. I learnt OpenCV's basics programmed in C++ through this project. From reading, writing to applying filters on frames, this project covered them all. This was a great learning experience for me as I was not only using OpenCV for the first time but also Xcode too. In the initial phase I faced some difficulties while linking libraries and image not loading but al those issues were overcome with time and help from Teaching Assistants.

IV. Acknowledgments

I am thankful to Prof. Bruce Maxwell for designing and executing a wonderful learning experience in Computer Vision. I would also like to extend my gratitude to this course's TA's for clearing the doubts whenever required. The materials that have helped in the completion of this project are:

1. CS 5330 Course Materials
2. OpenCV documentation: https://docs.opencv.org/3.4/index.html
3. OpenCV in Xcode installation Video :
   OpenCV 4 in Xcode // Download and Installation Guide
4. StackOverflow