Project 1: Real-time Filtering
CS 5330: Pattern Recognition & Computer Vision

Author: *Rishabh Singh*

Instructor:	*Prof. Bruce Maxwell*
Department:	Computer Science

Contents

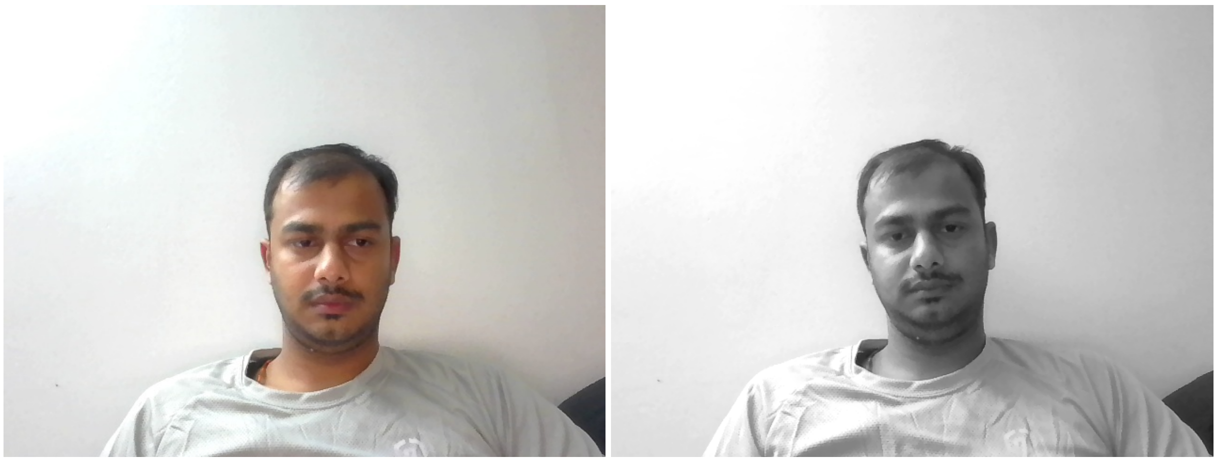

Figure 1: Original and cvtColor() version of the greyscale image

1 Summary

In this project, I created a program in C++ that uses openCV to implement multiple filters. To write the code in Windows, I used the Visual Studio IDE. The project shows a live video with various filters and effects achieved with the C++ and OpenCV libraries. I implemented Greyscale, Sobel, Gaussian filter, blurring images, cartoon images,sparkles into the image where there are strong edges are all available, and mirror images.

2 Displaying Greyscale live video

First, I accessed the live video coming from our webcam and then convert it to a grayscale image using header files of OpenCV. We use function cvtcolor() to convert the image to it's Grayscale equivalent. This function does require changes to the Red, Green, Blue variants of color. This filter allows user to change frame to greyscale by pressing key 'g'.
Refer Fig. 1 for implementation image.

3 Displaying Alternative greyscale live video

In this section, I converted an image from BGR to HSV color space and then extracted the V channel of the HSV image which corresponded to the intensity channel of the image, resulting in a grayscale image. This filter allows user to change frame to greyscale by pressing key 'h'.

- Using split(), I splitted the destination image (which is now in HSV color space) into 3 separate images, one for each channel (H, S, and V) and these 3 images are stored in the "hsv" vector.
- Using hsv[2]: Assigned the value of the 3rd element of the "hsv" vector (which is the V channel) to the destination image. This effectively replaced by the HSV image with the V channel, resulting in an intensity image.

Refer Fig. 2 for implementation image.

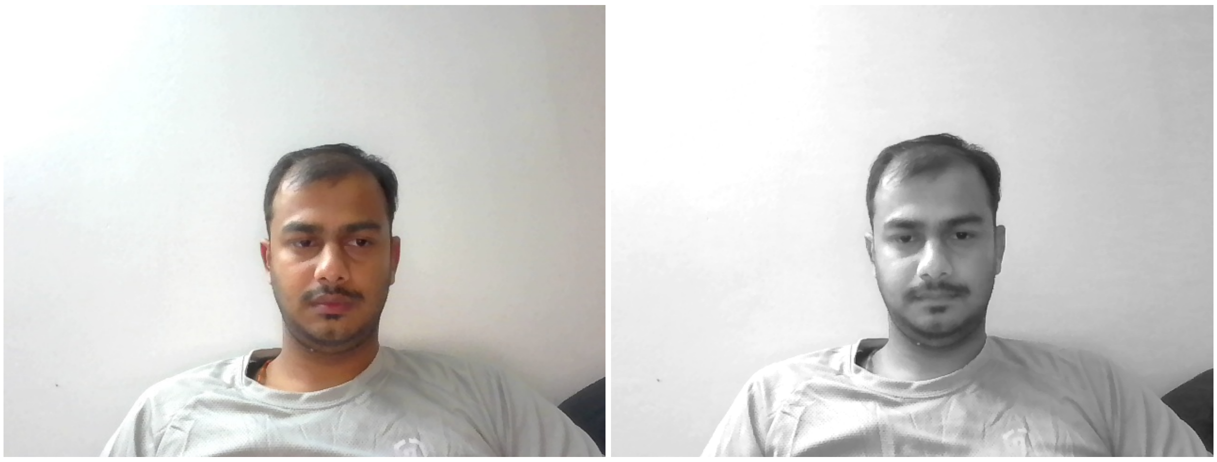

Figure 2: Original and customized greyscale image

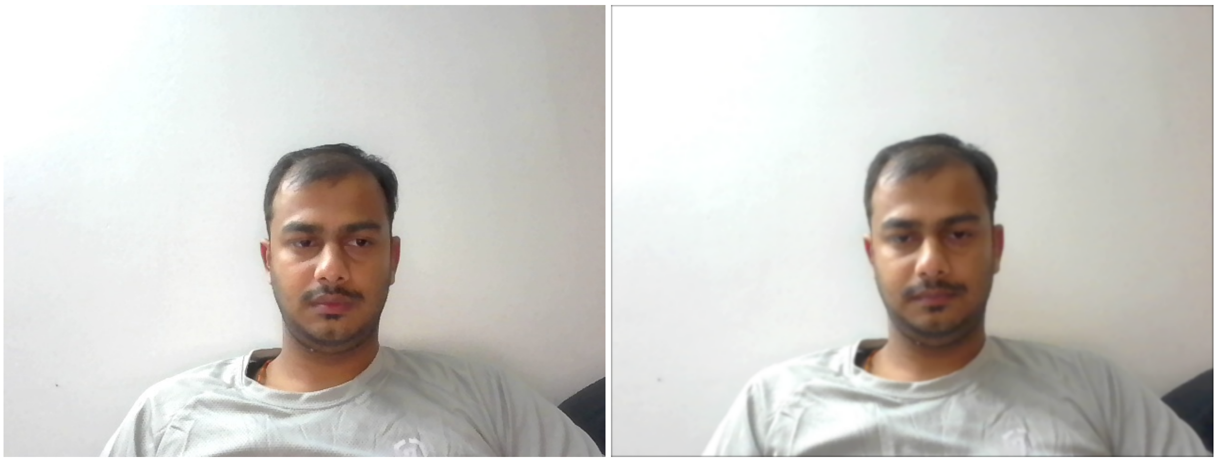

Figure 3: Original and the blurred image

4 Implementing a 5x5 Gaussian filter as separable 1x5 filters

A Gaussian Filter is a low pass filter used for reducing noise (high frequency components) and blurring regions of an image. The filter is implemented as an Odd sized Symmetric Kernel (DIP version of a Matrix) which is passed through each pixel of the Region of Interest to get the desired effect. This filter allows user to change frame to blur by pressing key 'b' .

- Using split(), I splitted the destination image (which is now in HSV color space) into 3 separate images, one for each channel (H, S, and V) and these 3 images are stored in the "hsv" vector.
- Using hsv[2]: Assigned the value of the 3rd element of the "hsv" vector (which is the V channel) to the destination image. This effectively replaced by the HSV image with the V channel, resulting in an intensity image.

Refer Fig. 3 for implementation image.

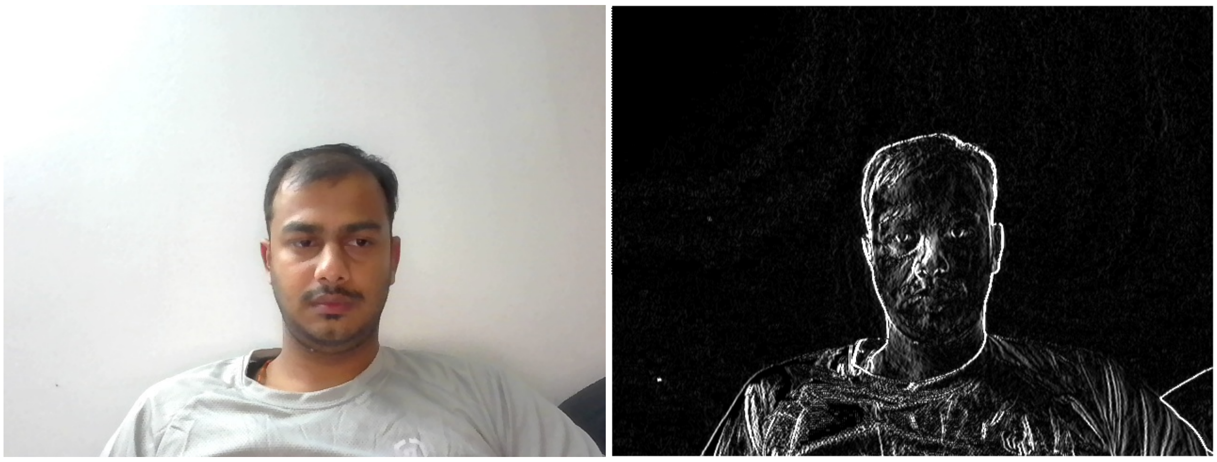
Figure 4: Original and the vertical edge detector image

5 Implementing a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges.
Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.
The operator consists of a pair of 3×3 convolution kernels. One kernel is simply the other rotated by 90°.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix} S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

5.1 3x3 Sobel X (Vertical Edge Detector)

This filter draws attention to areas where the color changes quickly as we move horizontally. By pressing the 'x' key, the user can change the frame to Sobel X. Refer Fig. 4 for implementation image.

5.2 3x3 Sobel Y (Horizontal Edge Detector)

This filter emphasizes areas where the color rapidly changes as we move vertically. By pressing the key 'y,' the user can change the frame to Sobel Y. Refer Fig. 5 for implementation image.

6 Implementing a function that generates a gradient magnitude image from the X and Y Sobel images

Sobel kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Sx and Sy). These can then be combined together to

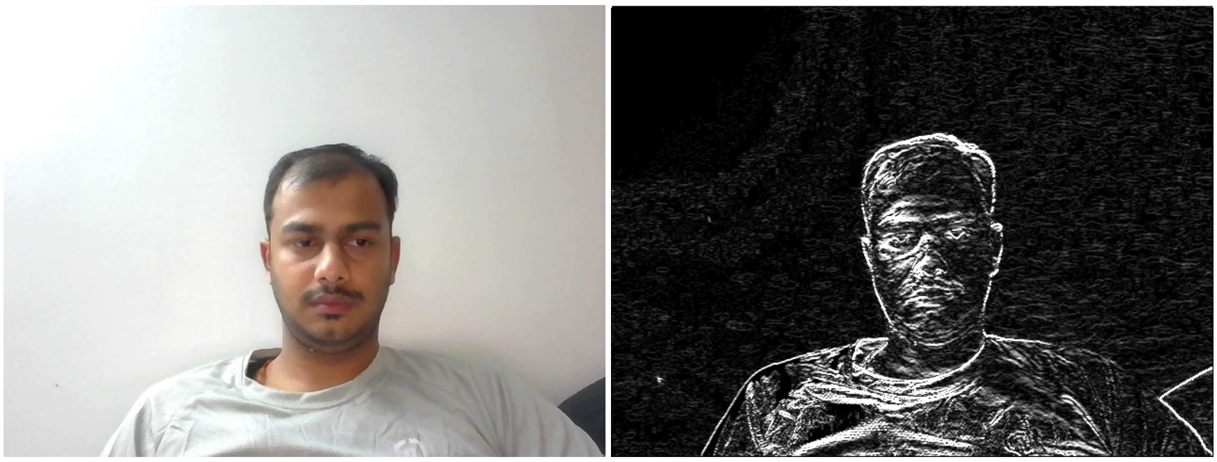

Figure 5: Original and the horizontal edge detector image

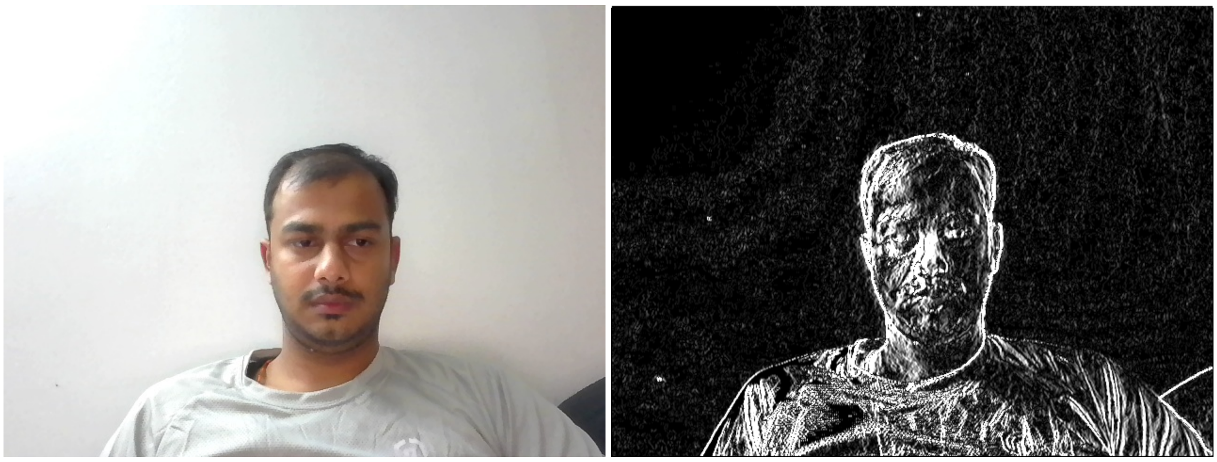

Figure 6: original and the gradient magnitude image

find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|S| = \sqrt{({S_x}^2 + (S_y)^2} \tag{2}$$

$$|S| = |S_x| + |S_y| \tag{3}$$

$$\theta = \arctan(G_y/G_x) \tag{4}$$

This filter detects the edges of the color change. By pressing the 'm' key, the user can change the frame to Gradient Magnitude. Refer Fig. 6 for implementation image.

7 Implementing a function that blurs and quantizes a color image

In this section, the blur filter is used first, and then the image is quantized into a fixed number of levels as specified by a parameter. This filter detects the edges of the color change. By pressing the 'i' key, the user can change the frame to Blur Quantize. Refer Fig. 7 for implementation image.

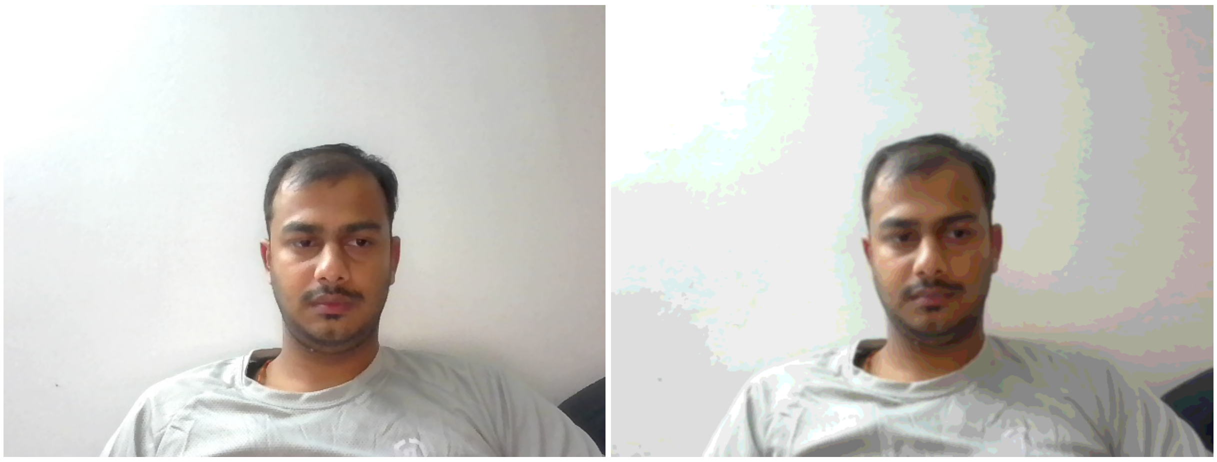

Figure 7: original and blur quantized image

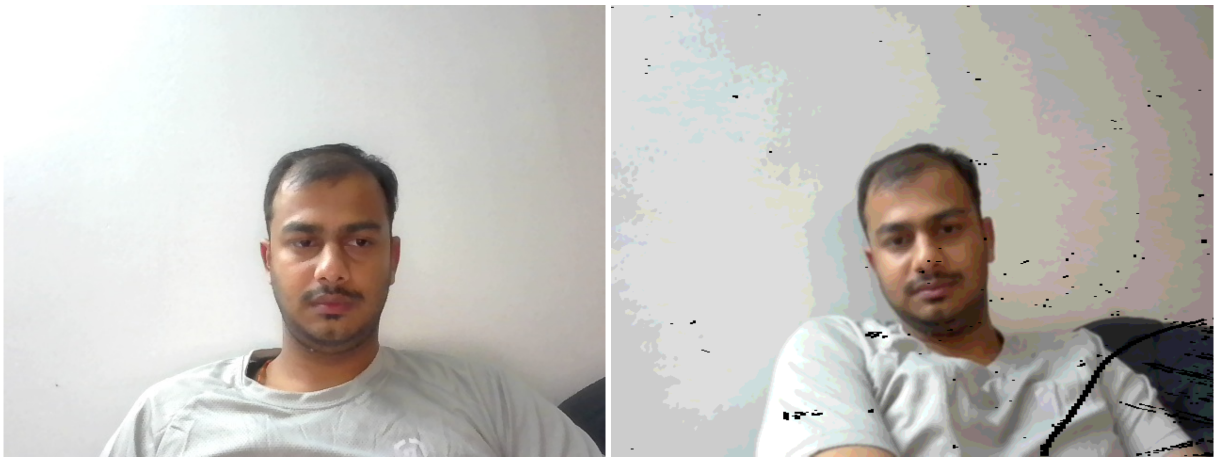

Figure 8: original and the cartoonized image

8 Implementing a live video cartoonization function using the gradient magnitude and blur/quantize filters

On the original frame, using SobelX and SobelY, this filter applies a gradient magnitude filter. On the original image, this filter also applies the Blur and Quantize filters. The filter will then darken the pixels with gradient magnitudes greater than a certain threshold. By pressing the ’c’ key, the user can change the frame to Cartoonization. Refer Fig. 8 for implementation image.

9 Implementing another effect - sparkles into the image where there are strong edges

The Canny algorithm aims to satisfy three main criteria:

- Low error rate: Meaning a good detection of only existent edges.
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.

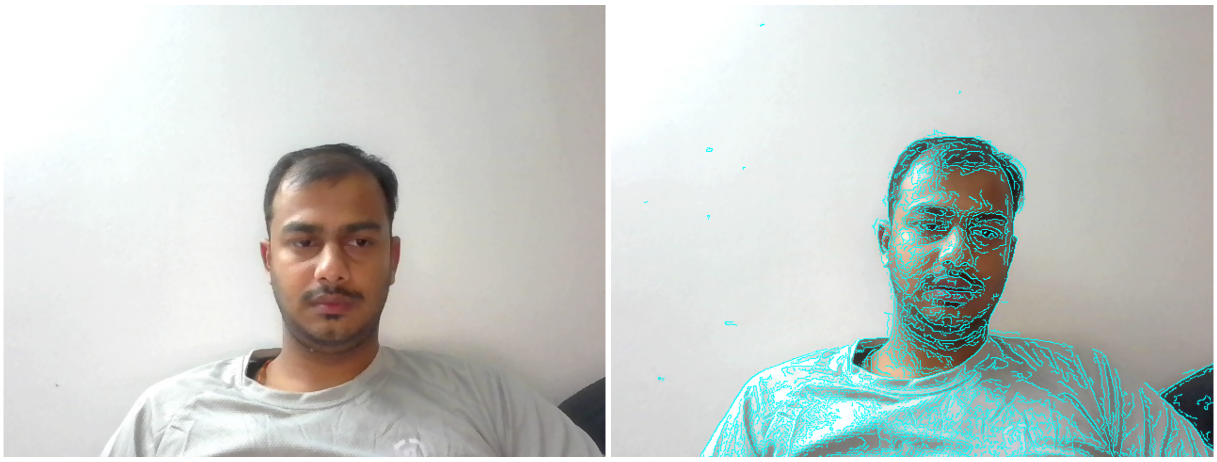

Figure 9: original and the sparkled image

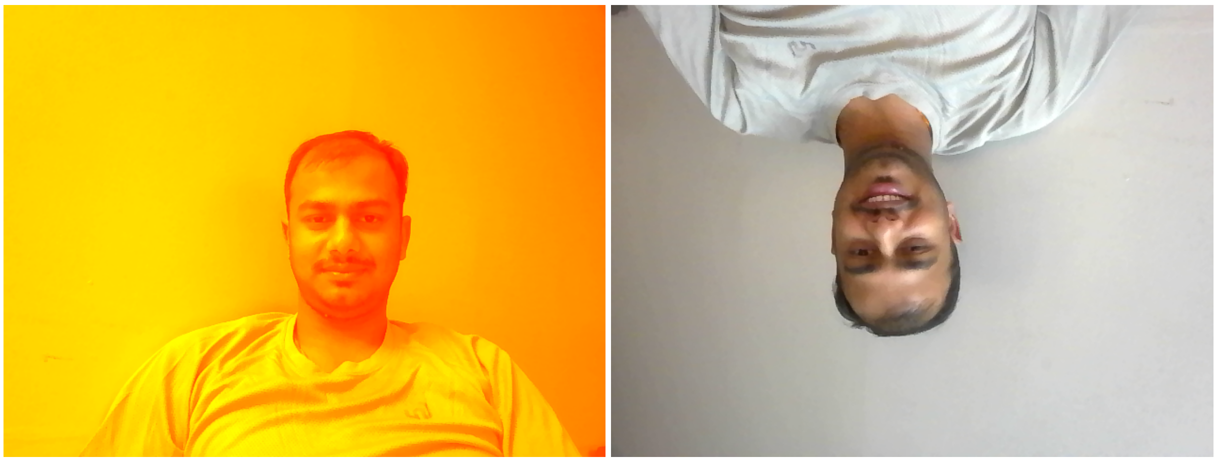

Figure 10: Original and Mirror Image in X and Z axis

- Minimal response: Only one detector response per edge.

Refer Fig. 9 for implementation image.

10 Extension

10.1 Mirroring the image in X and Z axis

I used the flip function which is inbuilt in the opencv library. flip function takes 3 parametres.

- First is mat object for input image
- Second is mat object for another desired output image
- last parameter is integer value 0 or 1

0 is used if desired output is water image and 1 is used if the desired result is the mirror image of the corresponding input image. Then I've also applied colormap function in x-axis mirror image. Refer Fig. 10 for implementation image.

11 Acknowledgement

- https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm
- https://cppsecrets.com/users/2031103105114105115104100115991154957641
  C00-OpenCv-cvcvtColor.php
- https://cppsecrets.com/users/2042115104117981049710910851535049641031
  C00-OpenCV-cvflip.php
- TAs and my friend Harshit