CS 5330 - Pattern Recognition and Computer Vision
Project 1 : Real-time filtering

Name : Soorya Vijayaragavan
NUID : 002921909

Introduction:

The main aim of this project is to understand the functionalities of various filters that are used in the real time filtering process and build it from scratch without directly using the inbuilt functions. This project helped in understanding the various additional effects being used by us in editing our images and how those works. Other objectives of the project include getting used to the OpenCV environment with C++.

Tasks:

1. Read an image from a file and display it:

   This task was done using the imread() function, by importing an image from the project folder. And it was displayed by creating a new window. And The functionality to close the window when 'q' is pressed was also set.

2. Display live video:

   To perform the capturing a new window was created and the cap function was used to capture the live frames, it was put under an infinite loop to capture continuously and display the output each time through the loop. And additional functionalities as mentioned, like when 'q' is clicked the program quits and when 's' is pressed that frame is saved in the project folder using the imwrite function.

3. Display grayscale live video

   The same above steps were followed to display the live video, but when the 'g' is pressed the function **cvtColor(frame, gray_frame,COLOR_BGR2GRAY);** was used to convert the normal video to grayscale. Flag values are used inside the loop to update the action that is to be performed . Below are the frames captured from a live video, which displays the normal image and the converted grayscale image.

Fig.1 a) Original Image, b) Grayscale Image which were captured from live video

4. Display alternative greyscale live video:

   When ‘h’ is pressed an alternate greyscale version of the live video is to be displayed for this to be done, copying one color channel to the other two is a method used. The value of the blue color channel was copied to both the red and green channel.By doing so the blue channel of the video is spread evenly across the red and green channels and it creates a grayscale image.

Fig.2 a) cvtColor(grayscale) , b) Alternate grayscale

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

   A 5x5 gaussian filter can be implemented as a 5x1 and 1x5 filter by splitting it in two parts. It is done to reduce the computation time. It gives the advantage of implementing the convolution of two smaller kernels which reduces both the amount of data to be processed and the amount of the memory used. Coming to our implementation it is given that both 5x1 and 1x5 filters have the same values {1,2,4,2,1}. Initially a temp image of the same size and type as the original image filled with zeros is created. Kernel 1 is now convolved with the image and the values are normalized and stored in the temp image created.These values are calculated leaving out all the four extreme rows and columns. Now the kernel2 is convoluted with the temp image and the calculated values are normalized and are saved in the final output image. The gaussian filter actually blurs

the image and removes the high frequency components. The results obtained are given below,

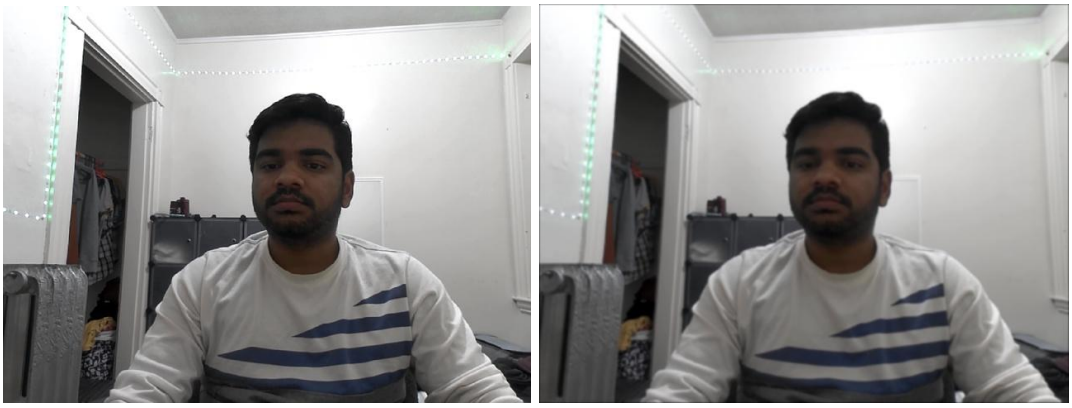

Fig.3 a) Original Image, b)Blurred Image

6. Implement a 3x3 Sobel X and 3x3 Sobel Y filter as separable 1x3 filters

Basically Sobel is an edge detection filter. The Sobel filter works by convolving an image with two 3x3 kernels, one for detecting horizontal edges(sobel Y) and one for detecting vertical edges(sobel X). The output of the convolution is the gradient of the image intensity at each point, which is then used to detect edges in the image. Sobel X 3x3 filter can be separated as [1 2 1] in vertical and [-1 0 1] in horizontal. Sobel Y 3x3 filter can be separated as [-1 0 1] in vertical and [1 2 1] in horizontal. Same type of convolution is done both in Sobel X and Sobel Y with a change in kernels.The output of the filter is a binary image, with edges represented by white pixels and non-edges represented by black pixels. In our case the output is assigned to be signed short. Here we do not take into account the outer rows and columns in the convolving part.The result obtained are shown below,

Fig.4 a) Sobel X, b) Sobel Y

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

The output of both Sobel X and Sobel Y filters is given as input to this function. It is done by calling the already created function.Then, the Euclidean distance can be computed for each of the 3 channels using the formula I = sqrt(sx*sx + sy*sy). This will

give us 3 separate gradient magnitude images for each of the 3 channels. Then the 3 separate gradient magnitude images are combined into a single uchar color image . This can be done by combining the 3 images into a single 3-channel image, where each channel contains the same gradient magnitude value. Finally, it is important to note that the gradient magnitude image should be normalized to the range of 0 to 255 before it is converted to a uchar image, so that the image can be properly displayed.

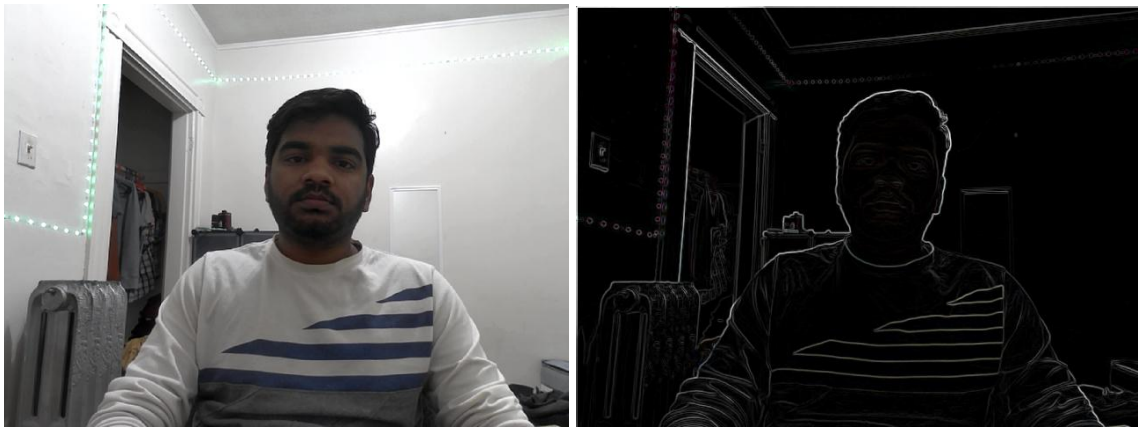

Fig.5 a)Original Image, b)Gradient Magnitude Image

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

Blurring of the image is done by calling the initially created blur function. Moving on to the quantization process,the first step in this process is to figure out the size of a bucket. The size of a bucket can be calculated by dividing 255 (the maximum possible value for a color channel) by the number of levels we want to quantize to. This can be represented as b = 255/levels. Next, we need to take a color channel value x and apply the quantization. To do this, we first execute xt = x/b. This value xt represents the number of buckets that a particular color channel value falls into. We can then execute xf = xt*b which gives us the quantized value for that particular color channel. The result of this process will be a color image that has been blurred and quantized into a fixed number of levels.

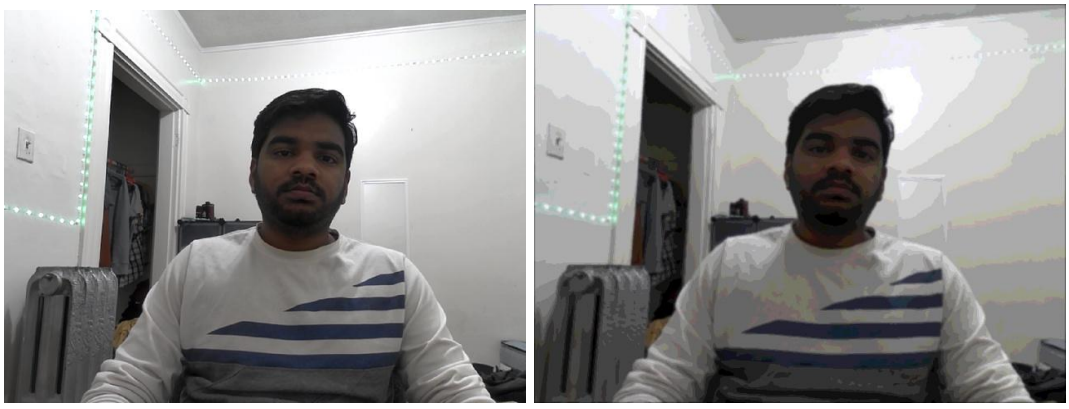

Fig.6 a)Original Image, b) Blurred and Quantized Image(Level=15)

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

The process starts by getting the captured frame, the level value the image is to be quantized and a threshold as input. To calculate the gradient magnitude of the image, the previously created magnitude function is called. The next step is to blur and quantize similarly as that function was also created earlier, the function calling is done. The final step is to set to black any pixels with a gradient magnitude larger than the threshold. By doing so we get the cartoon version of the original image. The output obtained is displayed below,

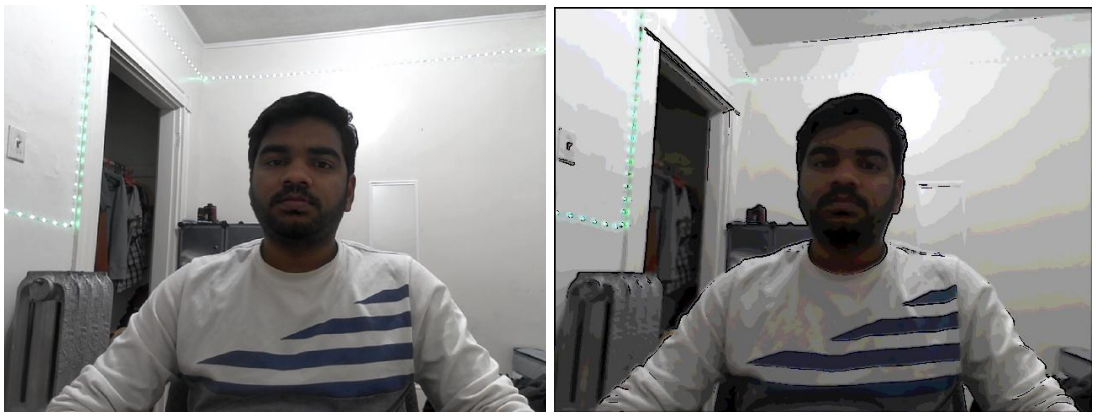

Fig.7 a)Original Image, b) Cartoon Image (Level=15, Threshold=15)

Extensions

10. Obtaining the Negative of the Original Image

In order to obtain a negative image, the values of the pixels in the image must be reversed. This means that the dark pixels become lighter, and the lighter pixels become darker. This done by subtracting the pixel value from 255, this actually converts the image to its negative form.The result obtained is displayed below,

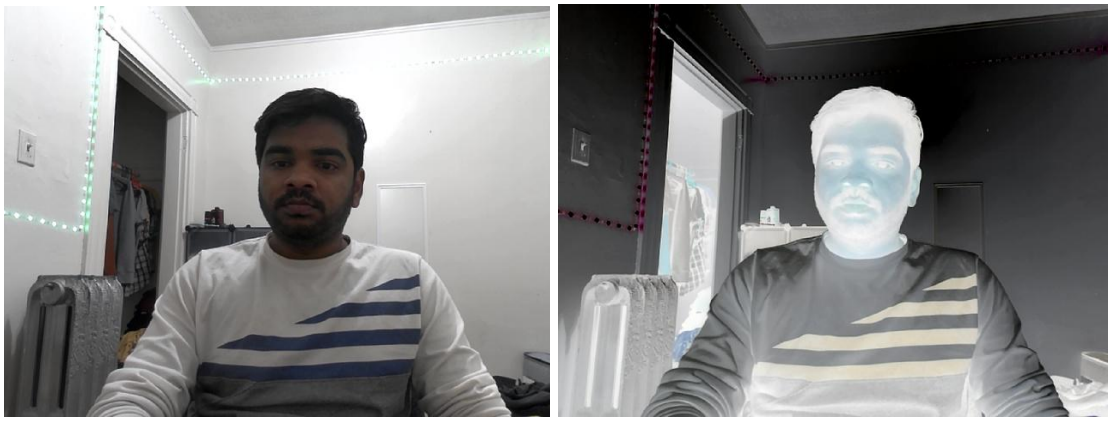

Fig.8 a)Original Image, b) Negative Image

11. Changing Saturation Value of Image

In this task, initially the cvtColor() function was used to convert the image from the BGR to HSV. This cvtColor() function takes in an image and a conversion code, which specifies the type of color conversion desired. Then the saturation value was adjusted by simply changing the second channel (saturation value) of the HSV image. After changing the saturation value, the image is converted back to BGR form.When the saturation value is increased, colors become more vivid and intense. When the saturation value is decreased, colors become more muted and washed out.

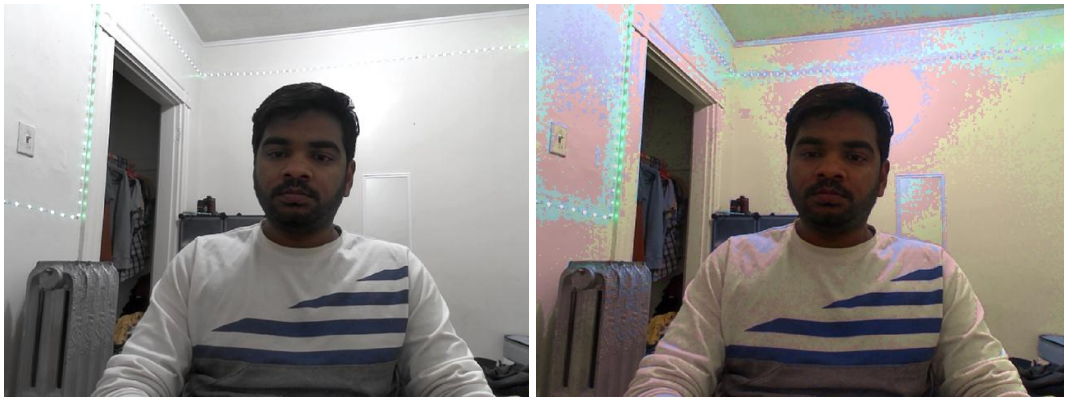

Fig.9 a)Original Image, b) Saturation Value Changed Image(increased by a value of 50)

12. Sharpening the Image:

Sharpening an image is done by applying a filter to the image that increases the contrast of adjacent pixels. A sharpening filter is a convolution filter that is used to make an image appear sharper. It increases the contrast between adjacent pixels, giving the image a sharper look. The above mentioned filter is a 3x3 filter which is as ([0, -1, 0,], [-1, 5, -1],[0, -1, 0]). The function filter2D() was used to apply the filter to the image. This filter2D() function takes an image and applies a convolution filter to it.The final output gives a sharpened image.

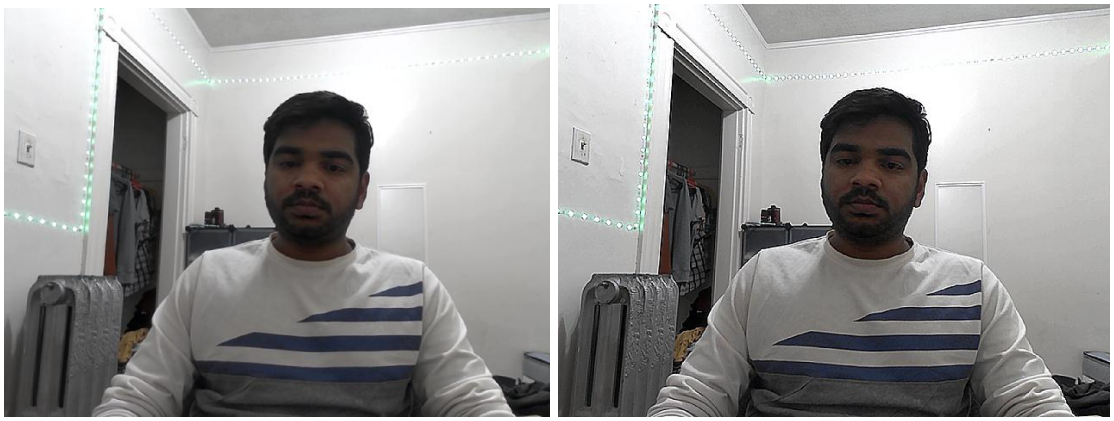

Fig.10 a) Original Image, b) Sharpened Image

Conclusion:

This project has given a good understanding of OpenCV, we were able to develop an efficient and effective method of filtering video streams in real-time.

References:

https://www.geeksforgeeks.org/reading-and-displaying-an-image-in-opencv-using-c/
https://docs.opencv.org/3.4/files.html
https://www.opencv-srf.com/p/introduction.html
https://www.bogotobogo.com/cplusplus/opencv.php
http://opencv-tutorials-hub.blogspot.com/