CS5330: Pattern Recognition and Computer Vision

Project 1: Real-time filtering

Name: Krishna Prasath Senthil Kumaran

Email: senthilkumaran.k@northeastern.edu

Aim of the Project:

This project helps us to familiarize ourselves with the concepts of C/C++ and various OpenCV packages. The project majorly focuses on creating various types of filters which use the convolution method to interact with the pixels of the image. An output is generated as a result of manipulation. We use multiple types of filters in this project such as **Greyscale**, **Gaussian**, **3x3 Sobel filter,** and much more.

A video feed is passed through the functions as an input and a frame of the live feed is captured through a key press and is processed by the filters. This project also introduces various OpenCV library functions as well as to few in-built functions as well.

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

In task 1, an image is read from a location and displayed in a window. The letter ***'q'*** was added to the program as a function key. Its functionality was to close the window that displays the image and quit the program.

Task 2: Display live video

In task 2, a program is created under "***vidDisplay.cpp"*** to open the video channel. Another key 's' is added as a function key that captures the current frame in the video channel. The captured frame is displayed in another window named ***"image"***; the image is proportionately small compared to the actual video feed. When the key ***'q'*** is pressed, it closes the image window and stops the program as well.

Task 3: Display greyscale live video

In task 3, the vidDisplay.cpp file is updated with a greyscale filter that uses OpenCV "***cvtColor"*** function. The letter ***'g'*** key is added as a function key which captures the current frame and converts the image into a greyscale image using **"*cv::COLOR_BGR2GRAY"***. Fig.1 shows the transformation.

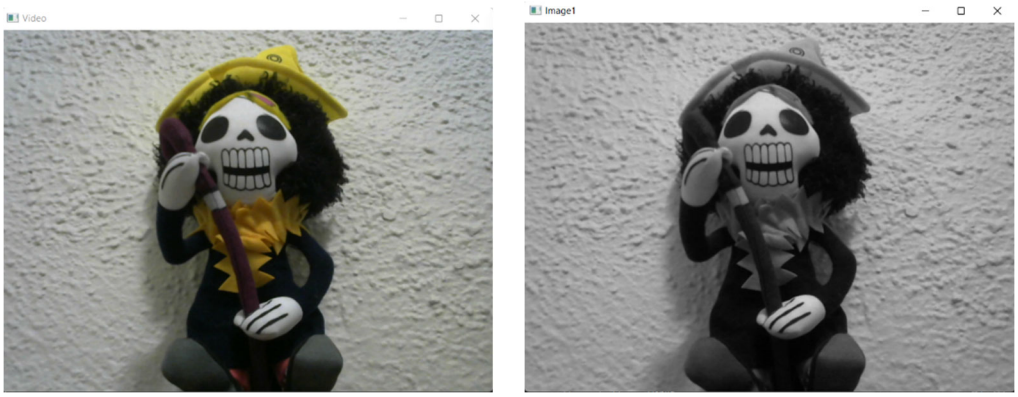

Fig.1 Image 1 captured from the video and converted into greyscale.

Task 4: Display alternative greyscale live video

In task 4, a custom greyscale filter is used that accesses each pixel and the average of three channels under every pixel is calculated and applied to each pixel. The convoluted image is stored in ***'Mat dst'***. The transformed image is viewed in ***"Image2"*** window. This filter is executed on the press of the letter ***'h'***. Fig.2 illustrates the before and after applying the filter.

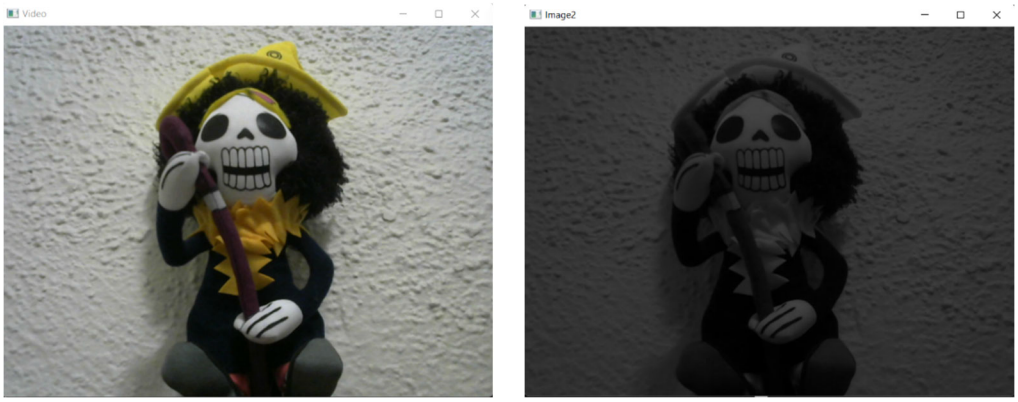

Fig.2 Greyscale image created from averaging of RGB channels.

Task 5: Implement a 5x5 Gaussian filter as separable 1x5 filters

In task 5, a 5x5 gaussian filter is applied through two linear filters. The pixels are looped through, only when there are at least two pixels in the surrounding. This filter produces a slightly blurred image, the image is portrayed in ***"Image3"*** window. This filter can be accessed when the letter ***'b'*** is pressed. Fig.3 shows the output generated.

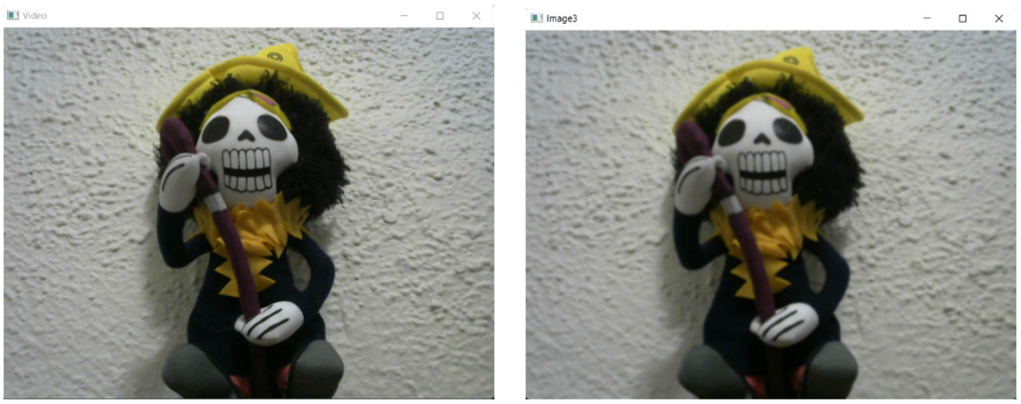

Fig.3 Transformation after applying Gaussian filter.

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

In task 6, Sobel X and Sobel Y filter accesses each pixel and is convoluted with the 1x3 filter. On the press of the keys ***'x'*** and ***'y'***, the output images are displayed under the named windows ***"Image4"*** and ***"Image5"***. Fig.4 & 5 shows the result of convolution by 1x3 separable filters.

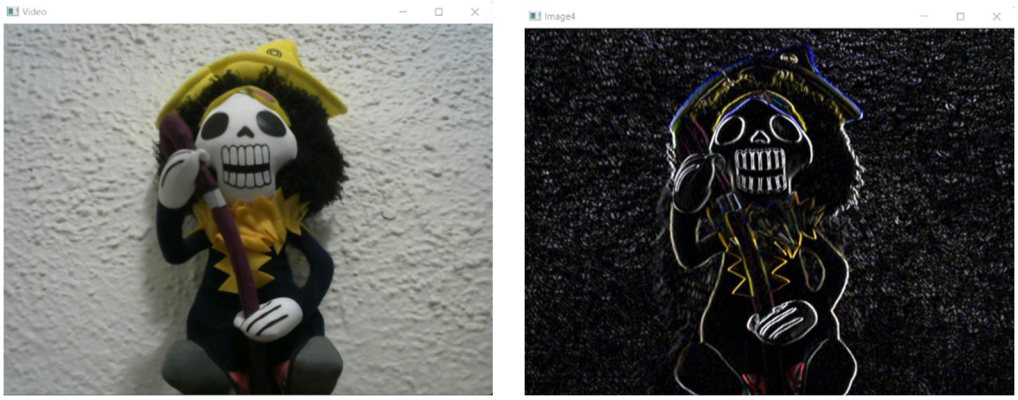

Fig.4 After applying the Sobel X filter on the video channel frame.

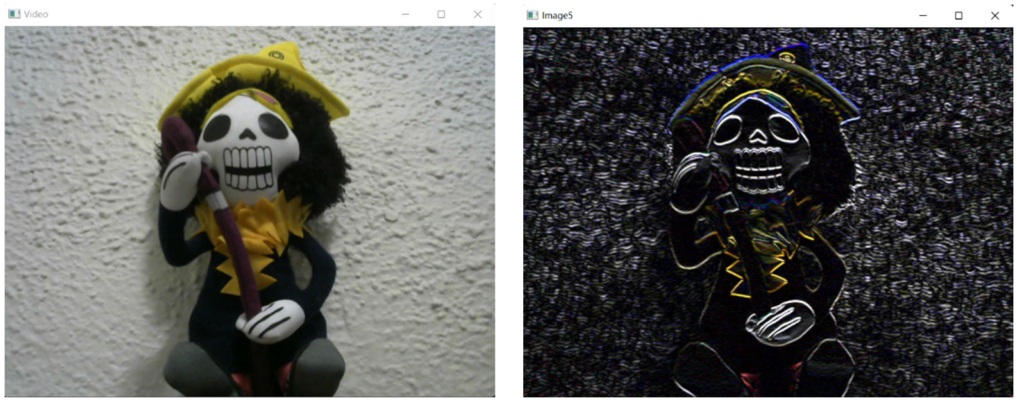

Fig.5 After applying the Sobel Y filter on the video channel frame.

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

In task 7, the function gradient magnitude uses Sobel X and Sobel Y functions to generate Sobel X & Y images. These images are used to generate gradient magnitude images and the ***Euclidean distance*** formula is used to calculate the distance. The filter is called on the key press of ***'m'***. Fig.6 illustrates the result.

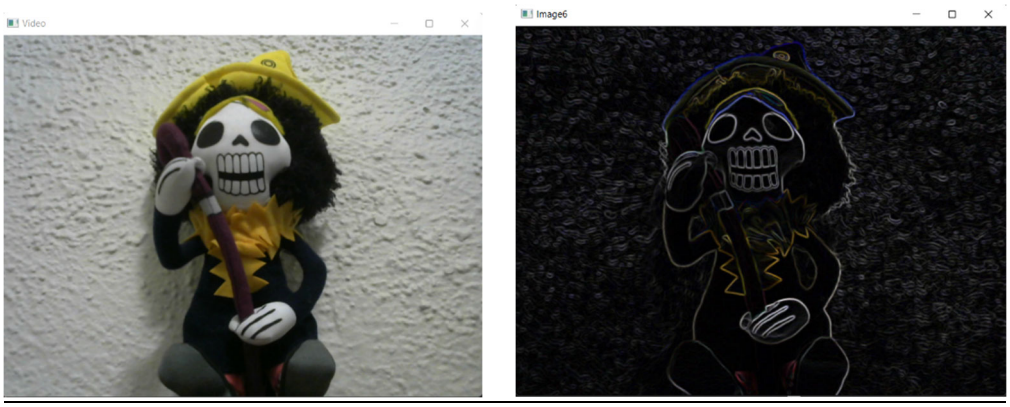

Fig.6 Gradient Magnitude Image.

Task 8: Implement a function that blurs and quantizes a color image

In task 8, the image is first blurred using the gaussian filter that was created in one of the previous tasks. Then, the blurred image is quantized using a level parameter, in this case, it is 15. The functionality can be accessed on the keypress of the letter ***'i'***. Fig.7 shows the blurred and quantized image.

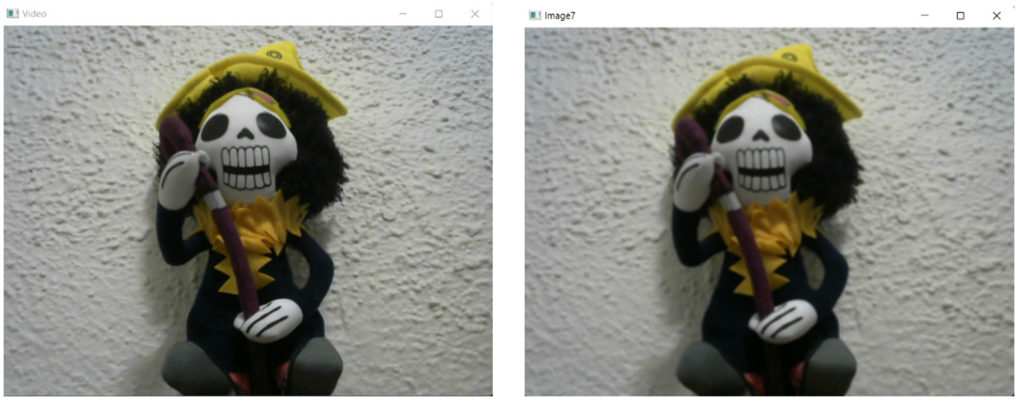

Fig.7 Blurred and Quantized image.

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

In task 9, the captured frame is passed through several functions that were created in the previous tasks, first, we calculate the gradient magnitude, then we blur and quantize the image. A threshold of 15 is set so that the pixel that is beyond the gradient magnitude is set to black. The letter 'z' key press calls this function. Fig.8 illustrates the cartoonized image.

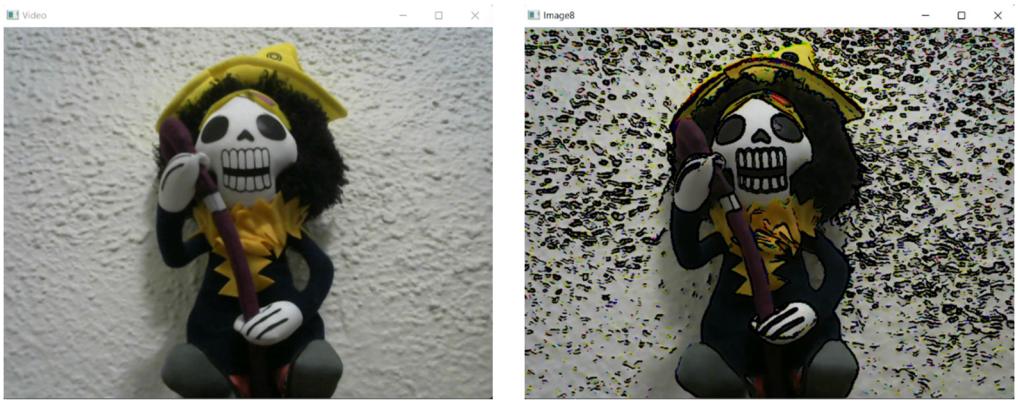

Fig.8 Cartoonized Image.

Task 10: Allow the user to adjust the brightness of the image

In task 10, this function allows the user to enter an input, which will increase or decrease the brightness of the image. If the user enters a value that is not within the range, then it shows a prompt.Fig.9 & 10 show the transformed image.

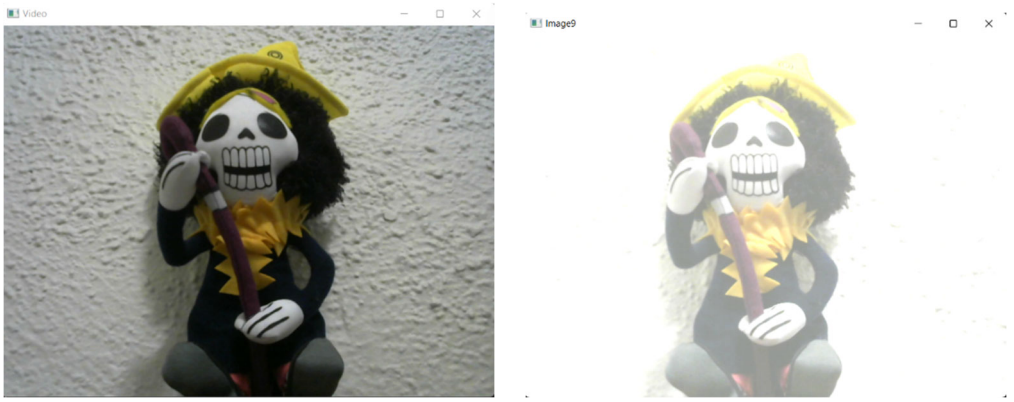

Fig.9 Brightened Image.

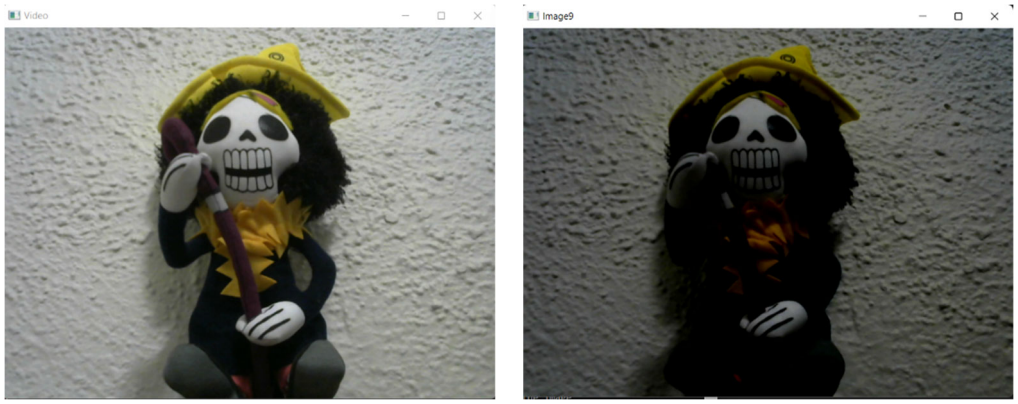

Fig.10 Reduced brightness image.

Extension 1: Inverting the color intensity of the image

In extension 1, this function accesses the pixel individually, the three channels of each pixel's values are subtracted off from 255. This filter can be accessed through the ***'v'*** key. The image is portrayed in ***"Ext.Image10"*** window. Fig. 11 illustrates the negative version of the image.

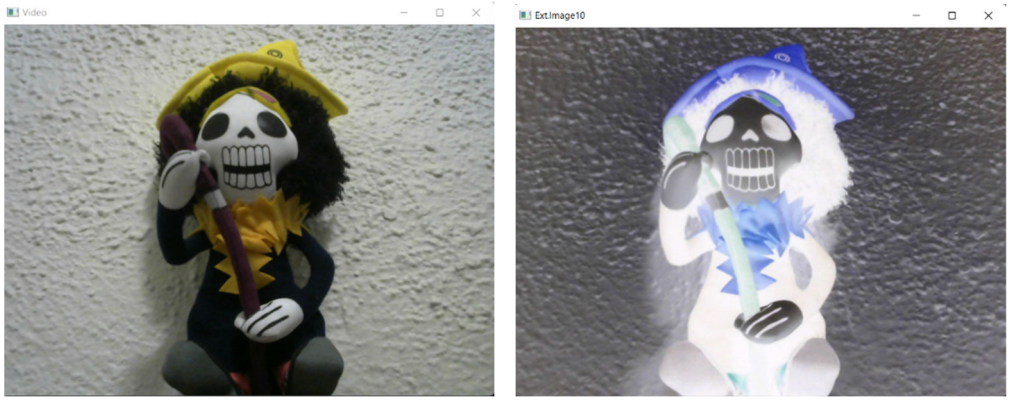

Fig.11 Negative version of the captured frame.

Extension 2: Edge Detection of the object

In extension 2, OpenCV in build functions is used to illustrate this transformation. First, the input image is converted into a grayscale image and then Sobel X and Y filters are applied, then the magnitude gradient is applied and then the image is normalized. These built-in functions help to detect the edges of the object. This function can be called on the keypress of the letter ***'o'***. Fig.12 shows the difference between the original and the filter applied to the image.

Fig.12 Edge Detection of the object captured.

Extension 3: Color Reduction

In extension 3, this function reduces the color values of the image through saturation casting by providing a divisor that is of power 2 (this number is used for better results). The output image will have the same resolution and channel depth as the input image, but with the color reduced by rounding the values to the nearest multiple of the divisor. ***'f'*** letter calls in this function. Fig.13 shows the output of this color reduction function.

Fig.13 Illustration of color-reduced image.

Acknowledgment of resources or people consulted for this project:

- OpenCV Tutorials: https://docs.opencv.org/4.5.1/index.html
- Brightness Filter: https://www.opencv-srf.com/2018/02/change-brightness-of-images-and-videos.html#:~:text=If%20you%20want%20to%20increase,every%20pixel%20in%20the%20image.&text=If%20you%20want%20to%20decrease,every%20pixel%20in%20the%20image.
- Sobel filter: https://fiveko.com/sobel-filter/