Project 1: Real-time Filtering

Hardik Devrangadi devrangadi.h@northeastern.edu

Real Time Filtering – Image Manipulation using OpenCV with C++

This project aims to perform real-time filtering on images through the webcam of the laptop. In this project, several tasks such as converting images to greyscale, adding gaussian blur, cartoonizing the images using gradient magnitude and blur/quantize filters, adding salt and pepper noise is performed. In this project, the use of convolution to manipulate an image is highly enforced, which makes the understanding of image manipulation much more solid.
For the tasks, the live video is captured from the webcam as a series of frames, and these frames are processed to obtain filtered or manipulated images. Apart from applying filters by accessing individual pixels, built-in OpenCV functions are also used, which helps to familiarize with the library and its capabilities.

Tasks

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

In this task, a simple code was written in the file "imgDisplay.cpp" which reads an image file whose path is passed as a command-line argument during runtime. There are two functionalities added:

- When the key 's' is pressed, the image is loaded with $1/8^{th}$ the original size in color.
- When the key 'q' is pressed, all windows are closed and the program ends.

The demo for this task is displayed here.

Task 2: Display live video from the webcam

In Task 2, a new file "vidDisplay.cpp" is created, which creates a video channel, creates a window, and displays the live video from the webcam. The same functionalities as Task 1 is implemented. When the key 's' is pressed, the image is saved. When the key 'q' is pressed, the program quits.
The demo for the task is displayed here.

Task 3: Display greyscale live video from the webcam

In Task 3, the live video is obtained from the webcam and each frame is processed to a greyscale, using the OpenCV cvtColor function which uses the *cv::COLOR_BGR2GRAY* transformation.

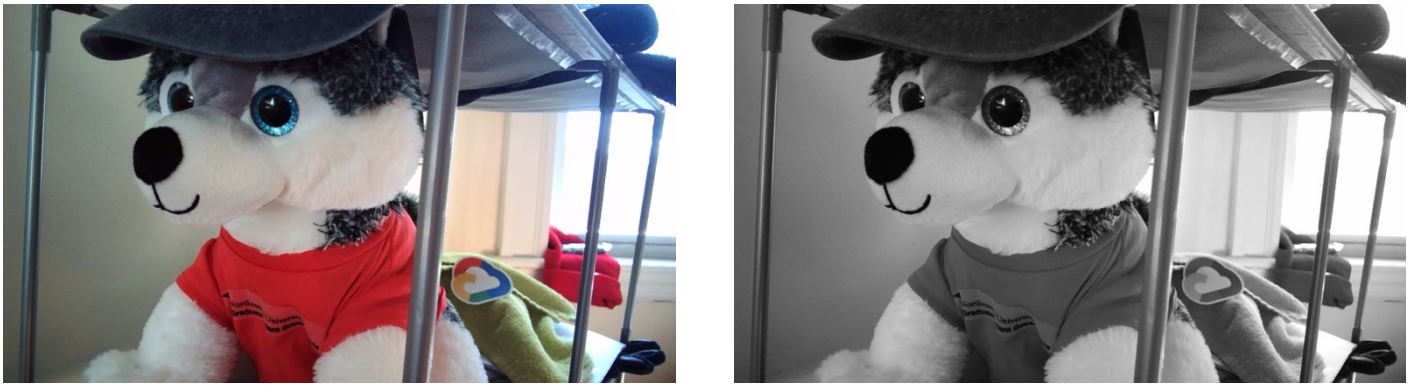

Figure 1

This is done by allocating each color channel (Y) to be allocated as follows:

$$Y \leftarrow 0.299R + 0.587G + 0.114B$$

Figure 1 shows the original and cvtColor version of the greyscale image. This filter is triggered on the press of the 'g' key.

Task 4: Display alternative greyscale live video from the webcam

In Task 4, the live video is obtained from the webcam and each frame is processed to a greyscale by individually accessing the pixels. This is done by applying the average of RGB to dst. with each color channel (Y) as follows:

$$Y \leftarrow \frac{R + G + B}{3}$$

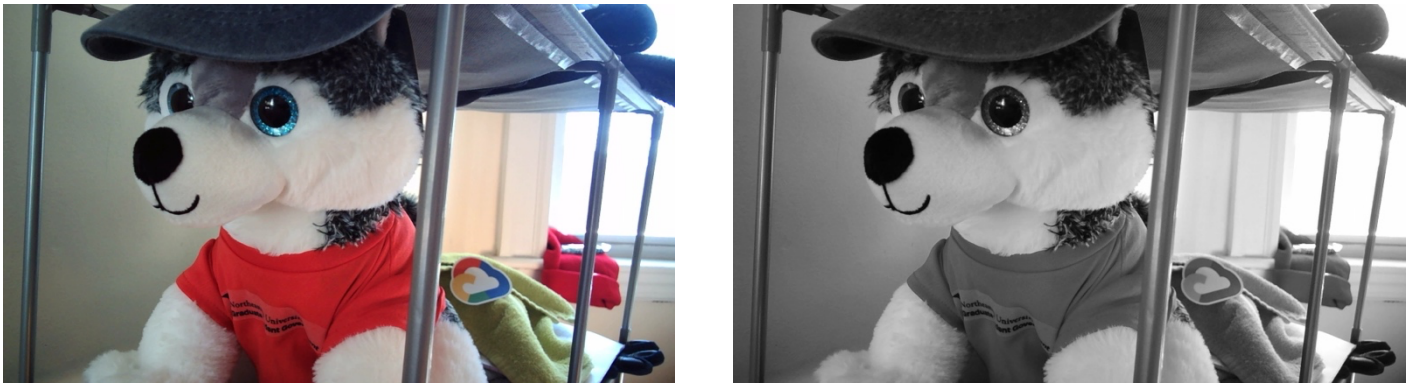

Figure 2

Figure 2 shows the original and greyscale image obtained by the above formula. This filter is triggered on the press of the 'h' key.

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

In Task 5, the gaussian filter is applied by creating two separable filters, 1 x N and N x 1. The pixels off the edges while applying the 5x5 filter are handled by padding the boundaries with reflections over that edge i.e., padding by mirroring.

Figure 3

Figure 3 shows the original and the blurred image obtained by using two separable filters, by accessing the pixels individually. This filter is triggered on the press of the 'b' key.

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

In Task 6, this function processes the image and applies the filters by accessing pixels directly and applies the Sobel X and Sobel Y filters using separable 1x3 filters. This uses the created 1xN and Nx1 filters that were created for Task 5.


Figure 4


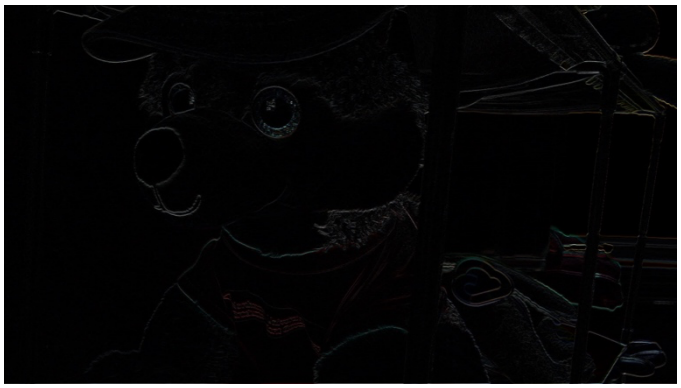

Figure 5

In Figure 4, the original and processed image with 3x3SobelX is displayed. This filter is triggered on the press of the ‘x’ key.
In Figure 5, the original and processed image with 3x3SobelY is displayed. This filter is triggered on the press of the ‘y’ key.

Task 7: Generate Gradient Magnitude image from the Sobel X and Sobel Y Filter images

In Task 7, Sobel X and Sobel Y filters are applied to the image, and the gradient magnitude is calculated by calculating the Euclidean distance for the magnitude, which is sqrt(sx^2 + sy^2). This magnitude image will display all the edges of the image.


Figure 6

In Figure 6, the original and processed image with the gradient magnitude image is displayed. This filter is triggered on the press of the ‘m’ key.

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

In Task 8, The image is first blurred with the function created in Task 5, and then is divided into several buckets. For the result shown, 10 buckets are used which means the level parameter is 10.

Figure 7

In Figure 7, the original and processed image with blur and quantization applied is shown. This filter is triggered on the press of the 'l' key.

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

In Task 9, a function is written to cartoonize an image. This is done by first calculating the gradient magnitude, and then the image is blurred and quantized. A threshold is set so that any pixels with the gradient magnitude greater than that threshold is set to black. In this example, the threshold is set to 15.

Figure 8

In Figure 8, the original and cartoonized image is displayed. This filter is triggered on the press of the 'c' key.

Task 10: Implement a filter to add salt and pepper noise to Color and Greyscale images

In Task 10, This function takes in a color image and a noise ratio and adds salt and pepper noise to the image. The noise ratio specifies the fraction of the pixels that should be affected by the noise. For each pixel, a random number is generated, and if the value is less than noise ratio / 2, salt noise is added by setting the value to 255. If the value is greater than 1 – noise ratio / 2, pepper noise is added by setting the value to 0.

The function uses the rand function from the cstdlib library to generate random numbers, and time from the ctime library to seed the random number generator with the current time. This means that each frame has unique noise, with the salt and pepper pixels varying each time. This is done for both the color image and the greyscale image.

Figure 9

Figure 10

In Figure 9, the original and image with salt and pepper noise for a colored image is shown.
In Figure 10, the original and image with salt and pepper noise for a greyscale image is shown. Both processed images are generated when the ‘p’ key is pressed.

Extensions

These are the extensions; other implementations of some functionalities, that have been added to this project.

Extension 1: Adding text to image (Meme Generator)

In Extension 1, functionality has been added to add text to the image, which is input to the console via the getline() function. The text obtained from the console is printed on the screen starting from the center of the image. These images are then saved to the hardcoded path.

Figure 11

In Figure 11, The original and the processed image where the text is obtained from the console is shown. This functionality is triggered on the press of the ‘d’ key.

The text can be added to an image which already has a filter applied, making it more “meme-worthy’. An example of this is shown in Figure 12.

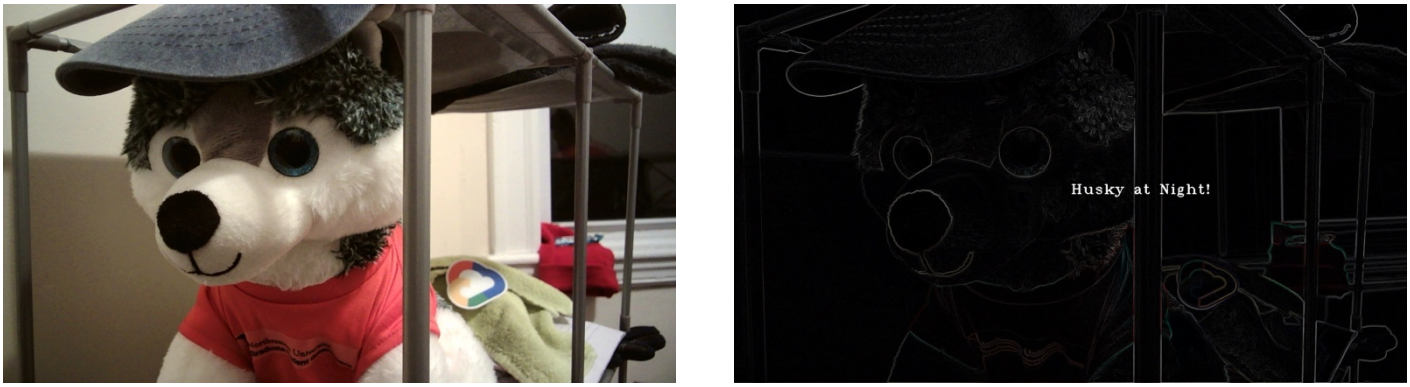

Figure 12

In the above example, text has been added on the image where the gradient magnitude filter is applied.

Extension 2: Inverting an image

In Extension 2, functionality has been added to invert the colored image. This is done by subtracting 255 – channel value for each of the channels to obtain the negative of the image.

Figure 13

In Figure 13, the original and inverted images are displayed. This filter can be triggered by pressing the ‘i’ key.

Extension 3: Creating a Pencil Sketch of an image

In Extension 3, the pencil sketch of an image was obtained in three steps, first by converting the image to greyscale. This greyscale image is then blurred using gaussian blur to remove sharp edges for better results. Finally, this blurred image is passed through a canny edge function which is a built-in OpenCV function. This helps to extract the edges.

The final post-processing step is to invert the image by subtracting it from 255. This gives us black lines on a white screen, simulating a pencil sketch. Two examples are showcased in Figure 14 and 15.

Figure 14

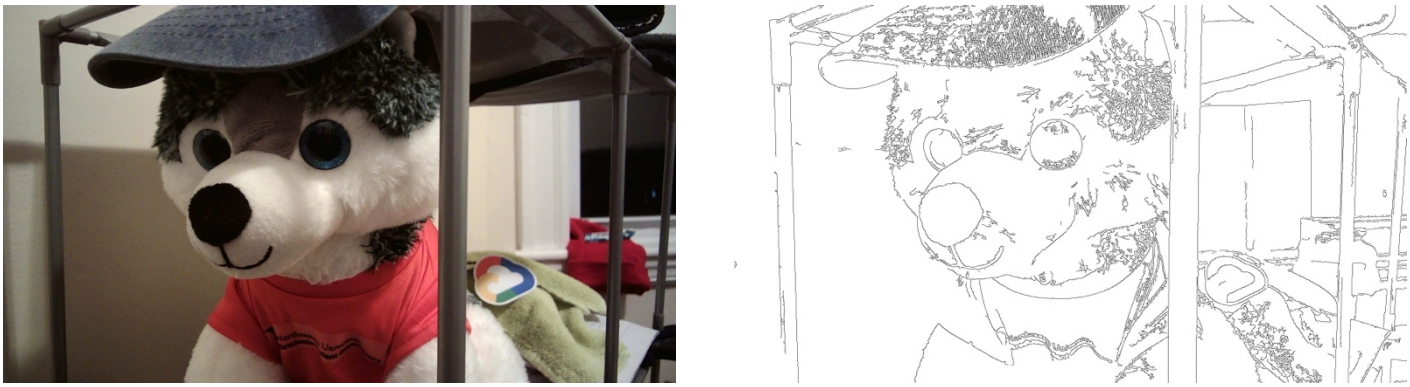

Figure 15

This filter can be triggered on the press of the ‘f’ key.

Project Learnings and Insights

Being a novice at OpenCV, this project gave a very good opportunity to work on image manipulation hands-on. Working on this project helped to reinforce the concepts taught during the lectures. The image manipulation techniques learnt in this project are very useful and provided a strong foundation in OpenCV.

One mathematical concept that I felt is the most important for image manipulation is convolution. Working on Task 5 and Task 6 was most challenging but very rewarding after seeing the filters come to life.

One of the challenges that was encountered through the project was matching the image type of the matrices created, which led to many debugging sessions. However, the overall project was very informative, fun, and provided in-depth understanding of how images work.

Acknowledgements and Resources used for this project

- OpenCV Tutorials: https://docs.opencv.org/4.5.1/index.html
- Gaussian Blur: https://datacarpentry.org/image-processing/06-blurring/
- Pencil Sketch: https://towardsdatascience.com/generate-pencil-sketch-from-photo-in-python-7c56802d8acb
- Images used in Pencil sketch examples: Manish and Pramod (Roommates)