# Introduction –

The project – Real time filtering focuses on implementing filters (built in and user-defined) on frames of video-capture. The project is divided into 10 tasks for understanding OpenCV.

## Task I –

Reading an image and displaying it using imread function. The code should also close the image window using ‘q’ key. The following screenshot is attached for reference ->

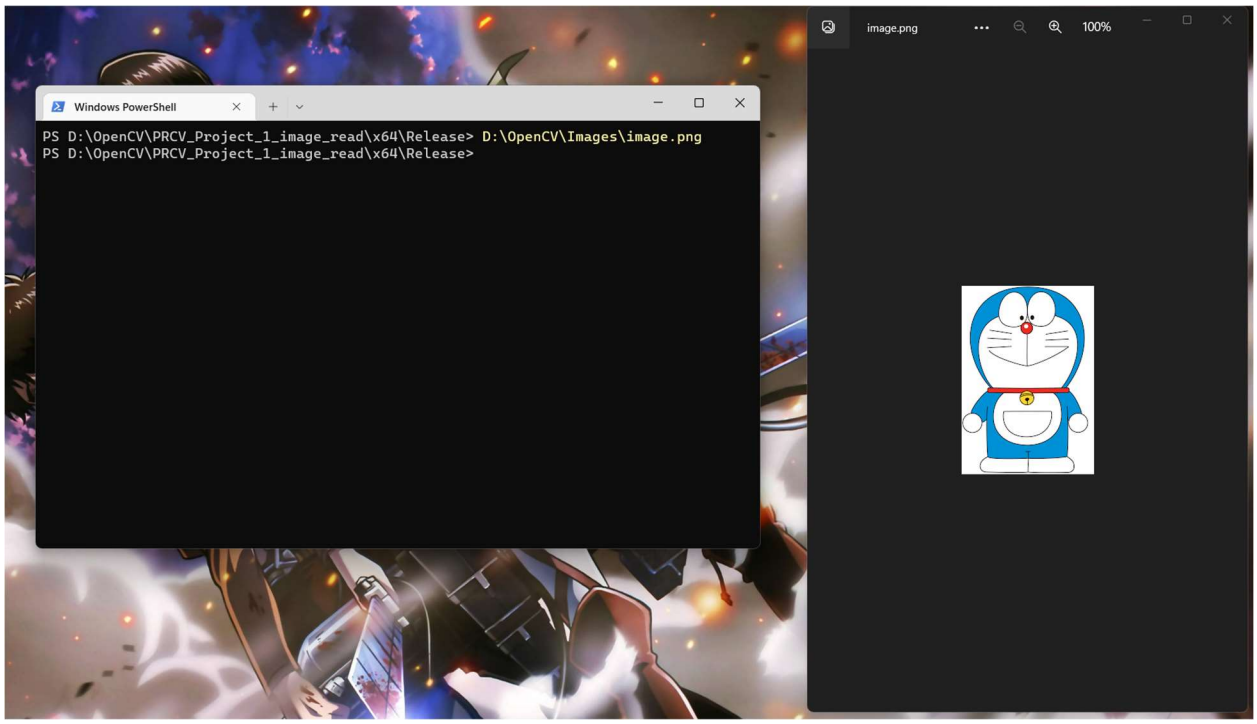


Fig. 0 – Image opening and quiting demonstration.

No extensions are added in this task.

## Task II –

In this task, we capture live video using webcam. Added functionality is ‘s’ keystroke for saving image. ***Mini-extension – 1*** -> Saved name changes per filter // Also user can take as many screenshots as required without overwriting same image names.

## Task III –

We use OpenCV grayscale feature using the function cv::COLOR_RGB2GRAY. This converts our RGB color image to a grayscale format with intensity values of

GS = 0.299·R+0.587·G+0.114·B where R, G and B are the intensity values of red, green and blue colors respectively and GS is the resultant intensity in grayscale all values as 8 bit ‘uchar’ hence range [0-255]. This is most likely because humans have greater sensitivity to green color as opposed to red and blue. PFA the screenshots for the same below in Fig.1.

Fig. 1- OpenCV Inbuilt grayscale feature(right) comparison with RGB (left)

## Task IV –

For this task of using alternative greyscale I used average value of RGB intensity values as the final intensity value for greyscale.

Fig. 2- OpenCV Inbuilt grayscale feature(left) comparison with custom greyscale filter(right)

From Fig. 2. We can infer that though there is not a major difference in the comparison the 1st one is more appealing to eyes because of the majority in green scale intensity values coming in output greyscale.

## Task V –

Gaussian blur implementation using separable matrices – [1 2 4 2 1] and $[1\ 2\ 4\ 2\ 1]^T$. Filtering is applied to copy image so that the edges are covered.

***Major-Extension – 2*** ->

In this extension the user can input x, where x is the times the image is blurred with the gaussian matrices. User can input value between 1-5 otherwise the function will throw an error. This is done for sanity check and speed of the program.

Fig. 3- RGB image (left) comparison with Gaussian blurring filter(right)

The right image is gaussian filter implemented 3 times using my extension. The image can be further blurred to 5 times. Comparison shown below.

Fig. 4- 5x blurred (left) comparison with 1x blurred(right)

## Task VI –

Sobel filtering implemented as separable filters. PFA the X and Y sobel filtering on live stream below ->

I have used zeros matrices in output so that we don't have any irregularities in 1$^{st}$ and last rows and columns.

Fig. 5- Sobel x filter (left) comparison with Sobel Y filter (right)

## Task VII –

The magnitude of Sx and Sy is calculated for respective filters and displayed below in magnitude gradient image.

Fig. 6- Normal RGB (left) comparison with gradient magnitude of Sx and Sy (right)

## Task VIII –

The function implements blurring and quantifying the values in the bucket bin size the user inputs. For example, the following is the result of bin size 15.

Fig. 7- Normal RGB(left) comparison with blurred/quantified filter (right)

## Task IX –

"Cartoonizing" effect is done by implementing the blurring/quantifying and then blackening the pixel values wherever the intensity magnitude is higher than the user specification. Below is the example for blurring effect and magnitude 10.

Fig. 8- Normal RGB(left) comparison with cartoon filter (right)

## Task X –

The individual task implementation I did is live negative filtering of the video. Below is the screenshot reference for the same.

Fig. 9- normal RGB filter (left) comparison with Negative filter (right)

## EXTENSION 3 –

I have implemented a filtering system which detects human skin color using HSV format. It took a lot of trial and error to find the right range to accommodate fair ranges of skin colors. The rest pixels where HSV values don't match they are colored as black. This can be used a premature human detection filter. PFA the demonstration below.

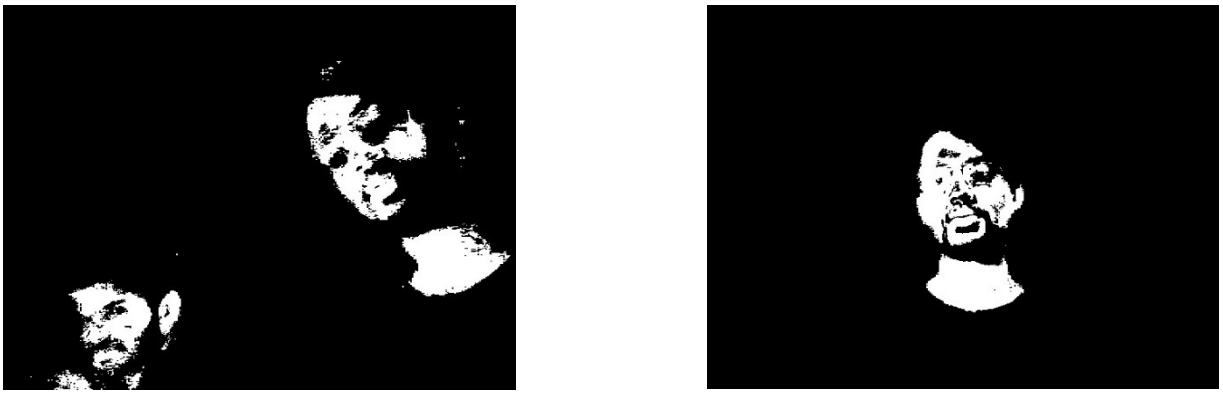

Fig. 10- Skin color detection with different skin colors.

## Reflection –

Never had this much fun doing assignments. While implementing these filters rather than using inbuilt functions directly, implementing them from scratch helped me understand how fast our system is :P. I'm confident that I can make innovative filters just by playing with the convolution matrices.

## Acknowledgement –

I have used Professor's lecture notes, tutorial and OpenCV documentation for assignment and Stackoverflow for resolving system issues.