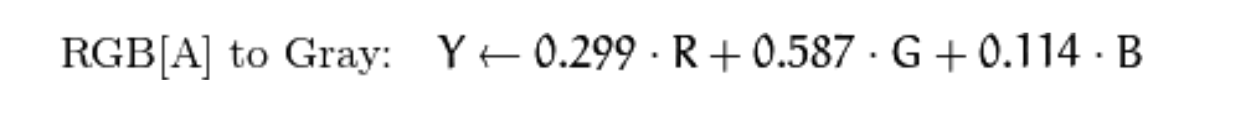
**Task 1:**

**Task 2:**

**Task 3:**

I store the last key selection in a mode variable (MODE\_COLOR, MODE\_GRAY, MODE\_BLUR, MODE\_FLIP, MODE\_INVERT). Each captured frame is processed according to the current mode so the effect persists across frames.

how each color channel is weighted in the conversion:



I checked [Wikipedia](https://en.wikipedia.org/wiki/Luma_(video)#Rec._601_luma_versus_Rec._709_luma_coefficients) for this equation and learned that this RGB-to-grayscale conversion is based on the Rec. 601 luma standard.

The green channel contributes the most to perceived brightness because the human eye is most sensitive to green wavelengths, while the blue channel contributes the least because human vision is less sensitive to blue light in terms of brightness perception.

The original and cvtColor version of the greyscale image:A person wearing glasses and a hoodie

Description automatically generatedA person wearing glasses and a hoodie

Description automatically generated

**Task 4:**

My customized greyscale image: A person wearing glasses and a hoodie

Description automatically generated

how I decided to generate my greyscale version:

Invert the red channel value: R->255-R, and use this value for each of the three channels so that it becomes a greyscale image.

the differences with the default greyscale image:

|  |  |  |
| --- | --- | --- |
|  | default | My custom |
| Number of channels | 1 | 3 |
| How the value of Luma component Y ′is calculated | 0.299R+0.587G+0.114B | 255-R |
| Whether it aligns with human brightness perception | Yes | No |

**Task 5:**

A person taking a selfie

Description automatically generated  
how I ensured using the original RGB values in the computation:  
I avoided modifying the source image in place, instead, I created a destination cv::Mat as a buffer, read the value from the source, do the computation and then write results to the destination. Then I made a deep copy of des: sepiaImg.copyTo(display);

Task 6:

The output of timing:

Time per image (1): 0.1741 seconds

Time per image (2): 0.0264 seconds

Terminating

blur5x5\_2 is faster because it uses a separable filter, use one horizontal filter and one vertical filter to replace the . The number of operations per pixel dropped from 25 to 10.

Another reason is in blur5x5\_2 I used row pointers Mat::ptr<> instead of Mat::at<>, this is more cache friendly.

A person wearing a green beanie and glasses

Description automatically generatedA person wearing a green beanie and glasses

Description automatically generated

Task 7: :

Task 8:

Sodel X:

A screenshot of a video game

Description automatically generated

Sodel Y:

A person wearing a hat and glasses

Description automatically generated

the gradient magnitude:

//TODO

Required Images 5: show the original, X-Sobel, Y-Sobel, and the gradient magnitude image in your report.

Task 9:

// TODO

Required Image 6: show the original and the blurred/quantized image in your report.