UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

SPECIAL EFFECTS ON COLOR IMAGES

Urcan Diana-Cristina,

Group 30236

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Content

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

1. Introduction

1.1. Topic

This project describes an implementation for interesting special effects on color images. Chosen effects are grayscale, red-eye detection and removal from image, and red reduction.

1.2. Main problems

For the grayscale part, the implementation should be pretty straight-forward, the main concern was choosing the right formula.

Red-eye detection included detecting eyes from an image and determine if there is any red to be removed. The hardest part was to fit and extract red pixels only from the pupil. Not having an algorithm that was verified on multiple images – neural train – made everything even more difficult, as for the red shade and eye differ from person to person.

Red reduction needed a method for modifying the Red channel by a value – given or chosen.

1.3. Target

Finding the most accurate formula for grayscale, red eye remove only from the pupil part of the eye – without having an algorithm trained on multiple possible input images, saturation by dynamically choosing the value of Red.

2. Theoretical foundation

2.1. Grayscale

There are many grayscale conversion algorithms, their principal steps are:

1. Get the red, green, and blue values of each pixel from image.
2. Use a math formula to combine those three values in order to perfectly create a shade of gray.
3. Replace the original red, green, and blue values with the new gray value into the result image.

"Averaging" method.

The simplest method is by averaging values from the pixel:

$$\frac{Red + Green + Blue}{3}$$

This formula generates a reasonably nice grayscale equivalent, and its simplicity makes it easy to implement and optimize. But it does a poor job of representing shades of gray relative to the way humans perceive luminosity (brightness). For that, we need something a bit more complex.

"Luma" method.

Because humans do not perceive all colors equally, the "average method" of grayscale conversion is inaccurate. Instead of treating red, green, and blue light equally, a good grayscale conversion will weight each color based on how the human eye perceives it.

A common formula in image processors is:

$$Gray = (Red * 0.3 + Green * 0.59 + Blue * 0.11)$$

This formula requires a bit of extra computation, but it results in a more dynamic grayscale image.

The original International Telecommunication Union (ITU) recommendation (BT.709, specifically) is the historical precedent.

This formula, sometimes called Luma, looks like this:

$$Gray = (Red * 0.2126 + Green * 0.7152 + Blue * 0.0722)$$

Some modern digital image and video formats use a different recommendation (BT.601), which calls for slightly different coefficients:

$$Gray = (Red * 0.299 + Green * 0.587 + Blue * 0.114)$$

"Desaturation"

Desaturating an image works by converting an RGB triplet to an HSL triplet, then forcing the saturation to zero. This takes a color and converts it to its *least-saturated variant*. A pixel can be desaturated by finding the midpoint between the maximum of (R, G, B) and the minimum of (R, G, B), like so:

$$Gray = \frac{(Max(Red, Green, Blue) + Min(Red, Green, Blue))}{2}$$

"Single color channel"

The fastest computational method for grayscale reduction - using data from a single color channel. This method requires no calculus. All it does is pick a single channel and make that the grayscale value, as in:

$$Gray = Red,$$

where Gray can be Green or Blue.

But this image may have dark shades of gray as for it reduces image luminosity. Also, it has results that may vary if the saturation of the image is different from one testing image to another.

2.2. Red-eye detection and removal from digital images

The algorithm for eyes detection that uses geometrical information for determining in the whole image the region candidate to contain an eye, and then the symmetry for selecting the couple of eyes.

Works on complex images without constraints on the background, skin color segmentation and so on.

OpenCV provides a training method Cascade Classifier Training or pretrained models, that can be read using the **CascadeClassifier::load** method.

Haar classifiers**,** classifiers that were used in the first real-time face detector. A Haar classifier, or a Haar cascade classifier, is a machine learning object detection program that identifies objects in an image and video.

After identifying the face and eyes, color needed to be compared to different values of red. Many types of masks could be computed – RGB comparing matrices, HSV or HSL. Images have different luminosities and saturation, so there is a very wide range for red shades.

Possible solutions:

Direct removal – having images only front facing could led to a small change of not really having red everywhere. By direct elimination, we could detect face and then search for red, but face may have different (lighter) shades of red – so removing red without making sure we are on the eye is inaccurate and is not a good idea for the result.

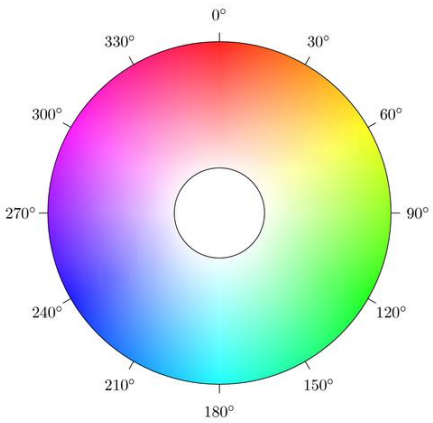
RGB

For red removal, we can use RGB - RGB color space constructs all the colors from the combination of the Red, Green and Blue colors.

The red, green and blue use 8 bits each, which have integer values from 0 to 255. This makes 256*256*256=16777216 possible colors. Therefore, we have multiple possible red combinations from Red, Green and Blue, so this is not a very good idea, because we cannot extract ranges from this values, so we have to determine and write each shade of red.

HSV

HSV (for hue, saturation, value) are alternative representations of the RGB color model to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top.



HSL

HSL Value. Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue. Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color. This is also a good way to detect ranges of red.

After detecting red ranges, a mask can be made with values of white or black, white is detection of red (any shade from defined range) and black is absence of a visible shade of red. This mask can be used after we found the eyes and we surrounded the pupil of each eye, by changing the color of the pupil where the mask is white.

2.3. Red Reduction

HSV

As for red reduction, HSV can be used in order to reduce the saturation (red reduction is similar with changing image saturation), so by converting input image from RGB to HSV, we create a channel of Saturation – if we modify this channel, then the whole image saturation is decreased / increased.

RGB

Reducing each value from each channel of Red in RGB is a direct way of reducing red, so we can modify each channel with a given value (or chosen value) from the user.

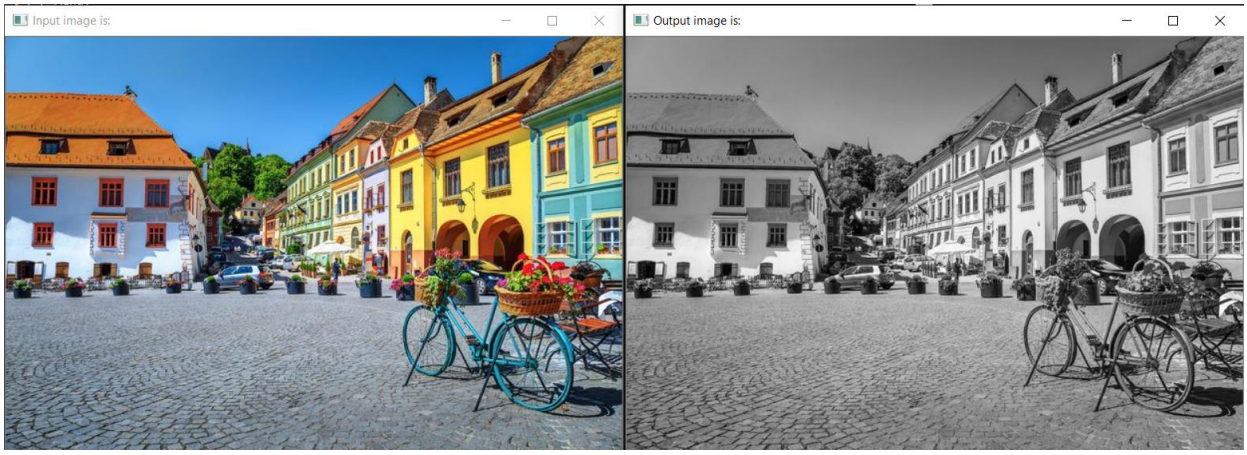
3. Implementation

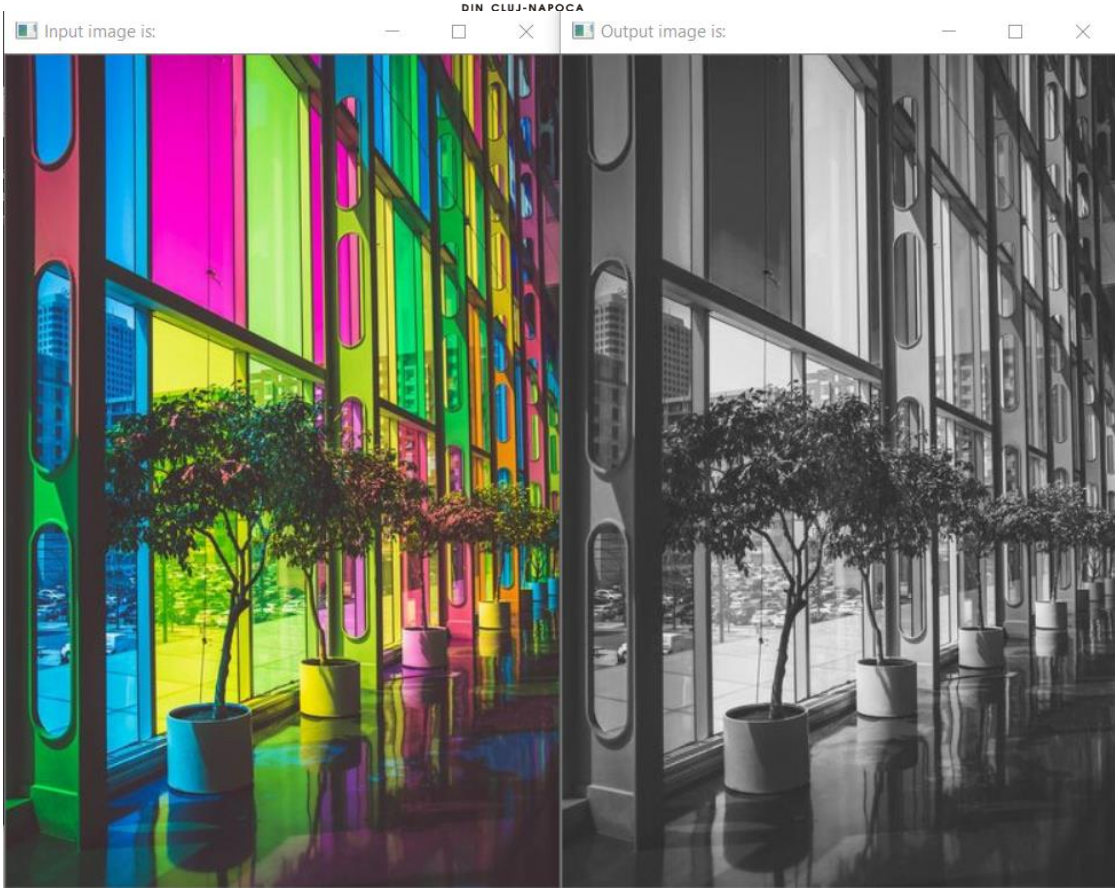
3.1. Grayscale

The chosen implementation is the Luma method with the formula explained in the theoretical part. Even though this is a more complex calculation that the average method, it is a standard and more accurate way of creating grayscale images.

$$Gray = (Red * 0.2126 + Green * 0.7152 + Blue * 0.0722)$$

Results

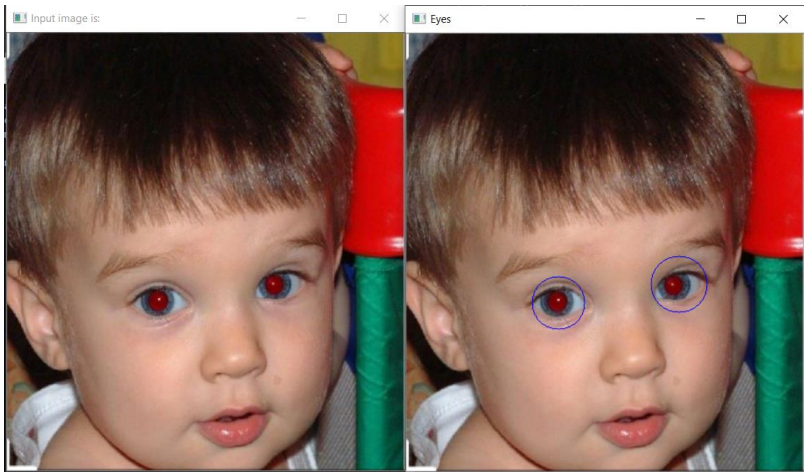


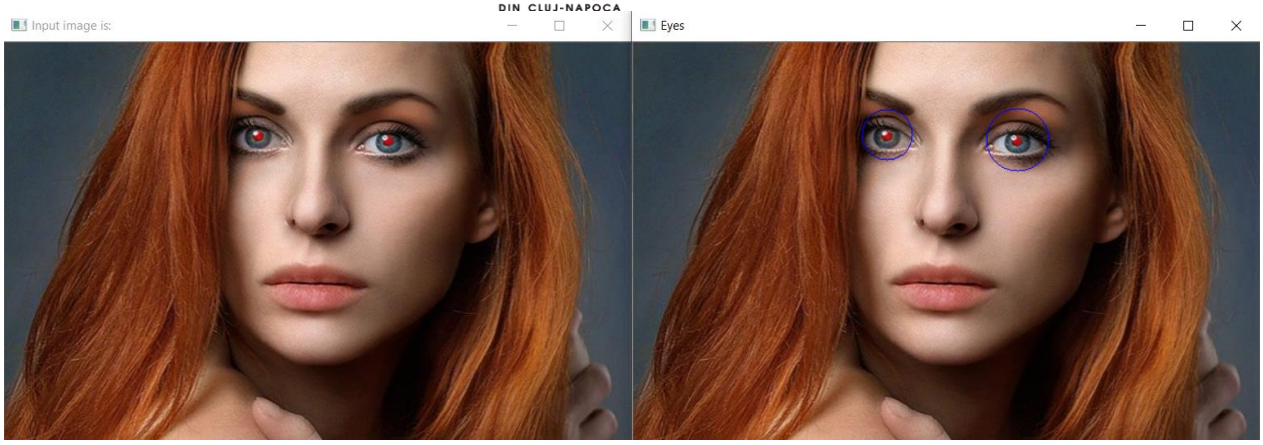


3.2. Red-eye detection and removal from digital images

First, we detect the face using **haarcascade_frontalface_alt.xml** and then we approximate the eye part by **haarcascade_eye_tree_eyeglasses.xml,** from where we assume that eyes are at 0.2 distance from face's width with the eye's x value, so we try and calculate the center of the eye and the radius to try and find then the pupil.

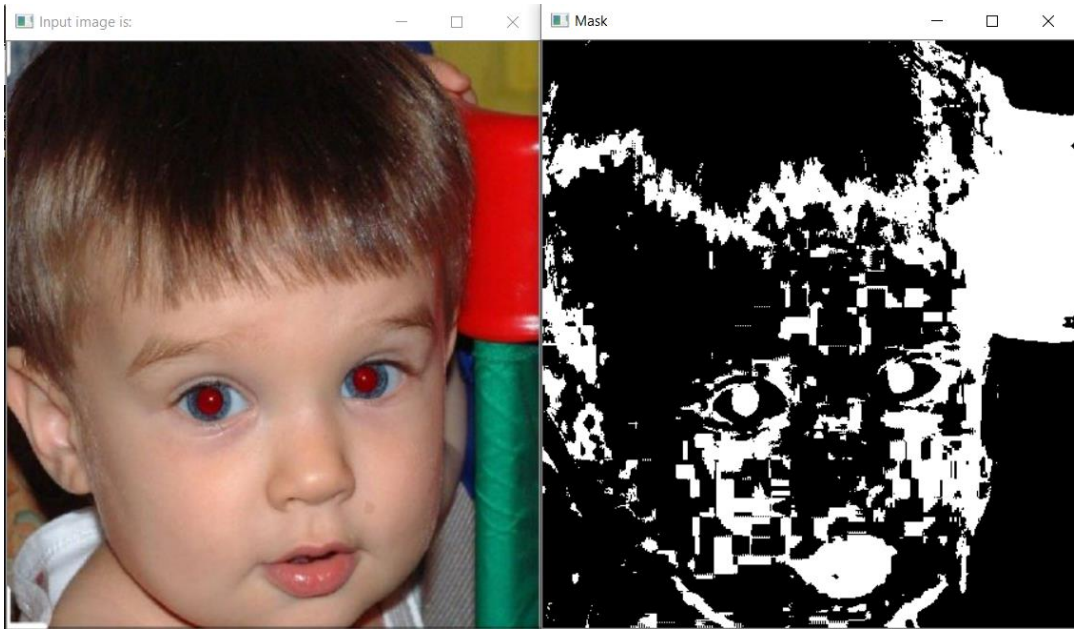
Eye detection

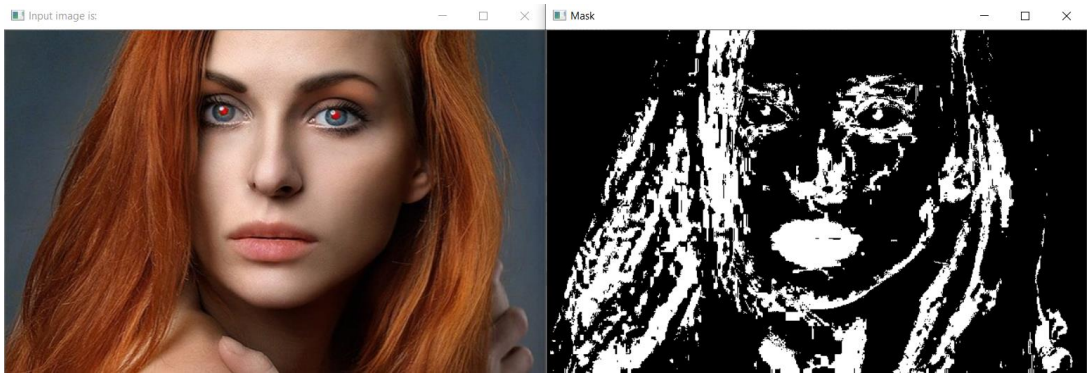




After eye detection, I made a mask from HSV image of the input RGB image – with ranges of 0-8 and 165-180 for Hue (Red value), 70-255 for the value, 40-255 for the saturation.

Masks





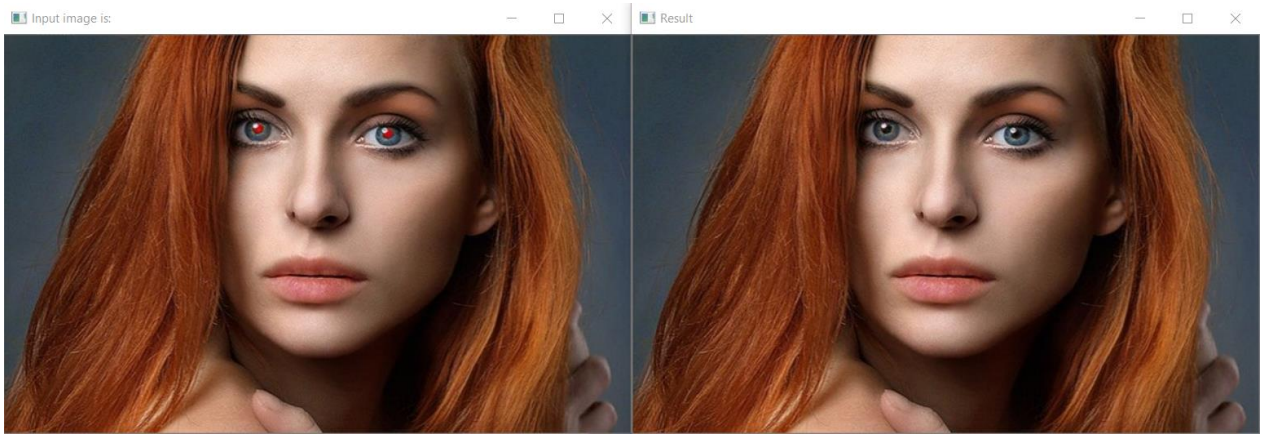
From the masks we can clearly see that if we do not select only the eye part, then we may have many inaccurate eliminations of red – mouth, parts of the environment.

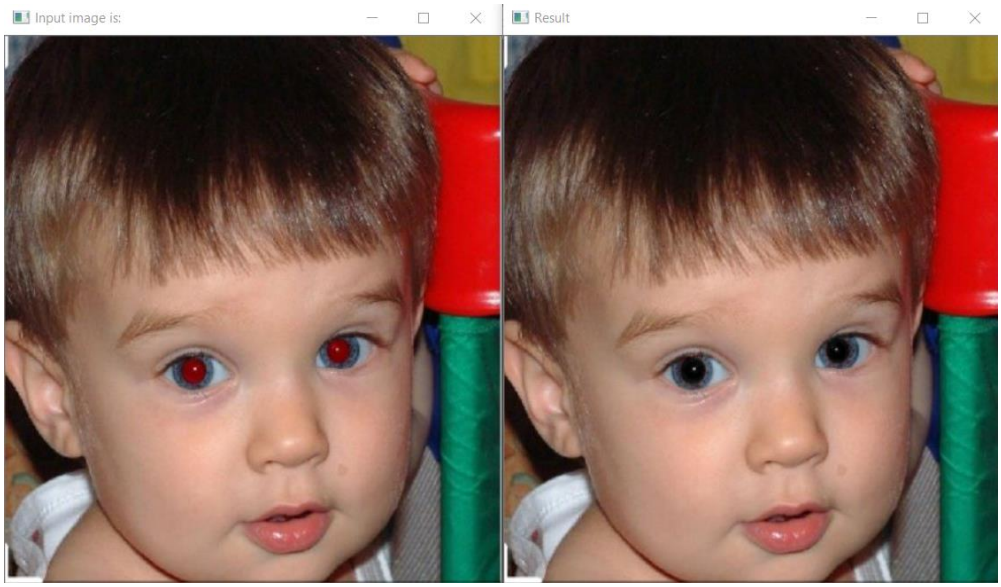
So, after creating the mask, I searched for the eye with its radius and center and in mask I took the middle of the eye image and searched for the first pixel of white in the right part – to find the pupil – I search until a find a reasonable diameter for the eye. Then, I remove the red pixels by iterating in circle the pupil, starting from the center and going as far as the radius (half of the calculated diameter) goes – in each direction.

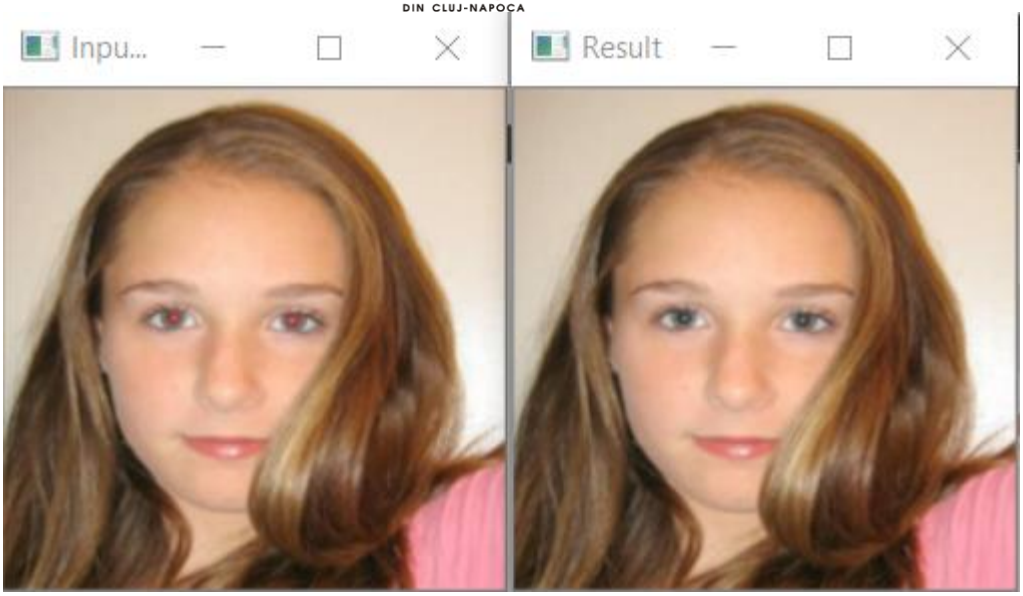
Accuracy

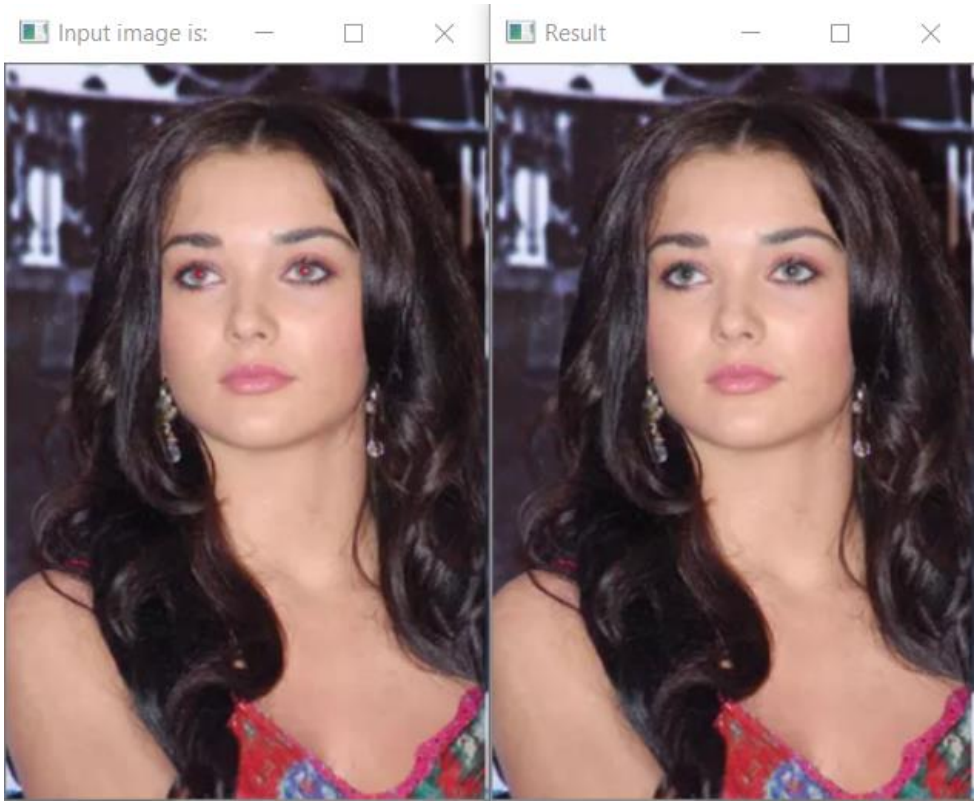
After doing the first circle of checking and removing, I calculated area of red pixels left – going on the whole eye again and after finishing I compared this area with the eye dimension – if the area was bigger than 1/5 of the eye, I supposed there were many inaccurate replaces, so I took a bigger circle and removed red again.

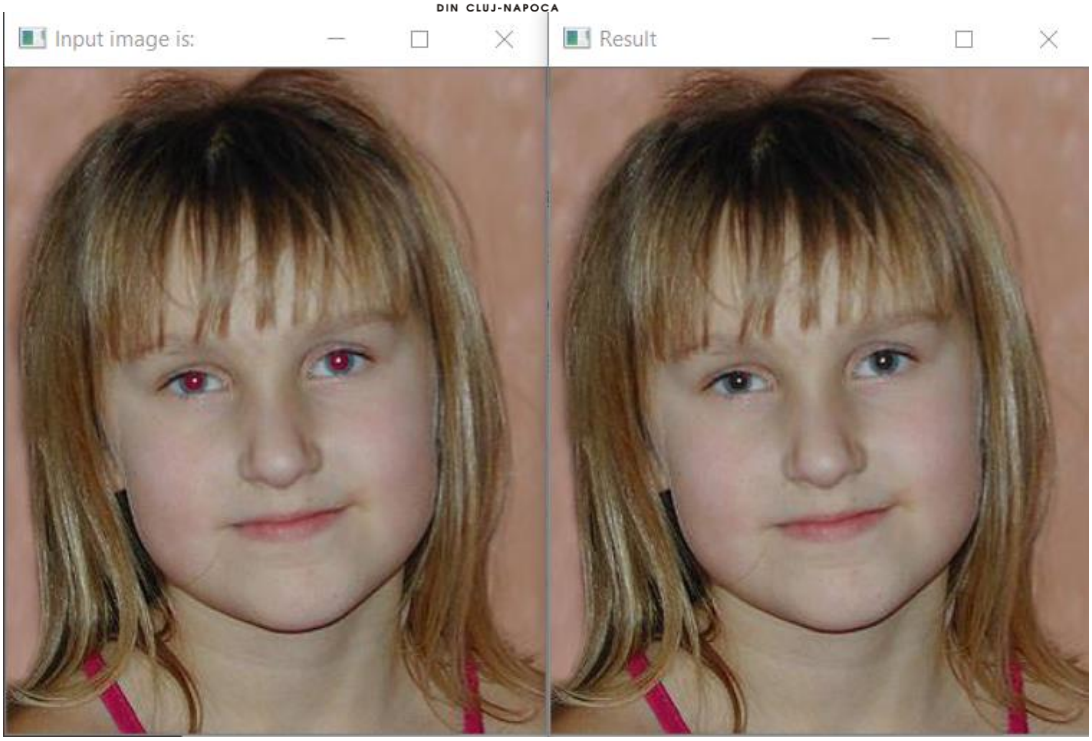
Results






Inpu...
Result


Input image is:
Result

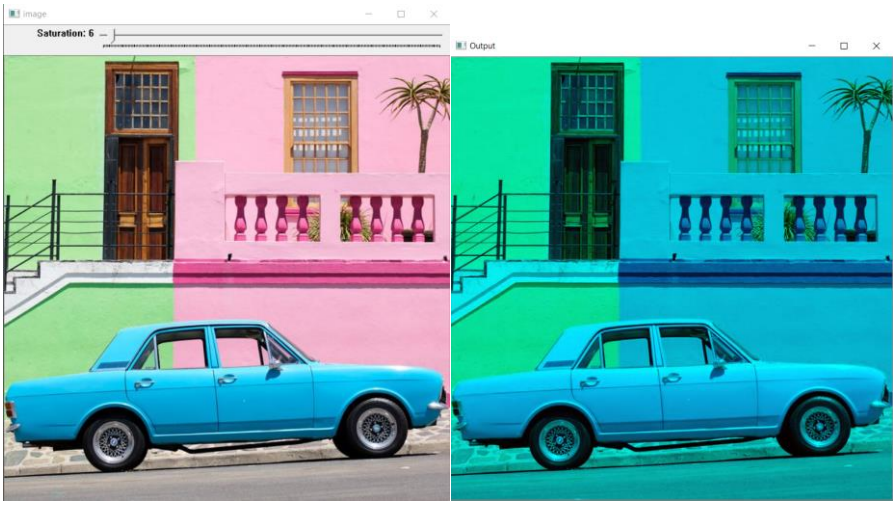


As we can see, red eyes are perfectly removed from the images, having different shades, luminosities, ranges, eye dimensions as input and besides maybe some pixels near the eye region, it perfectly removes red.

3.3. Red Reduction

For red reduction, I modified all Red channels from each pixel of the image with the chosen value from the user. The user has an interactive bar and he can choose the value of the red, by sliding the bar.

Results



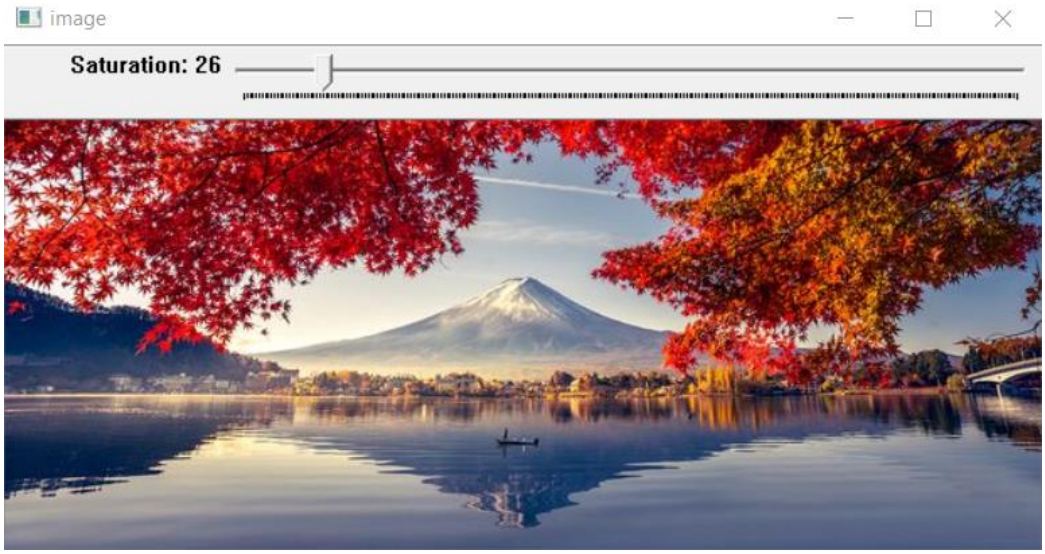
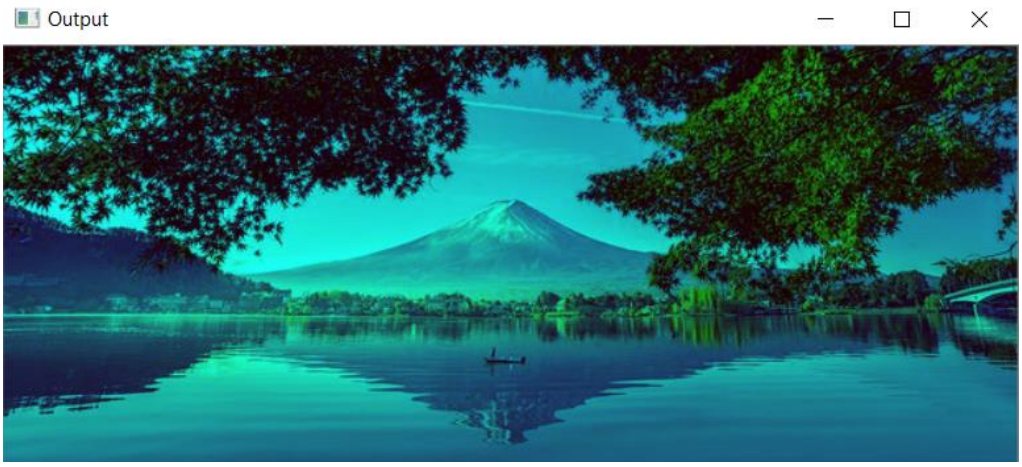


image
Saturation: 26


Output

As we can see from red reduction, the results have an intense red color or a very reduced red shade and the whole image saturation is modified, being more intense or more reduced.

4. Conclusions

All three image filters are implemented using one of the theoretical method, the results were good for each filter.

Problems

During implementation, there were some problems especially at red eyes, not having an algorithm for removing or detecting only the pupil, that was trained on more images, I struggled with finding the perfect range of RED in RGB and then selecting only the important region of the eye. Problems were given by the dimension and position of the eyes too.

Future development

This project is a needed way of editing images, so it would be a great idea to work with ML and trained algorithms in order to remove red eyes. The other two filters and fully implemented, another future development may be choosing another gray filter formula, depending on the wanting result.

5. Bibliography

https://tannerhelland.com/2011/10/01/grayscale-image-algorithm-vb6.html

https://users.utcluj.ro/~tmarita/HCI/L9/L9.pdf

https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

https://people.ucalgary.ca/~ranga/enel697/ColorImageProcessingIntro.pdf