

Rodrigo Ronconi Richter, 217445






Fundamentals of Image Processing, Prof. Manuel M. Oliveira



Implementation Assignment #1

Part I - Reading and Writing Image Files

A simple python script with OpenCV was made to open and save jpeg images, with a *_new* extension tag. The results of the images processed by the program are:

Image Name	Image File	Size
Gramado_22k		21,7kB
Gramado_22k_new		41,1kB
Gramado_72k		73,4kB

Gramado_72k_new		50,7kB
Space_46k		46,2kB
Space_46k_new		88,3kB
Space_187k		191,3kB
Space_187k_new		108,8kB

Underwater_53k		53,6kB
Underwater_53k_new		100,8kB

After being processed by the script, none of the images had any visible differences on quality. What did change was the file sizes. At first, the changes in size appear to be quite random, but it is not the case. JPEG is very efficient at making the trade-off between quality and size. Between a scale from 0 to 12, JPEG compresses a file more or less, making the range of the possible file size vary drastically. Its size also varies according to the degree of smoothness of the image. For example, JPEG is very bad at doing logos and images with rough edges, ending up with a big file and low quality. For that, formats like PNG are more recommended.

Bringing this explanation to our test images, we can, for example, compare the images *Gramado_22k* and *Gramado_72k*. The *Gramado_22k* doubles its size, going to ~42kB, while the *Gramado_72k* actually gets its size reduced to ~50kB. The files had a difference in size of 50kB, and now are only ~10kB apart. This happens because the two images were originally compressed with a different compression degree, and the python script compresses them again with the same degree. This same result can be observed with *Space_46k* and *Space_187k*.

But what's the matter with the last one, *Underwater_53k*? Why it doubles in size? The same thing happens on the other images, and it's because there are more factors to it. First, there is additional metadata being added on compression. The process of compression also generates image artifacts, which are pixels that "pop up" and are initially not very visible. The thing is, compression is less effective when an image already contains artifacts and the images tested all contain a balance between smooth and rough edges, therefore image artifacts are certain to appear. The higher the quality of the compression, the more effective another subsequent compression will be. This can be seen in the fact that the images with low file sizes (higher compression degree) end up getting bigger, and the images with large file sizes (lower compression degree) get smaller.

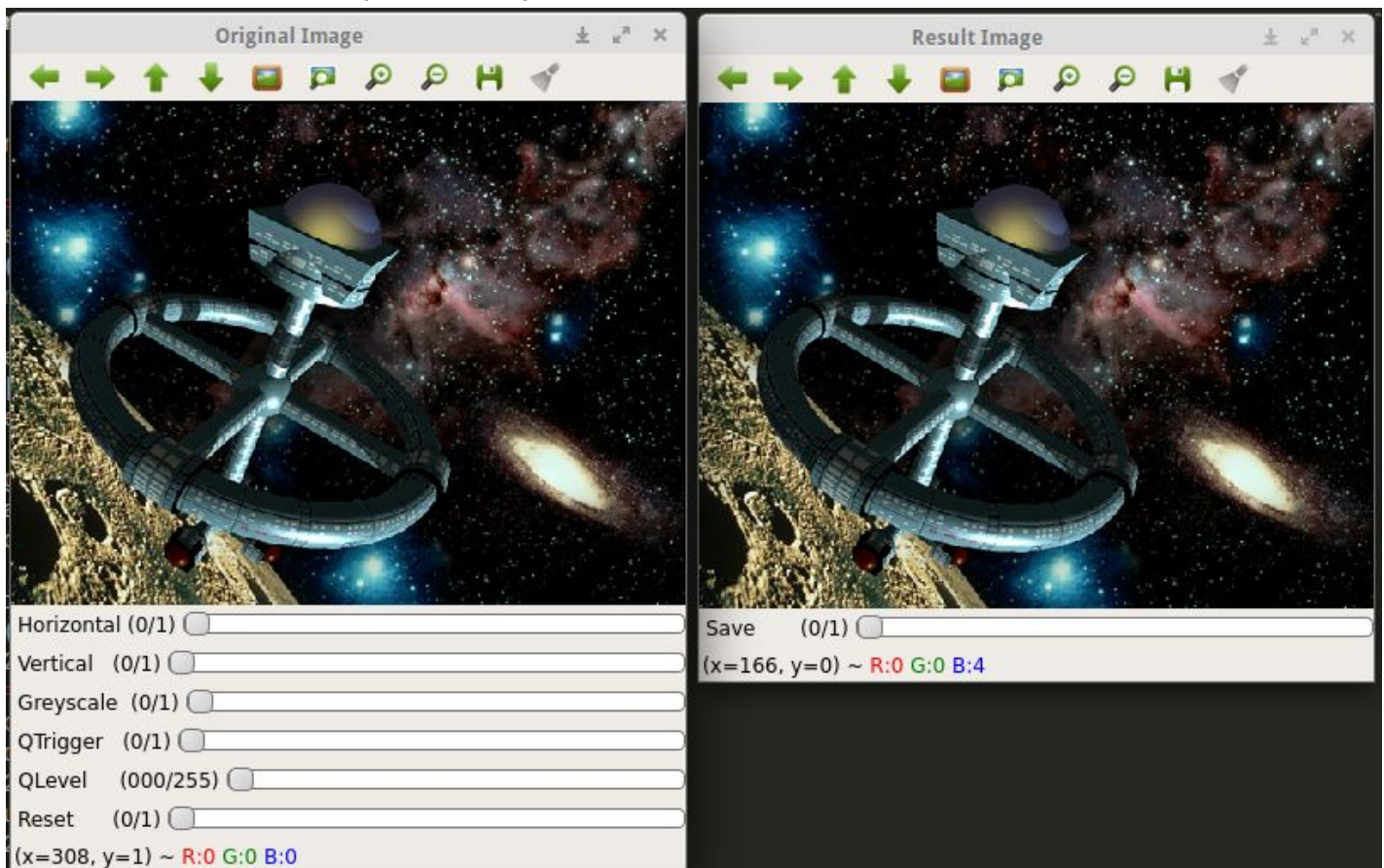
Part II - Reading, Displaying and Executing Operations on Images

The python script was expanded to include additional features:

1. GUI Interface
2. Horizontal Mirroring
3. Vertical Mirroring
4. Conversion to Grayscale
5. Grayscale Quantization
6. Image Save
7. Reset All Operations

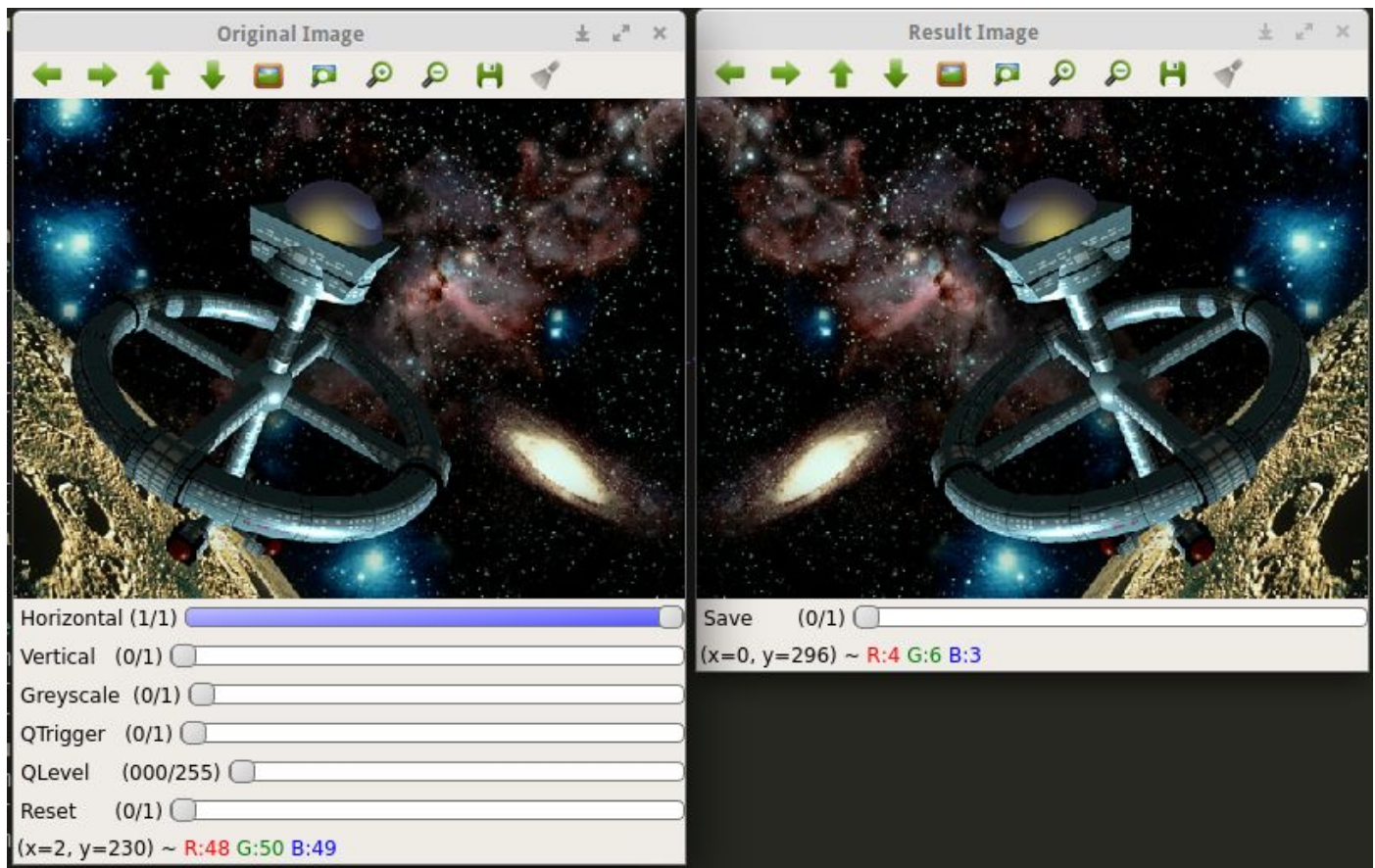
GUI Interface

The script uses an intuitive GUI interface. It uses triggers to apply the operations. Each time a trigger changes, it applies the operation, from 0 to 1 and from 1 back to 0. OpenCV and NumPy are used to open and save the images. All other operations and features are implemented with only common Python functions and properties.



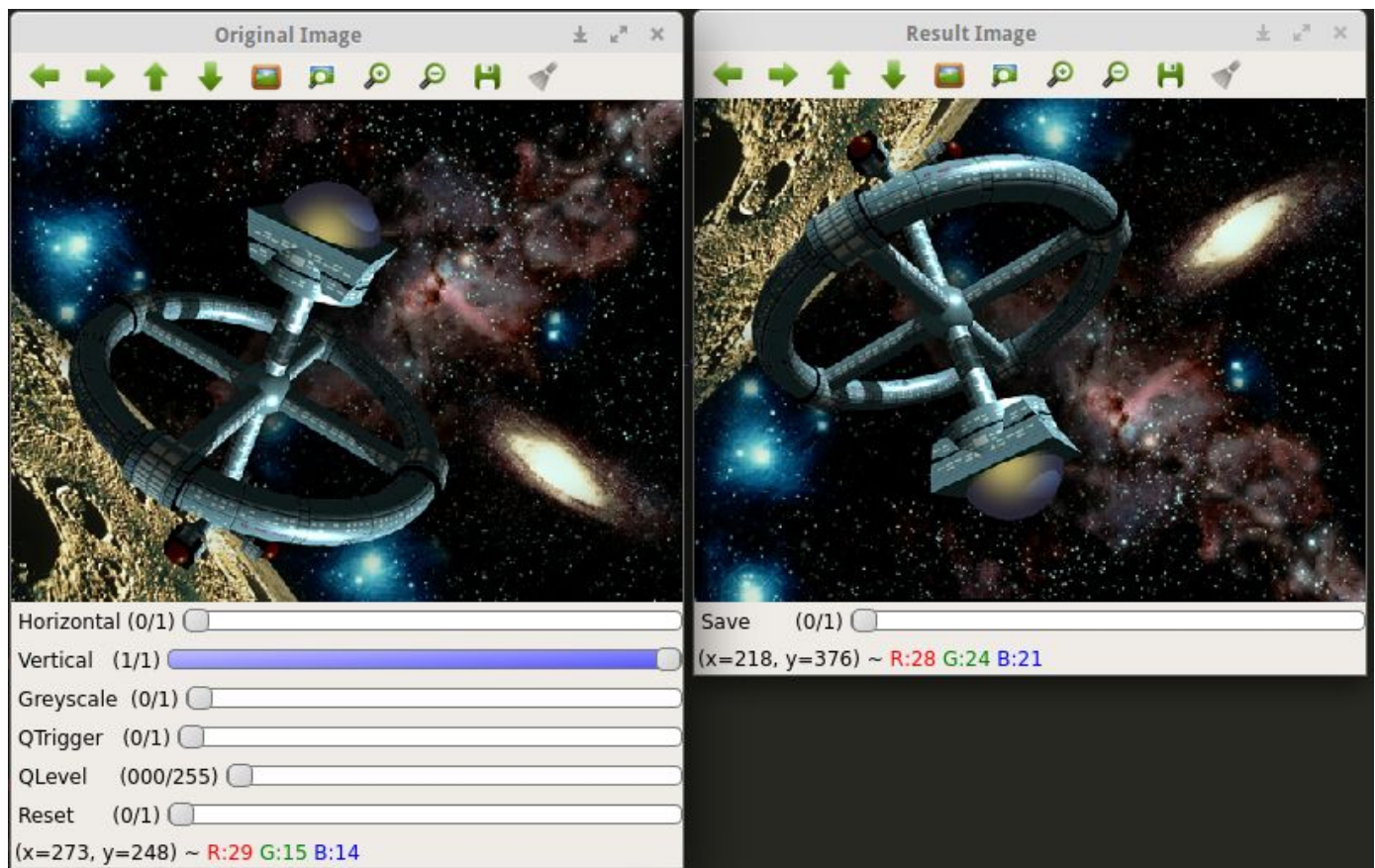
Horizontal Mirroring

When the image is loaded, it is stored as a matrix. The algorithm for horizontal mirroring simply uses Python manipulation of matrices to shift the order of the rows of the matrix.



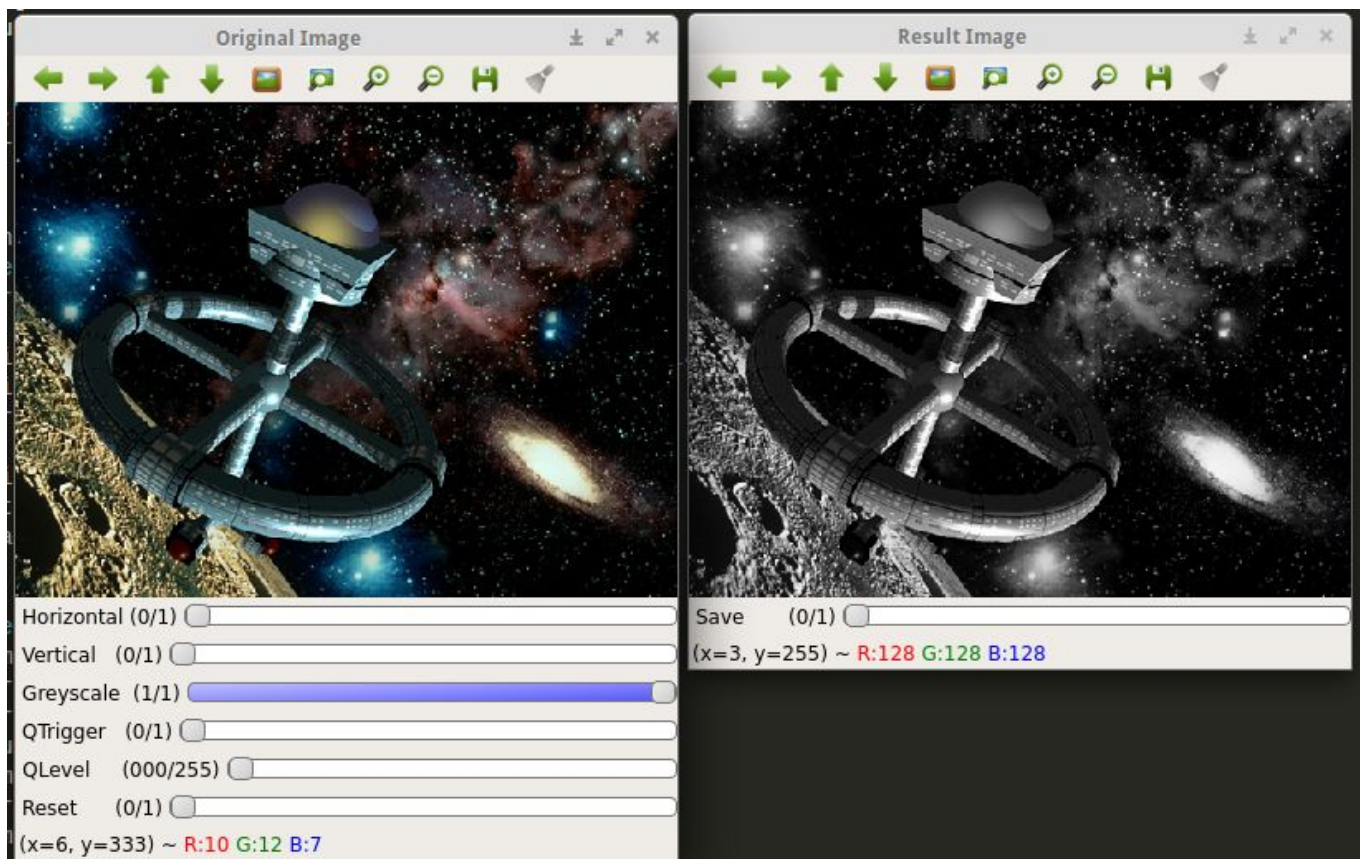
Vertical Mirroring

Just as in horizontal mirroring, the vertical mirroring invert the order of the lines of the matrix, using python manipulation. Note that NumPy or OpenCV is never used for manipulating the image once it's loaded.



Conversion to Grayscale

To convert the image to grayscale, and therefore removing all its color, it is necessary to iterate through the whole matrix of the image. For each pixel, it uses a formula for converting the pixel's RGB values to one luminance value. This new value replaces the three RGB components of the pixel. This conversion can be applied repeatedly, but it has no effect whatsoever on an image that is already in grayscale. This is because the formula for luminance takes the three RGB components and apply a different weight to each of them and gets the sum. The weight is a number between 0 and 1, but the sum of the three weights is exactly 1. This means that if you apply the luminance formula to three components that have the same value, it will remain the same value.



Grayscale Quantization

The GUI has a 0-255 trackbar that is used to receive the number of shades the user wants. Once this number is set, the Quantize Trigger is used to trigger the quantization algorithm. It works by discovering a quantize level, that is, the size of the interval between each shade in the shade histogram. For example, if it is wanted 7 shades of gray, then the image will only contain shades every 36 shades, from 0 to 255(0, 36, 72, ...). By dividing the shade of a pixel from the quantize level, it is possible to discover how far the shade is from where it must go. If a pixel has a luminance of 40, it is 4 shades away from 36, so it is subtracted by 4, and the new value is applied to the pixel.

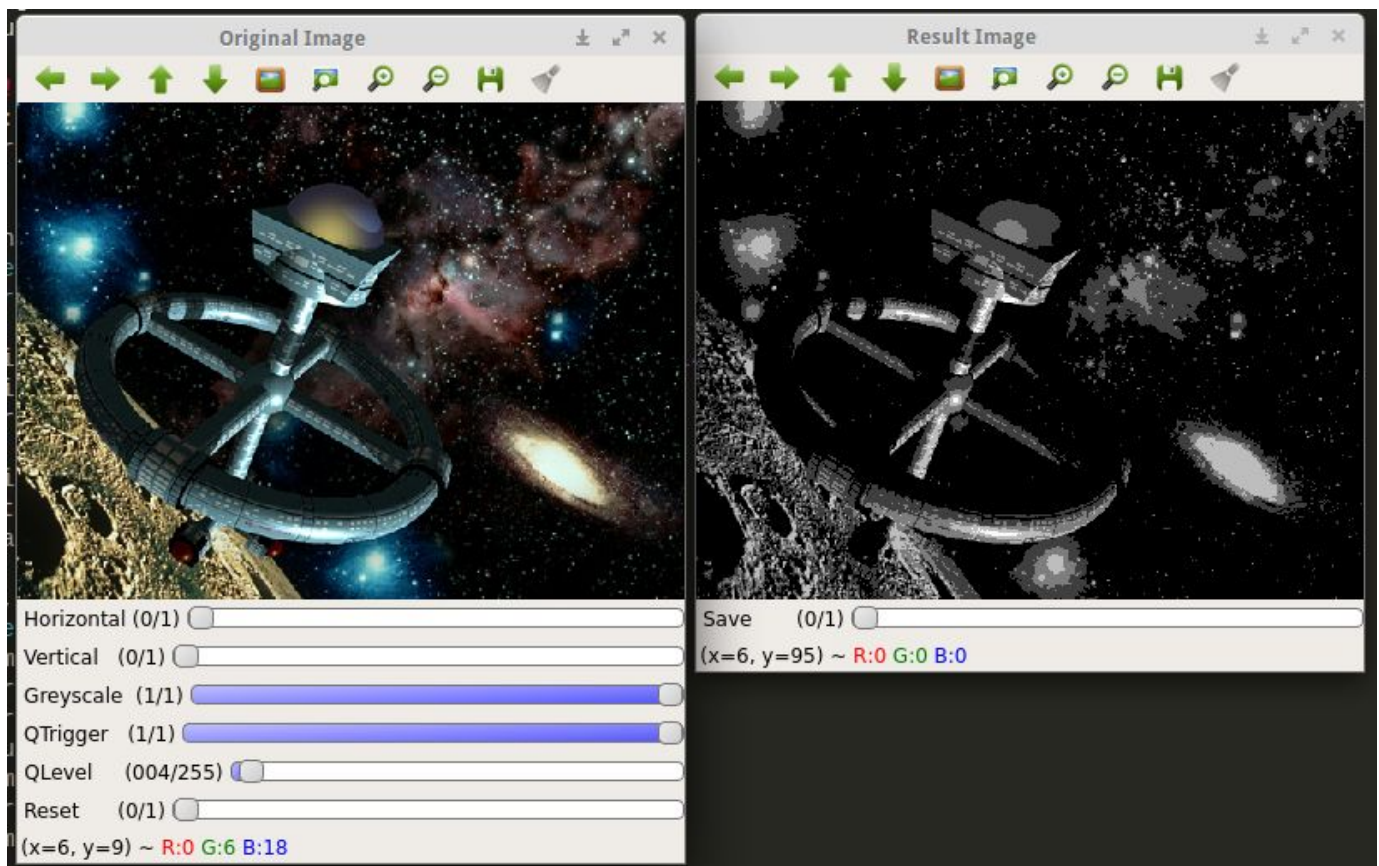
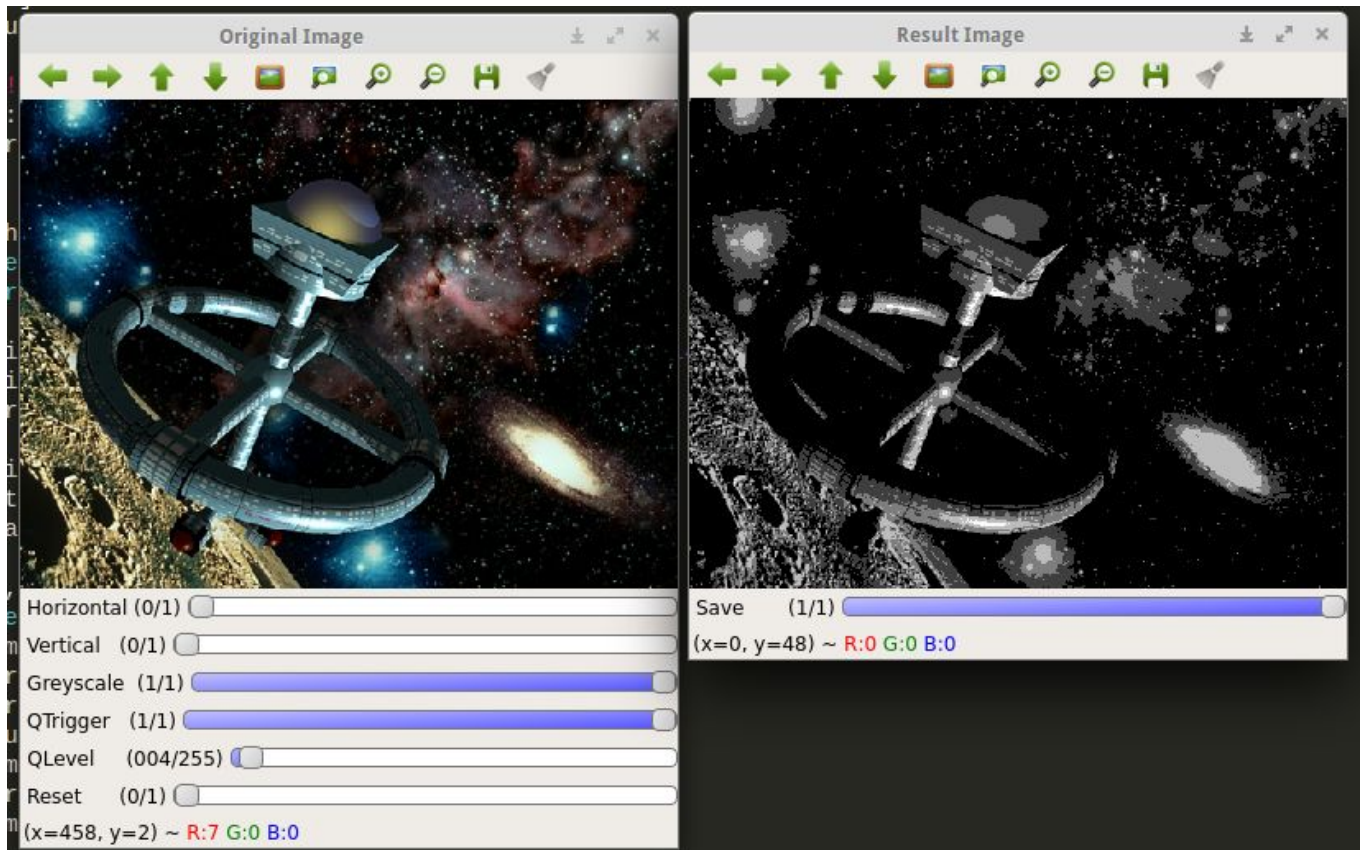


Image Save

The script uses OpenCV to save the matrix of the image to a new JPEG file.



The Save trigger is placed on the right side of the UI, to make it clear that is the right image that is being saved. When the trigger is pressed, a new file is instantly created with the *_new* extension after its current name.



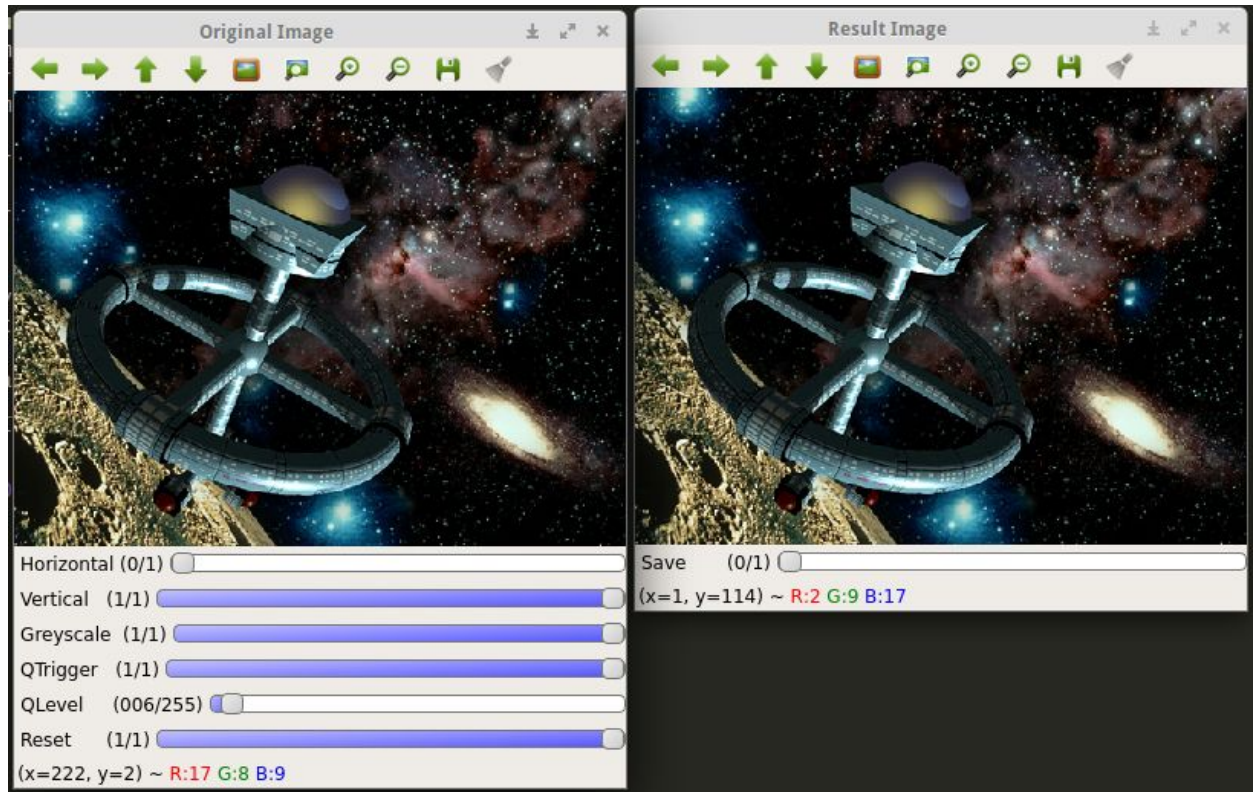
Space_187k.jpg



Space_187k_new.jpg

Reset All Operations

There is a new extra feature implemented. It gives to possibility to reset all past modifications done to the image, returning it to its original state. This way, it is more convenient for the user, who won't need to open and close the program many times to try out different operations.



The source code for this assignment can be found at
https://github.com/rrrichter/ufrgs/tree/master/image_processing/assignment1