Problem Set 1
Image Cartoonifier
Applying Image Processing Filters For Image Cartoonifying

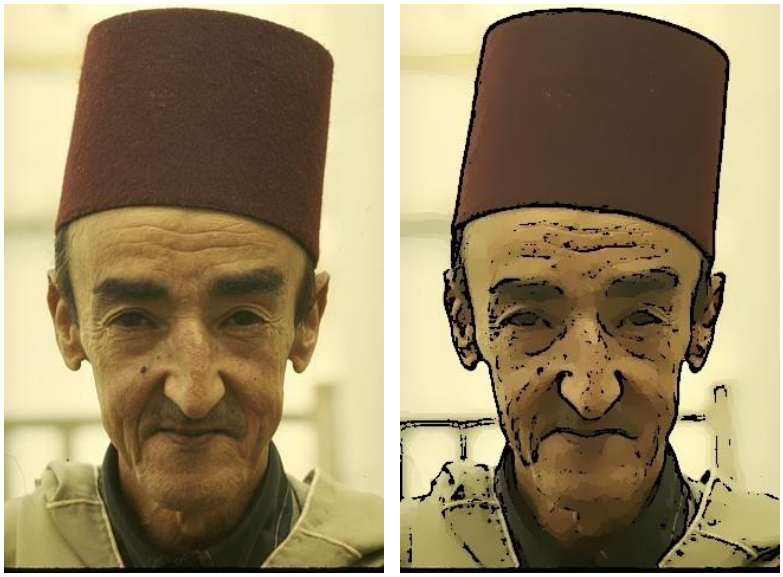
In this problem set we want to make the real-world images look like they are genuinely from a cartoon. The basic idea is to fill the flat parts with some color and then draw thick lines on the strong edges. In other words, the flat areas should become much more flat and the edges should become much more distinct. We will detect edges and smooth the flat areas, then draw enhanced edges back on top to produce a cartoon or comic book effect.

This problem set will cover:

- Applying image processing filters for image smoothing and edge detection.
- Converting a real-life image to a sketch drawing.
- Converting a real-life image to a painting and overlaying the sketch to produce a cartoon.

The following screenshots show the final output for the image cartoonifier program:

Figure 1: Final output for image cartoonifier program



(a) Original image (b) Image after cartoonifying

1 Background - OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS.

We will use OpenCV in this problem set for applying several image processing filters for edge detection and noise reduction.

1.1 Useful Functions

- CreateMat(): Creates a matrix header and allocates the matrix data.
- Smooth(): Smooths the image in one of several ways.
- Threshold(): Applies a fixed-level threshold to array elements.
- Laplace():Calculates the Laplacian of an image.
- CvtColor(): Converts the input image from one color space to another.

2 Generating a black-and-white sketch

To obtain a sketch (black-and-white drawing) of the image, we will use an edge-detection filter, whereas to obtain a color painting, we will use an edge-preserving filter (bilateral filter) to further smooth the flat regions while keeping the edges intact. By overlaying the sketch drawing on top of the color painting, we obtain a cartoon effect as shown earlier in the screenshot of the final program.

There are many different edge detection filters, such as Sobel, Scharr, Laplacian filters, or Canny-edge detector. We will use a Laplacian edge filter since it produces edges that look most similar to hand sketches compared to Sobel or Scharr, and that are quite consistent compared to a Canny-edge detector, which produces very clean line drawings but is affected more by random noise in the image.

Nevertheless, we still need to reduce the noise in the image before we use a Laplacian edge filter. We will use a Median filter because it is good at removing noise while keeping edges sharp.

2.1 Noise Reduction Using Median Filter

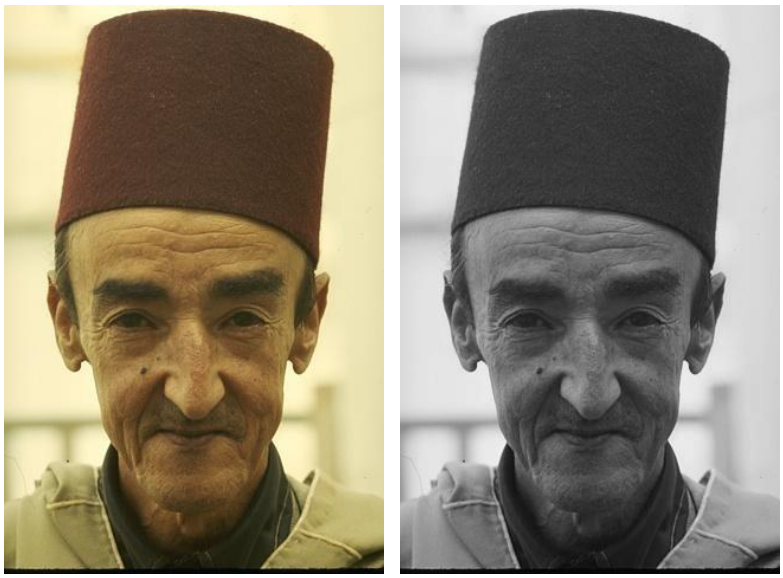
Since Laplacian filters use grayscale images, we must convert from OpenCV's default BGR format to Grayscale, this could be easily done using the CvtColor() function in OpenCV.

For noise reduction, we apply a Median filter with a 7×7 square aperture. This could be done using the Smooth() function.

2.2 Edge Detection Using Laplacian Filter

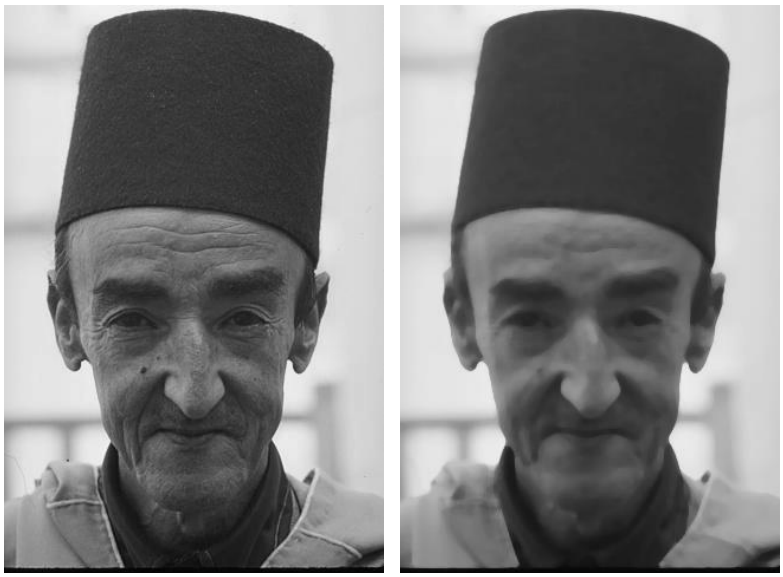
After noise reduction, a Laplacian filter of aperture size = 5 is used for edge detection.

Figure 2: Converting RGB Image to Grayscale



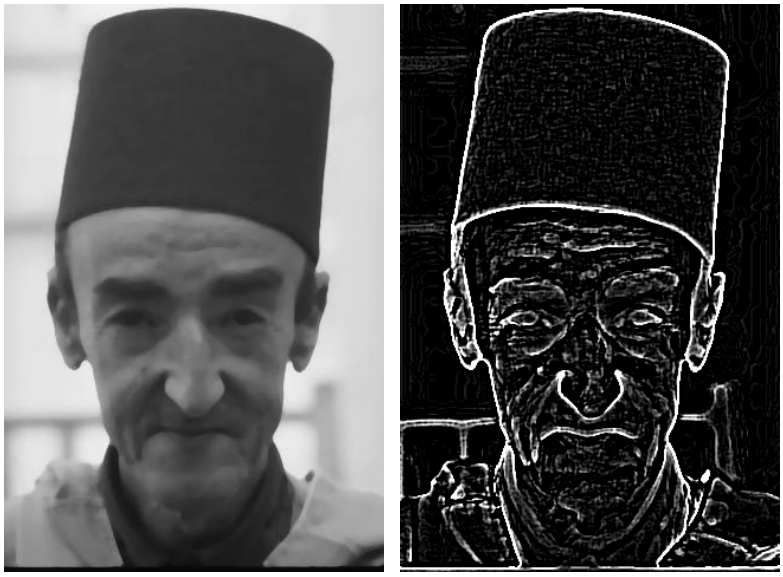
(a) Original RGB Image (b) Grayscale Image

Figure 3: Noise reduction using Median filter



(a) Original Graysclae Image (b) Smoothed Grayscale Image

Figure 4: Edge detection using Laplacian Filter



(a) Smoothed Graysclae Image (b) Edge Detection

The Laplacian filter produces edges with varying brightness, so to make the edges look more like a sketch we apply a binary threshold to make the edges either white or black. This could be done using Threshold() OpenCV function with threshold value = 125.

3 Generating a color painting and a cartoon

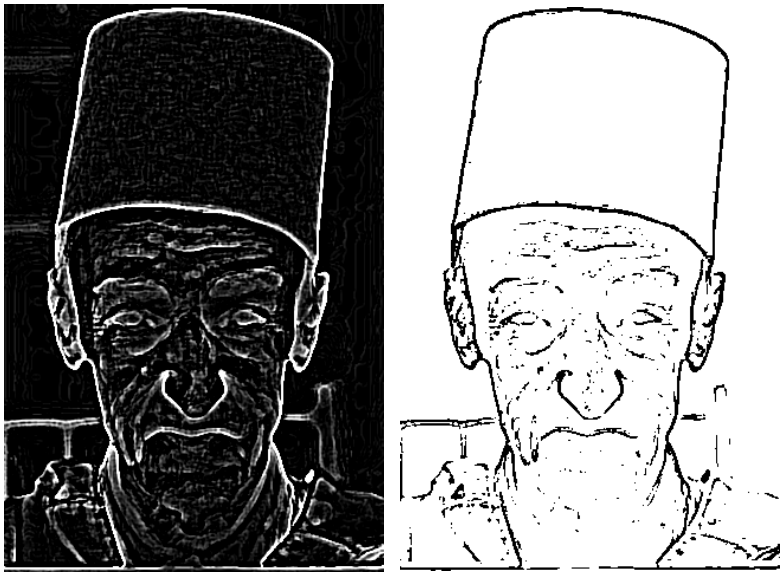
A strong bilateral filter smooths flat regions while keeping edges sharp, and is therefore great as an automatic cartoonifier or painting filter, except that it is extremely slow (that is, measured in seconds or even minutes rather than milliseconds!). We will therefore use some tricks to obtain a nice cartoonifier that still runs at an acceptable speed. The most important trick we can use is to perform bilateral filtering at a lower resolution. It will have a similar effect as at full resolution. This could be done using Resize() function for resizing the image and then applying the bilateral filter using the Smooth() function.

Rather than applying a large bilateral filter, we will apply many small bilateral filters to produce a strong cartoon effect in less time. We will truncate the filter so that instead of performing a whole filter (for example, a filter size of 21 x 21), it just uses the minimum filter size needed for a convincing result (for example, with a filter size of just 9 x 9).

We have four parameters that control the bilateral filter: color strength, positional strength, size, and repetition count. Suitable values for these parameters are:

- Color strength: 9

Figure 5: Edges Thresholding



(a) Output from Laplacian Filter

(b) Output after image thresholding

- Positional strength: 7
- Size: 9
- Repetition count: 7

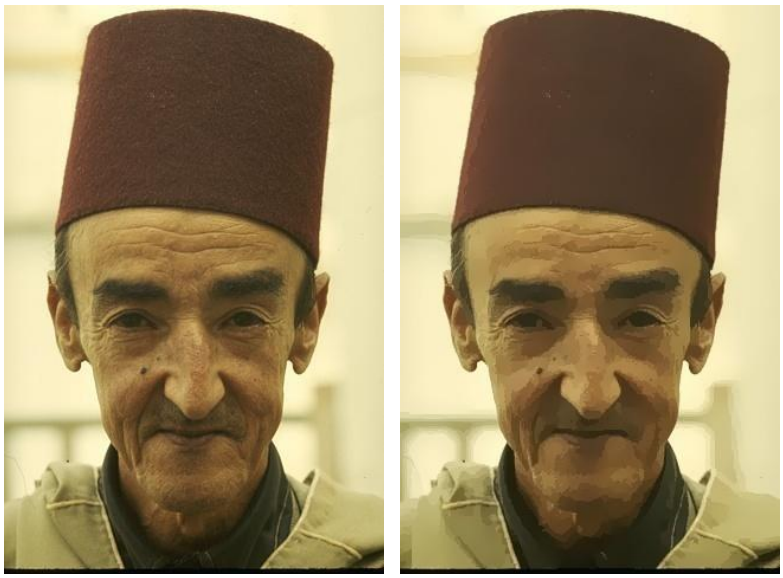
Then we can overlay the edge mask that we found earlier. To overlay the edge mask "sketch" onto the bilateral filter "painting", we can start with a black background and copy the "painting" pixels that aren't edges in the "sketch" mask. This could also be done using the And() OpenCV function.

4 Deliverables

You are required to deliver the following:

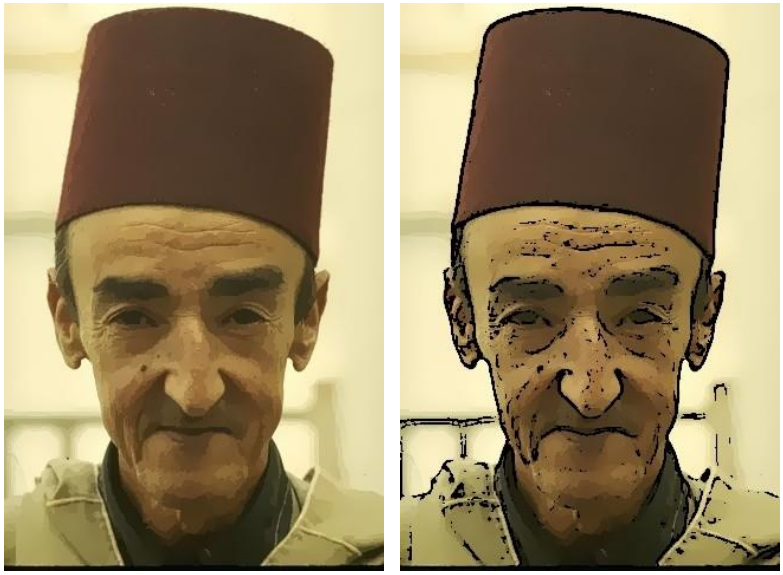
- Your code.
- Output for some test images.
- Report including explanation of your code and representative results on sample test images.

Figure 6: Applying Bilateral Filter



(a) Original Image (b) Output from bilateral filter

Figure 7: Creating Cartoon Effect



(a) Output from bilateral filter (b) Final output