Performance Measurement and Improvement Techniques

Goal

In image processing, since you are dealing with a large number of operations per second, it is mandatory that your code is not only providing the correct solution, but that it is also providing it in the fastest manner. So in this chapter, you will learn:

- To measure the performance of your code.
- Some tips to improve the performance of your code.
- You will see these functions: **cv.getTickCount**, **cv.getTickFrequency**, etc.

Apart from OpenCV, Python also provides a module **time** which is helpful in measuring the time of execution. Another module **profile** helps to get a detailed report on the code, like how much time each function in the code took, how many times the function was called, etc. But, if you are using IPython, all these features are integrated in a user-friendly manner. We will see some important ones, and for more details, check links in the **Additional Resources** section.

Measuring Performance with OpenCV

The **cv.getTickCount** function returns the number of clock-cycles after a reference event (like the moment the machine was switched ON) to the moment this function is called. So if you call it before and after the function execution, you get the number of clock-cycles used to execute a function.

The **cv.getTickFrequency** function returns the frequency of clock-cycles, or the number of clock-cycles per second. So to find the time of execution in seconds, you can do following:

```
e1 = cv.getTickCount()
# your code execution
e2 = cv.getTickCount()
time = (e2 - e1)/ cv.getTickFrequency()
```

We will demonstrate with following example. The following example applies median filtering with kernels of odd sizes ranging from 5 to 49. (Don't worry about what the result will look like - that is not our goal):

```
img1 = cv.imread('messi5.jpg')
assert img1 is not None, "file could not be read, check with os.path.exists()"

e1 = cv.getTickCount()
```