ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN
http://www.wscg.eu
WSCG 2025 Proceedings

Is Python OpenCV slower than its C++ version?

Adam L. Kaczmarek, Kacper Bieryło, Jan Gabryołek and Jakub Macioch
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
ul. G. Narutowicza 11/12
80-233 Gdansk, Poland
adakaczm@pg.edu.pl

ABSTRACT

The paper presents the comparison of the execution time of applications with regard to the programming language in which they were implemented. C++ and Python languages are considered. The selection of the programming language may have a significant impact on the overall performance of the application. The presented research applies to the OpenCV programming library which is a commonly used library in the field of image processing and computer vision. The paper presents experiments with Feature2D, Video I/O and highgui modules included in this library. In general it seems that programs written in C++ should be faster because it is a more low level language than Python. However surprisingly our research showed that in many cases using Python functions of OpenCV leads to the development of faster applications. The difference was particularly significant in case of the Feature2D module. In case of creating GUI it is negligible.

Keywords

Python; C++; OpenCV; programming languages; execution time

1 INTRODUCTION

The research presented in this paper is concerned with verifying the execution time of applications depending on whether they were implemented with the use of the C++ language or Python. Such a comparison is important for the purpose of developing real time applications. Real-time applications necessitate quick data processing, and results must be provided within the specified time limit to be used by the end user.

The investigation utilized here involves the use of the OpenCV library, which is one of the most essential programming libraries used in computer vision [BK08]. In particular, this library can be used to create autonomous robots. This kind of robots operate in real time and they are able to perform tasks without being directly controlled by human operators. Overall, the OpenCV library contains implementations of over 2500 algorithm. Despite its popularity and importance there are only a few research papers addressing the topic of selecting a programming language suitable for using OpenCV. OpenCV makes it possible to use its functions with many languages including Python and C++. Despite its popularity and importance there are only a few research papers addressing the topic of selecting a programming language suitable for using OpenCV. OpenCV makes it possible to use its functions with many languages including Python and C++.

Official data provides information that the Python version of OpenCV is in fact also written in C++, therefore there should not be a deterioration in the execution time caused by switching from C++ to Python [Ope]. Nevertheless, Bustamante et al. showed that there is a the difference between the execution time of applications with regard to the used programming language [BBM+22]. Ramon drew similar conclusions [Ram14]. Thus, we decided to perform tests in order to determine the difference in speed between applications written in C++ using OpenCV functions and their substitutes written in Python. Tests were based on developing the same functionality using these two languages and verifying their speed.

2 RELATED WORK

There are some research paper discussing the issue of speed comparison between applications implemented in C++ and those written in Python.

Bustamante et al. presented a paper which is so far the closest to the topic of our research in comparison to other research paper [BBM+22]. They analyzed two implementations of the same application using OpenCV. The application was designed to process

data for the purpose of performing real-time simulations using the Unity game engine. These simulations were aimed at developing unmanned aerial vehicles.

The first version of the application was implemented in C++. In the second setup they used Python and the Message Queue Telemetry Transport (MQTT) protocol for transferring data between Unity and OpenCV functions. They divided parts of their implementations into three categories. The first category was data processing necessary to perform prior to transmission, the second one was transmission process and the third category was data processing itself. This data processing was performed by OpenCV.

Their finding were such that in the phase of data processing it was much faster to use Python in comparison to C++ because data provided by Unity was more compatible to OpenCV functions available in Python. The processing time using Python was between 6 ms and 17 ms for the data they used. In C++ it was in the range from 35 to 57 ms. The time of data transmission was similar in case of both kinds of implementations. Bustamante et al. also noticed other differences related to the programming language. In case of an algorithm for color detection C++ proved to be faster. On average processing data with C++ lasted 0.7 ms while using Python the average processing time was 2.5 ms. However, the algorithm for Face Detection ran faster using Python. A single image was processed using Python in between 7 and 12 ms and using C++ between 5 and 12 ms. Their final conclusion was such that using Python is more advisable in case of software that they were developing mainly because of a better compatibility of data formats between Unity and OpenCV. Using MQTT also supported the speed of applications implemented in Python.

Ramon noticed that capturing images using Python functions implemented in OpenCV takes twice as much time as using its C++ version [Ram14]. He showed that relation on the basis of the execution time of the VideoCapture class methods. He performed his experiments on the Intel Galileo board.

Another paper concerned with the issue of comparing the speed of Python with regard to C++ was written by Di Domenico et al. [DDLC23]. They focused on the performance of GPU. They tested three configurations:

- Python with Numba environment for enabling CUDA support
- C++ operating on CUDA
- C++ with OpenACC

Domenico et al. took advantage of NAS Parallel Benchmarks for testing the execution time of these three kinds of programming environments. The conclusion of their research was such that applications implemented in Python had a similar performance as those written in C++ when the CUDA architecture is used. However, results that they presented indicate that in case of some tests the Python implementation was over 3 times slower. They also noticed that Python used with CUDA was faster than the configuration with C++ and OpenACC.

Plauska, Liutkevičius and Janavičiūtė presented a paper in which they compared the performance of C/C++, MicroPython, Rust and TinyGo Programming Languages on the ESP32 Microcontroller [PLJ23]. They worked on this comparison in the context of developing Internet of Things (IoT). Comparison of Python and C++ was also discussed by Ferron and Manduchi [FM22]. They focused on real-time applications.

Another researcher who investigated the speed of software using different languages were Briggs [Bri06]. He compared performance of applications for calculating exact real arithmetic in Python, C++ and C. He did not present exact times of executions of applications implemented with the use of different kinds of languages. He only noted that the code in C would typically be 10 times faster than C++ and more than 100 times faster than the Python version. He also claimed that achieved efficiency is compiler-dependent.

Dedner and Klöfkorn compared the performance of Python and C++ on the example of the Riemann problem [DK22]. They tested two methods for solving this problem, the first one used the Cartesian grid and the second one was based on the unstructured grid. When the first method was used they did not observed a difference in the performance depending on the programming language. However, in case of the second method they reported that the application implemented in Python was 10 % slower.

3 EXPERIMENTS

This paper aims at answering the question if it is advantageous to choose C++ instead of Python to develop software running faster even though the development time would be longer and more complicated. The development time is shorter using Python because this language is more expressive and its productivity is higher [Ber13]. We have performed a series of experiments with different functions of the OpenCV library in order to verify how the programming language selection affects the overall performance of a resulting application.

Our experiments covered functions included in Video I/O, Feature2D and highgui modules of OpenCV. In the Video I/O module the following three functions were tested: the constructor of the VideoCapture class, the read function of the VideoCapture class and the write function of the VideoWriter class Another experiment

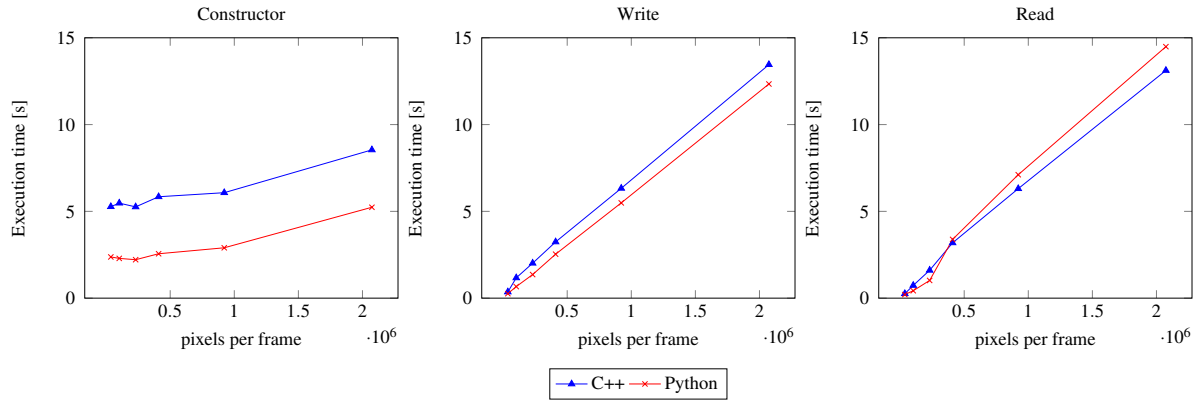

Figure 1: Videoio functions, execution time

was concerned with testing functions from the feature2D module. Three corner detector functions were selected for experiments: FAST, AKAZE and BRISK. The study of the highgui module included function imread, imwrite and imencode.

All tests of Video I/O involved running the code with a specified number of repetitions. The tests were repeated ten times and the two extreme results were discarded. Then, the average time was calculated from the remaining results. The time taken to create an object of this class for files of different resolutions was measured for 1000 repetitions. In order to test execution time of tested functions two programs were implemented in both programming languages ensuring that the code was as similar as possible to maintain consistency.

3.1 test data

We tested OpenCV functions with randomly selected images and videos. However input data was selected with regard to their resolution.

The data used for the experiment of VideoIO were copies of the same video in resolutions of 144p, 240p, 360p, 480p, 720p, 1080p (16:9 aspect ratio). The video is 2 minutes 28 seconds long and it has a frame rate of 23.98 frames per second.

High-resolution wallpapers were chosen for tests of Feature 2D due to containing a large amount of pixels and different elements on the image, such as still life, logos and cartoons. The tests were run on sets of ten, twenty, thirty, forty and fifty images. The size of the images varied form approximately 200Kb to 1,9 Mb, averaging around 600Kb. For tests concerning the size of a single image, one of the previous images was scaled to: 393Kb; 828Kb; 1,1Mb; 1,59Mb and 1,96Mb respectively. The same image was used, because while using different images for each file size, the results were highly inconsistent.

The input data for the experiment with highgui consisted of JPEG images, each with a resolution of 4624 x 2604 pixels. These images depicted various city objects, providing a diverse data set for the experiment.

4 RESULTS

In Fig. 1 charts are presented, showing the execution time of tested functions depending on the size of the input file. Results showed that the VideoCapture class constructor in Python has shorter execution time and this one is the most significant difference in the tested functions. For resolutions up to 720p, the execution time is over two times shorter. In the case of the "read" function, there are noticeable differences in favor of Python for lower resolution videos. However, the situation reverses for videos with resolutions greater than 360p, where C++ gains a slight advantage. The write function has a shorter execution time in Python, but its advantage diminishes as the file resolution increases.

Plots for data acquired during the tests of Feature 2D are shown in Fig. 2. The results show that for both C++ and Python, the Fast function is executed with similar speed. For Akaze and Brisk functions, Python is faster and scales better with the number of images processed. In both cases, the Fast function is the fastest, while Brisk is the slowest. Looking at the average time of execution for a single image it can be assumed, that for both C++ and Python it is unrelated to the number of images processed. The execution time size show that Python is faster when it comes to Akaze and Brisk functions.

The findings regarding the highgui module showed that on average the C++ implementation was around 5 percent faster than the Python counterpart when handling 50 input images. With fewer images, the difference was slightly smaller.

5 CONCLUSIONS

The answer to the question "Is Python OpenCV slower than its C++ version?" is such that in many cases applications written in Python are faster than their alternatives implemented in C++. It applies both to VideoIO

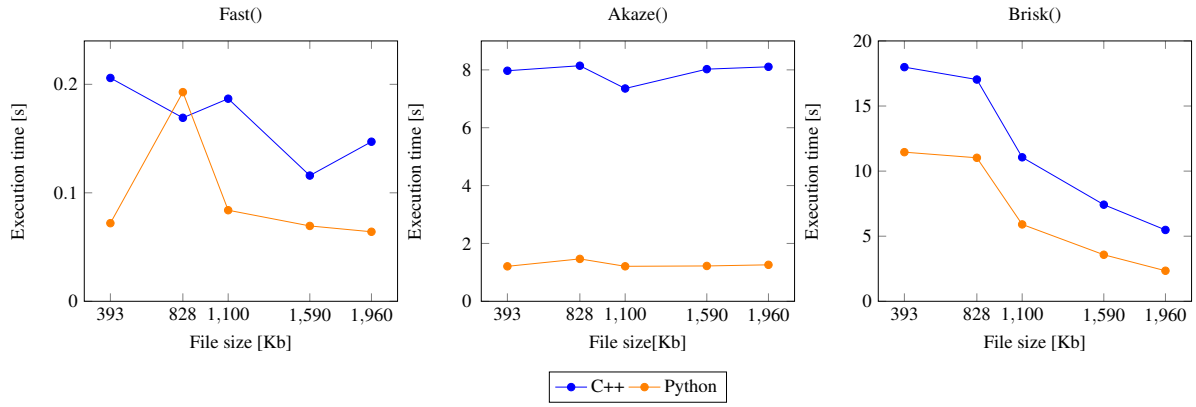

Figure 2: Time of execution for each method depending on the file size

and feature2D modules. In case of creating a user interface with OpenCV differences are negligible. It also needs to be noticed that C++ is a much more complicated language because using it exposes an application to segmentation fault errors which can occur if the program is not implemented correctly. Therefore, the conclusion of the research presented in this paper is such that it is advisable to select Python over C++ as a programming language for using OpenCV.

In future work we plan to further investigate the issue of programming language usage with OpenCV. There are two path of further development. The first one is analyzing a greater number of functions including experiments with a narrow scope focused on a particular functionality of OpenCV. The second path for further research is the investigation of the code of OpenCV in order to determine reason for differences in processing time with regard to programming language. That reason may lie in issues related to conversions in formats of input data used by OpenCV functions. A similar kind of problem was observed by [BBM+22] as described in Sect. 2.

6 REFERENCES

[BBM+22] András Bustamante, Lidia M. Belmonte, Rafael Morales, António Pereira, and Antonio Fernández-Caballero. Video processing from a virtual unmanned aerial vehicle: Comparing two approaches to using opencv in unity. *Applied Sciences*, 12(12), 2022.

[Ber13] Donnie Berkholz. Programming languages ranked by expressiveness. `https://redmonk.com/dberkholz/2013/03/25/programming-languages-ranked-/by-expressiveness`, 2013. Accessed: 2024-01-10.

[BK08] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning Opencv, 1st Edition*. O'Reilly Media, Inc., first edition, 2008.

[Bri06] Keith Briggs. Implementing exact real arithmetic in python, c++ and c. *Theoretical Computer Science*, 351(1):74–81, 2006. Real Numbers and Computers.

[DDLC23] Daniel Di Domenico, João V. F. Lima, and Gerson G. H. Cavalheiro. Nas parallel benchmarks with python: a performance and programming effort analysis focusing on gpus. *The Journal of Supercomputing*, 79(8):8890–8911, 2023.

[DK22] Andreas Dedner and Robert Klöfkorn. Extendible and efficient python framework for solving evolution equations with stabilized discontinuous galerkin methods. *Communications on Applied Mathematics and Computation*, 4(2):657–696, 2022.

[FM22] Nicolo Ferron and Gabriele Manduchi. Using python modules in real-time plasma systems for fusion. *Sensors*, 22(18), 2022.

[Ope] OpenCV. Introduction to opencv-python tutorials. `https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html`. Accessed: 2024-01-10.

[PLJ23] Ignas Plauska, Agnius Liutkevičius, and Audrone Janavičiūtė. Performance evaluation of c/c++, micropython, rust and tinygo programming languages on esp32 microcontroller. *Electronics*, 12(1), 2023.

[Ram14] Manoel Carlos Ramon. *Using OpenCV*, pages 319–400. Apress, Berkeley, CA, 2014.