A Comparative Analysis of Implementation Performances for Image Processing Applications Used to Control Robotic Arms

Radu-Stefan Ricman, Roland Szabo
Applied Electronics Department
ETcTI, Politehnica University Timisoara
Timisoara, Romania
ricman.radu@gmail.com, roland.szabo@upt.ro

Aurel Gontean
Applied Electronics Department
ETcTI, Politehnica University Timisoara
Timisoara, Romania
aurel.gontean@upt.ro

***Abstract*—This paper presents an analysis of code implementation performance for image processing algorithms. The test is made for image processing algorithms for robotic arms, but it is suitable for any type of image processing software. Image processing software can use quite big amount of resources, so this way it can be tested which platform, which operating system or which programming language is the most suitable for usage. The image processing task is a color detection task with some line and circle overlays for robotic arm's guidance. The implementations were tested base on code line numbers, code size on disk, binary file size on disk, used memory during execution and used CPU during execution.**

Keywords—code execution; code size; image processing algorithms; performance comparison; resource utilization; robotic arm control; software implementation.

I. INTRODUCTION

This paper evaluates different implementation of image processing programs in order to conclude which is the most suitable [1].

It is know that image processing algorithms can take a lot of resources [2], but with comparisons like this it can be concluded how to implement the specific task in an optimal way [3].

Comparisons of codes can be helpful for further implementations in order to know which is the best solution in the specific environment [4].

There were compared six different implementations, each on different platform, operating system and programming language.

In the implementations it is made a color recognition algorithm with some line and circle overlays to guide the robotic arm's movement.

The six implementations were:

- VHDL on FPGA;
- LabWin-dows/CVI and Robo-Realm in Windows;
- LabVIEW and NI Vision Development Module in Windows;
- LabWin-dows/CVI and NI Vision Development Module in Windows;
- C++ and OpenCV in Linux;
- Python and OpenCV in Linux.

The implementations were tested based on:

- Code line numbers;
- Code size on disk;
- Binary file size on disk;
- Used memory during execution;
- Used CPU during execution.

All this is according to Table I.

II. PROBLEM FORMULATION

In the laboratory there was a robotic arms for which we had been made different implementations to control it using image processing and cameras. The control program recognized the robotic arms position space [5] using video cameras and color filtering [6] and overlaid on the initial image some guide lines and circles in order to know where the robotic arm has to move. The implementations were made on different platforms [7], operating systems and programming languages and it was a useful information to know which implementation method is the most suitable in a specific condition [8].

The authors would like to thank Politehnica University Timisoara and Continental Automotive Romania for the given support.

978-1-5386-5186-5/18/$31.00 ©2018 IEEE

TABLE I. IMPLEMENTATION COMPARISON OF ROBOTIC ARM CONTROL SYSTEMS USING VIDEO CAMERAS

Used Resources	VHDL on FPGA	LabWindows/CVI and Robo-Realm in Windows	LabVIEW and NI Vision Development Module in Windows	LabWindows/CVI and NI Vision Development Module in Windows	C++ and OpenCV in Linux	Python and OpenCV in Linux
Code line numbers	13722	2091	N/A (graphical language)	1214	1030	539
Code size on disk	608 KB (13 *.vhd files)	64 KB (2 *.c files and 2 *.h files)	896 KB (6 *.vi files)	48 KB (1 *.c file and 1 *.h file)	40 KB (1 *.cpp file)	28 KB (1 *.py file)
Binary file size on disk	860 KB (1 *.bit file)	964 KB (1 *.exe file)	1.28 MB (1 *.exe file)	978 KB (1 *.exe file)	44 KB (1 file with no extension)	N/A (script)
Used memory during execution	N/A (1103 of 54576 Flip Flops)	47064 KB	60268 KB	42180 KB	79872 KB (of 2 GB RAM on PC) 50332 KB (of 512 MB RAM on ZYBO or ZedBoard)	67584 KB (of 2 GB RAM on PC) 54002 KB (of 512 MB RAM on Raspberry PI)
Used CPU during execution	N/A	52% (of Intel Core 2 Duo @ 2.4 GHz)	32% (of Intel Core 2 Duo @ 2.4 GHz)	24% (of Intel Core 2 Duo @ 2.4 GHz)	61% (of Intel Core 2 Duo @ 2.4 GHz) 76,6 % (of ARM Cortex – A9 Dual-core @ 650 MHz)	29,8% (of Intel Core 2 Duo @ 2.4 GHz) 46% (of ARM 11 @ 700 MHz)

III. PROBLEM SOLUTION

To know which would be the best implementation it was needed to be done an evaluation with some charts. the charts were done based on code line numbers, code size on disk, binary file size on disk, used memory during execution and used CPU during execution. These criteria were thought are essential, when deciding which would be the best platform, operating system or programming language do to the job done.

Table I. presents a comparison of the resources used on different implementations. It was gathered every parameter of different implementations, so it could be shown which implementation is optimal based on the used resources.

On Fig. 1 is represented the code line numbers in different implementations, the least line numbers are in the Python script and in the C code on the Linux operating system. These implementations are using the OpenCV library. On the opposite stands the VHDL code, where everything must be implemented from scratch.

There can be seen that for VHDL it is a monumental work to implement an image processing application for a robotic arm, because it has to be started with the basics, the VHDCI interface for the stereo cameras, the HDMI interface for the display and the RS-232 interface for commanding the robotic arm.

On top of this it has to be implemented the image processing libraries, which in the case of operating system usage are just included by the usage of the OpenCV or the NI Vision Development module.

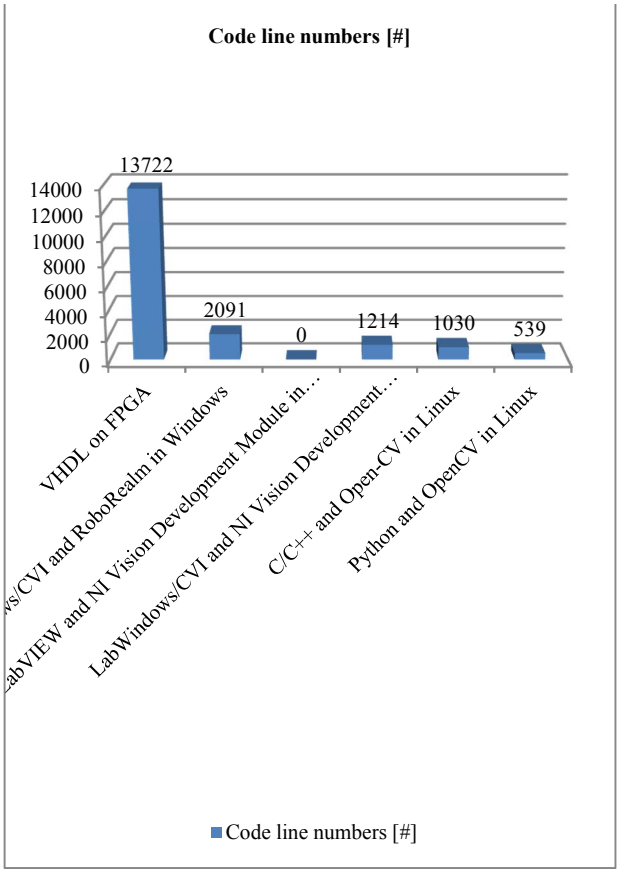

Fig. 1. Comparison of code line numbers in different implementations.

On Fig. 2 is represented the code size on disk in various implementations, where the champions are the Python script and C the code on the Linux operating system. On the other end it is the implementation in LabVIEW, which generates an (unexpectedly) big file.

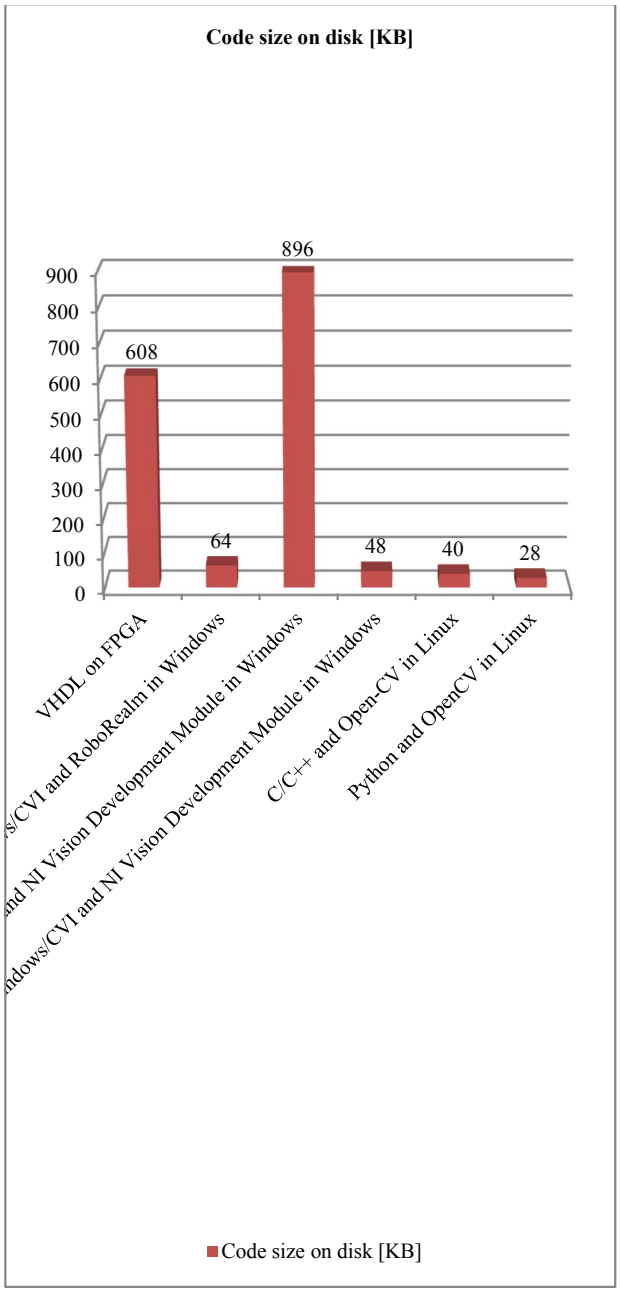

Fig. 2. Comparison of code size on disk in different implementations.

On Fig. 3 it is shown the binary file size on disk, the winner is the code in C language in Linux operating system, because the code is very short, and OpenCV library functions are allocated dynamically at runtime, so they are not compiled in the executable. For the Python script it is not the case to talk about an executable, because it is not generated, it is a scripting language, the Python script is running on the server. LabVIEW it is on the other side, where everything is compiled in the executable, this way resulting a relatively large file.

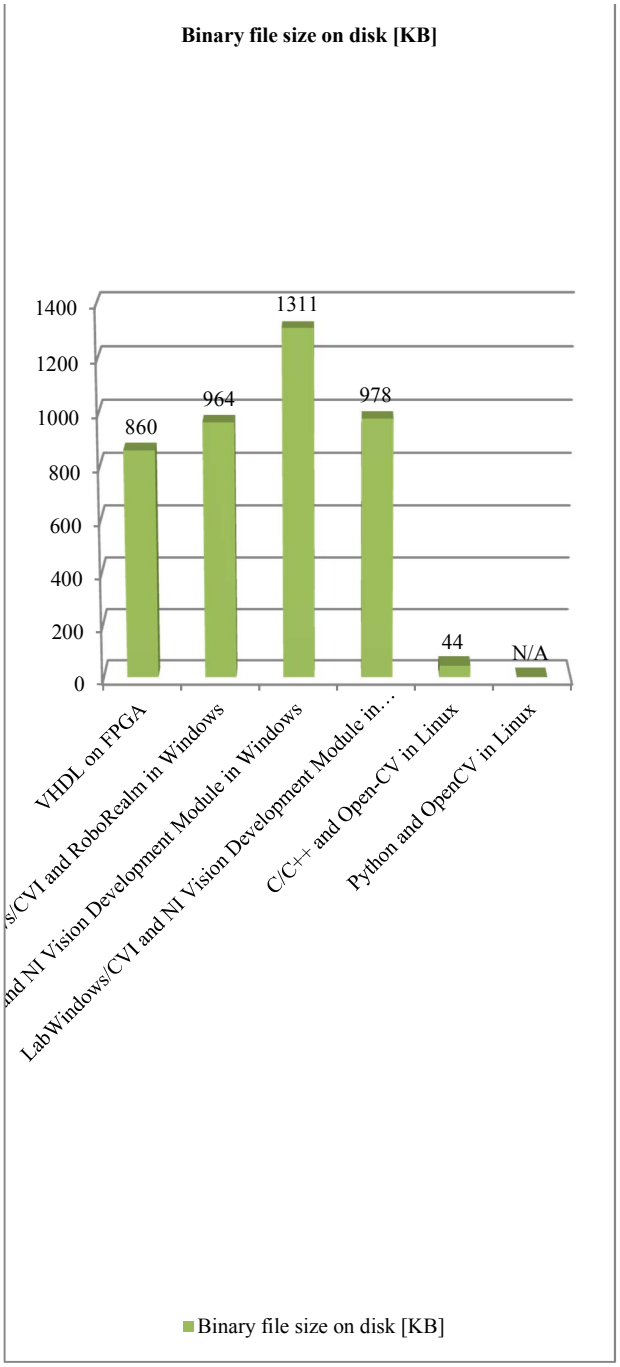

Fig. 3. Comparison of binary file size on disk in different implementations.

On Fig. 4 it is shown a comparison of the memory used in different implementations. The implementation in VHDL, it is the one without memory usage, in the implementation summary made by the Xilinx tool, the used resources were 2%. On the other hand a bad position has the implementation using the Python script and the code in C language in Linux operating system. The implementation occupies a larger memory, because all the OpenCV library is dynamically loaded into memory at runtime and this implementation it is programmed to use the OpenCV librarym which has a big variety of image processing functions, this way has a high memory usage.

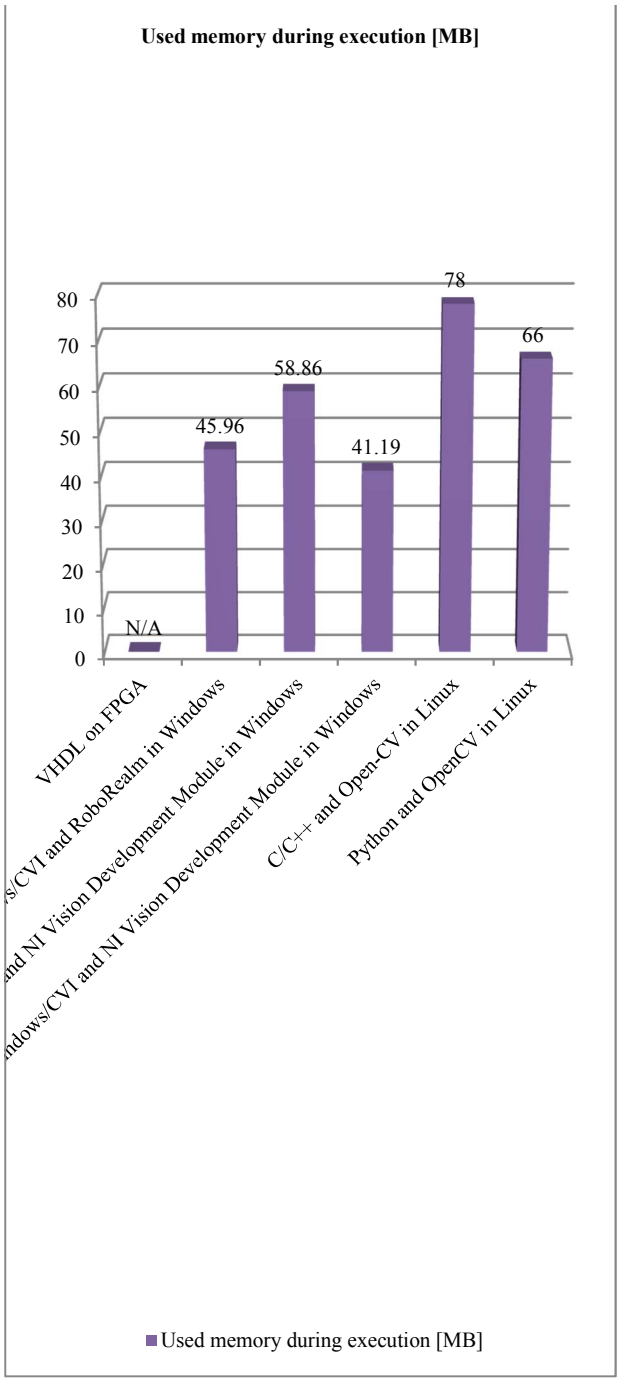

Fig. 4. Comparison of used memory during execution in different implementations.

On Fig. 5 it is shown a comparison of the percentage of the CPU usage (central processing unit) during the execution of the implementations described in this research study. Again, the winner with the fewest resources used, is the implementation in VHDL, and the other side it is the implementation in C language running on the Linux operating system. This must load the OpenCV dynamic libraries, so it requires the most processing power, on the chart it can be seen that this can be reduced when using a Python script, running on the server, and this way the CPU usage can be reduced.

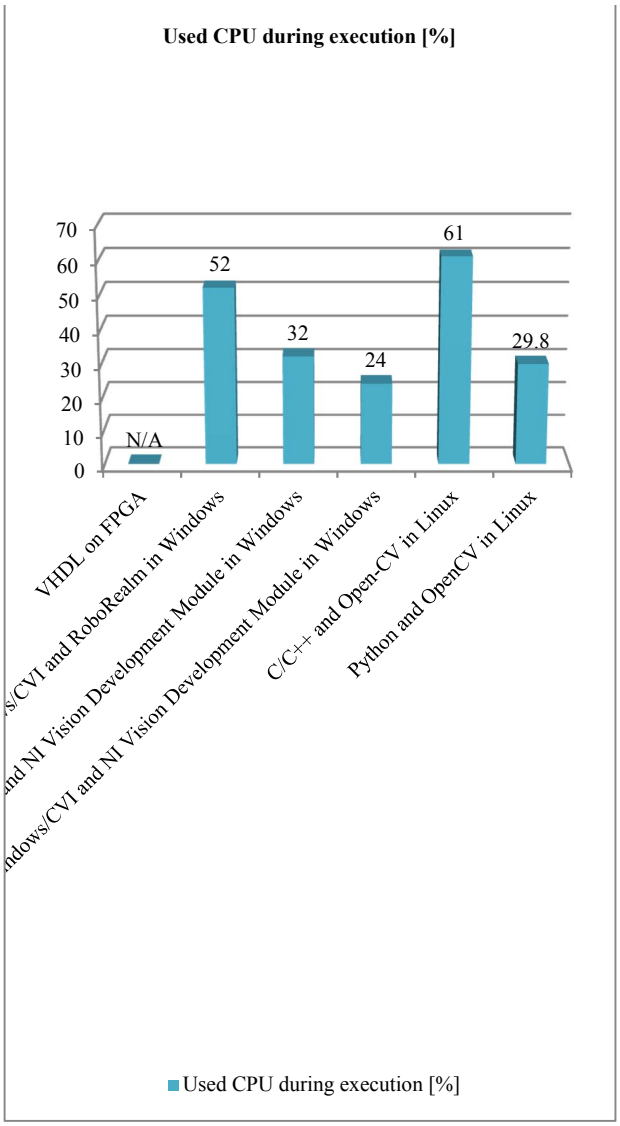

Fig. 5. Comparison of used CPU during execution in different implementations.

Finally (on Fig. 6) it can be seen that each implementation has its own particularities. The VHDL implementation occupied fewer resources, but the code is very long and very complicated, thus requiring more effort from the engineer who would make the implementation. In the Linux implementations the used resources during execution are relatively high, but it cannot be forgotten that the implementations require less effort, than other implementations, OpenCV library is the most flexible, but the resource consumption is relatively high. Since in our day's even the embedded systems are relatively powerful in terms of resources, the usage of more resources would not be a problem, especially since a system that controls a robotic arm has nothing running in parallel. However the winner would be the implementation in the Python script on the Linux operating system, which is relatively simple and does not use too much resources. It mustn't be forgotten that Linux implementations are also attractive, because they can be ported easily on several embedded systems.

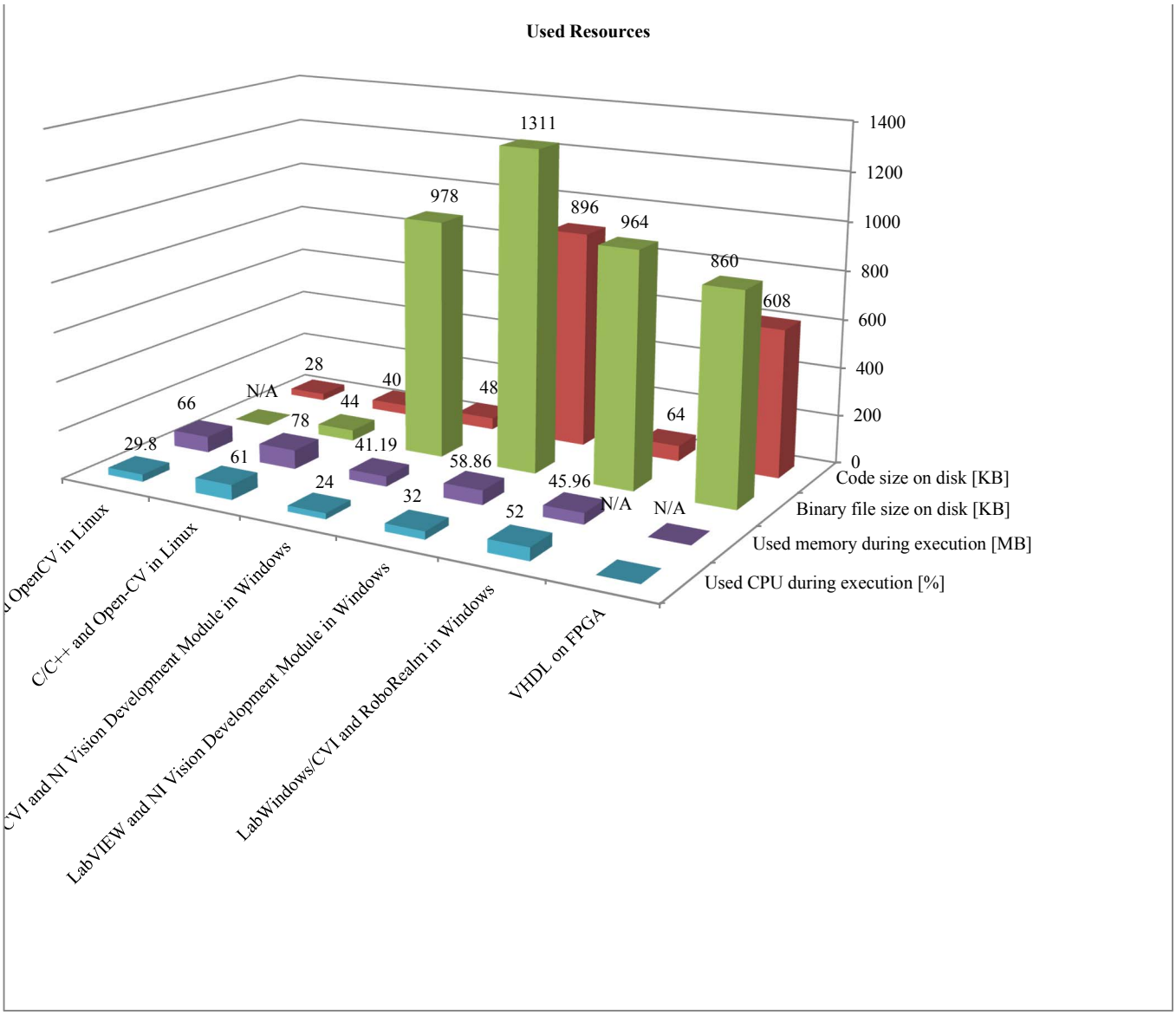

Fig. 6. Comparison of used resources in different implementations.

IV. Conclusion

Based on the successful control of the Lynxmotion AL5B educational robot, the table, contains virtually all implementations, combinations of resources, programming languages and operating systems available at the laboratory. The results were also validated with the control of the SCORBOT-ER III educational/industrial robot.

By comparing various implementations, the following was resulted:

a. For VHDL, the system occupied fewer resources, but the code is very long and very complicated, thus requiring more effort from the engineer, who makes implementation.

b. For the implementations in Linux the used resources during execution are relatively high, but it cannot be forgotten that the implementation requires less effort than other implementations, the OpenCV library is the most flexible, but resource consumption is relatively high.

c. Since these on days and embedded systems are relatively powerful, the usage of more resources would not be a problem, especially since a system that controls a robotic arm, does not require parallel computing.

d. The best compromise is the Python script in the Linux operating system, which is relatively simple and does not use too much resources.

e. It mustn't be forgotten, that Linux implementations are also attractive, because they can be ported easily on several embedded systems.

System testing was performed after moving the robotic arm's gripper in all eight corners in coverage of the robotic arm (left top - bottom, right top - bottom corners on both sides of the robotic arm) and the robotic arm managed to reach all

the corners autonomously, without being programmed with the target points by a previous calibration process. It was real-time control, the target position computation was performed by the algorithm (and the guide lines for computing the distances were redrawn in real-time on the acquired image, at each displacement of the robotic arm).

REFERENCES

[1] Zhang Hanqi, "Hand/eye calibration for electronic assembly robots," IEEE Transactions on Robotics and Automation, vol. 14, issue 4, 1998, pp. 612-616.

[2] Zhuang Hanqi, Wu Wen-Chiang, Zvi S. Roth, "Camera assisted calibration of SCARA arms," IEEE Robotics & Automation Magazine, vol. 3, issue 4, 1996, pp.46-53.

[3] R. Kelly, "Robust asymptotically stable visual servoing of planar robots," IEEE Transactions on Robotics and Automation, vol. 12, issue 5, 1996, pp. 759-766.

[4] F. C. A. Groen, G. A. den Boer, A. van Inge, R. Stam, "A chess-playing robot: lab course in robot sensor integration," IEEE Transactions on Instrumentation and Measurement, vol. 41, issue 6, 1992, pp. 911-914.

[5] Aghili Farhad, "A Prediction and Motion-Planning Scheme for Visually Guided Robotic Capturing of Free-Floating Tumbling Objects With Uncertain Dynamics," IEEE Transactions on Robotics, vol. 28, issue 3, 2012, pp. 634-649.

[6] P. K. Allen, A. Timcenko, B. Yoshimi, P. Michelman, "Automated tracking and grasping of a moving object with a robotic hand-eye system," IEEE Transactions on Robotics and Automation, vol. 9, issue 2, 1993, pp. 152-165.

[7] G. Silveira, "On Intensity-Based Nonmetric Visual Servoing," IEEE Transactions on Robotics, vol. 30, issue 4, 2014, pp. 1019-1026.

[8] Jiang Hairong, B. S. Duerstock, J. P. Wachs, "A Machine Vision-Based Gestural Interface for People With Upper Extremity Physical Impairments," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 44, issue 5, 2014, pp. 630-641.

[9] D. Kruse, J. T. Wen, R. J. Radke, "A Sensor-Based Dual-Arm Tele-Robotic System," IEEE Transactions on Automation Science and Engineering, vol. 12, issue 1, 2015, pp. 4-18.

[10] N. M. Garcia-Aracil, J. M. Azorin, J. M. Sabater, C. Perez Vidal, R. Saltaren Pazmino, "Visual Control of robots with changes of visibility in image features," IEEE Latin America Transactions, (Revista IEEE America Latina), vol. 4, issue 1, 2006, pp. 27-33.

[11] D. Caldwell, A. Wardle, O. Kocak, M. Goodwin, "Telepresence feedback and input systems for a twin armed mobile robot," IEEE Robotics & Automation Magazine, vol. 3, issue 3, 1996, pp. 29-38.

[12] Z. Pezzementi, E. Plaku, C. Reyda, G. D. Hager, "Tactile-Object Recognition From Appearance Information," vol. 27, issue 3, 2011, pp. 473-487.

[13] Yiu Cheung Shiu, S. Ahmad, "Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX=XB," IEEE Transactions on Robotics and Automation, vol. 5, issue 1, 1989, pp. 16-29.

[14] H. Hadj-Abdelkader, Y. Mezouar, P. Martinet, F. Chaumette, "Catadioptric Visual Servoing From 3-D Straight Lines," IEEE Transactions on Robotics, vol. 24, issue 3, 2008, pp. 652-665.

[15] R. F. Wolffenbuttel, K. M. Mahmoud, Paul P. L. Regtien, "Compliant capacitive wrist sensor for use in industrial robots," IEEE Transactions on Instrumentation and Measurement, vol. 39, issue 6, 1990, pp. 991-997.

[16] Xiao Shunli, Li Yangmin, "Visual Servo Feedback Control of a Novel Large Working Range Micro Manipulation System for Microassembly," Journal of Microelectromechanical Systems, vol. 23, issue 1, 2014, pp. 181-190.

[17] Vladimir J. Lumelsky, E. Cheung, "Real-time collision avoidance in teleoperated whole-sensitive robot arm manipulators," IEEE Transactions on Systems, Man and Cybernetics, vol. 23, issue 1, 1993, pp. 194-203.

[18] Usama Iqbal, Abdul Samad, Zainab Nissa, Jamshed Iqbal, "Embedded Control System for Autarep - A Novel Autonomous Articulated Robotic Educational Platform," Tehnicki Vjesnik-Technical Gazette, vol. 21, issue 6, 2014, pp. 1255-1261.

[19] G. Silveira, "On intensity-based 3D visual servoing," Robotics and Autonomous Systems, vol. 62, issue 11, special edition, 2014, pp. 1636-1645.

[20] Jason M. Godlove, Erin O. Whaite, Aaron P. Batista, "Comparing temporal aspects of visual, tactile, and microstimulation feedback for motor control," Journal of Neural Engineering, vol. 11, issue 4, 2014.

[21] D. Font, T. Palleja, M. Tresanchez, D. Runcan, J. Moreno, D. Martinez, M. Teixido, J. Palacin, "A Proposal for Automatic Fruit Harvesting by Combining a Low Cost Stereovision Camera and a Robotic Arm," Sensors, vol. 14, issue 7, 2014, pp. 11557-11579.

[22] S. Manzoor, R. U. Islam, A. Khalid, A. Samad, J. Iqbal, "An open-source multi-DOF articulated robotic educational platform for autonomous object manipulation," Robotics and Computer-Integrated Manufacturing, vol. 30, issue 3, 2014, pp. 351-362.