Improving Computer Vision Algorithms Through Multi-threading and the Use of Different Programming Languages.

José Pablo Guerra, Luis Alberto Rivera *Member*, *IEEE*
Deparment of Electronics, Mecatronics and Biomedical Enginieering
Universidad del Valle de Guatemala
Guatemala, Guatemala
{gue16242, larivera}@uvg.edu.gt

***Abstract*—In this work, a computer vision algorithm was implemented in an efficient way using multithreading programming. This was done by analyzing and improving the algorithm previously developed in `C++` language, migrating it to the `Python` language. The validations were carried out through comparative tests to obtain parameters such as execution times, CPU usage, among others. This paramaters helped to determine in which points there were improvements, or if both algorithms had the same performance. In addition, helped to validate that the results of the implementations were similar.**

The main functionality of the tool is to recognize the position of agents (term commonly used in swarm robotics). The tool was combined with a testbed, which was assembled to test the calibration of the camera.

***Index Terms*—computer vision, multi-threading**

I. Introduction

Today, computational tools are of great help for different activities carried out in different areas, both in industry and in scientific branches. Among these decisions, we find computer vision, whose main use is based on solving problems or taking problems that require great computing power, through the analysis of images and the environment.

However, the tasks that these tools can perform may represent a high cost in computational resources. For this, multi-threaded programming can be used to improve performance. For example, by having multiple threads, it is possible to capture and process data more efficiently. This allows computers (and the different applications that can be made on them) to reach results more quickly and efficiently.

Therefore, the main objective of this work is to improve the computer vision algorithm developed for the Robotat test table. To achieve this, the most suitable object-oriented language was selected, as well as the development or improvement of the algorithms to make them computationally efficient through the use of multi-threaded programming.

For this work, a software tool previously developed for the Robotat table was used, originally designed in C++. With this base, the different algorithms that make up the tool were migrated to the Python language. The tool is able to calibrate the camera to fit the desired physical space and detect the pose of objects (in this case robots on the Robotat table). In addition, it allows to generate markers that serve as visual identifiers for the robots on the table.

II. Background

A. Computer Vision

This is a tool with the objective is to develop algorithms to search for information in photograph or videos, to get relevant data from them, and then use them in other applications or analyses [1]. For example, Francisco Moreno and Esmitt Ramirez developed an algorithm to be use in a SBC (Single-Board Computer) like a Raspberry PI. With this algorithm they use OpenCV [2] library to get information from photos and videos and used them in applications such as face detection and movements. [3]

B. Camera Calibration

The camera calibration is a useful tool to obtain information from an image. This is because it allows to adjust the perspective of the pictures and focus on the relevant information on it. Camara calibration is used on Virtual Reality, 3D reconstruction and other areas [4] [5].

C. Multi-threading programming

A thread is a control unit inside a process. Programs have a main thread and this may will in charge of creating new threads. This will help programs to execute different tasks at the same time, and make process more efficient [6].

There are some advantages in the use of multi-threading. First, you can obtain data with one thread and use another one to process it. This allows process to obtain results much faster, unlike if it is sequential and expects to first get data and then process it. If no threads are implemented, the process is strictly sequential and some task may block the continuity, so time to get results can get longer. Additionally, multi-threaded communication is much more efficient than inter-process communication. [6].

D. Related Work

For this job, was used the algorithm previously developed by André Rodas [7]. The main function of this software tool is camera calibration, that consist on adjust the image to focus on specific area, and also adjust the perspective to obtain good result on obtaining objects position on this areas. The algorithm allows to take pictures, process them and identify robots on the testbed.

Luis Antón Canalís [8] on his thesis named "Swarm Intelligence in Computer Vision: an Application to Object Tracking", describes how computer vision can be applied on swarm robotic and give direction to the agents. He describes how this agents form colonies (like ants or bees) and their are capable to go through a space and search for relevant information. On the computer vision case, this spaces can be images, testbed (like on this work) or physical spaces (like forest or big areas).

Also, in the multi-trheading area, Hiroshi Inoue y Toshio Nakatani propuse a comparation between a orientede model using multi-threading and other oriented to multiprocess in a SMT processor (Simultaneous Multi-Threading). When they use all the processor's cores (8 for this case) they obtain and improvement on efficenty. When they use multi-threading, the improvement is 3.4% (with a maximum of 9.2%) compared to the multi-procesos model. This is because multi-threading allows to use parallelism, and this is a big advantages to get better results [9].

III. Methodology

The first step was migrate the original algorithm from C++ to Python. An example is shown in Figure 1 after a successful calibration of the camera using Python algorithm, and the Figure 2 illustrated in a general form how the propose algorithm works. First, the algorithm capture a photo from the testbed and apply a calibration method that consist in the use of the Canny Edge detection. This Canny detection is to recognize contours and with this find the proper corners of the table (desired by the user).

After identify the 4 corners, the algorithm proceed to take another photo to search for the robots and get their positions and IDs. Using the Canny Edge detector, the algorithm is able to identify objects in the testbed, and if those objects have an image (a visual ID like Figure 4, it obtain the position (X-Y coordinates) and rotations angle and its respective numerical ID by decoding the image. This process repeat N-times, where N = Number of robots in the testbed.

The Figure 3 shows how the algorithm uses Canny Edge to detect objects on the image. This figure shows a preview before calibrate the photo in the 4 desire corners (the circles next to corners of the big square)

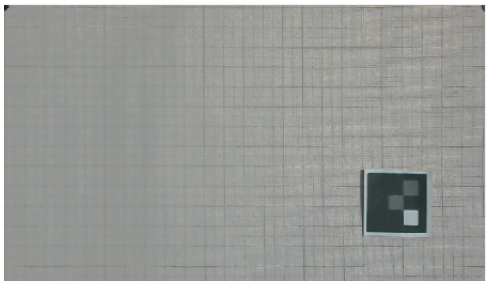

Fig. 1. Example of a calibrate photo to identify robots on it.

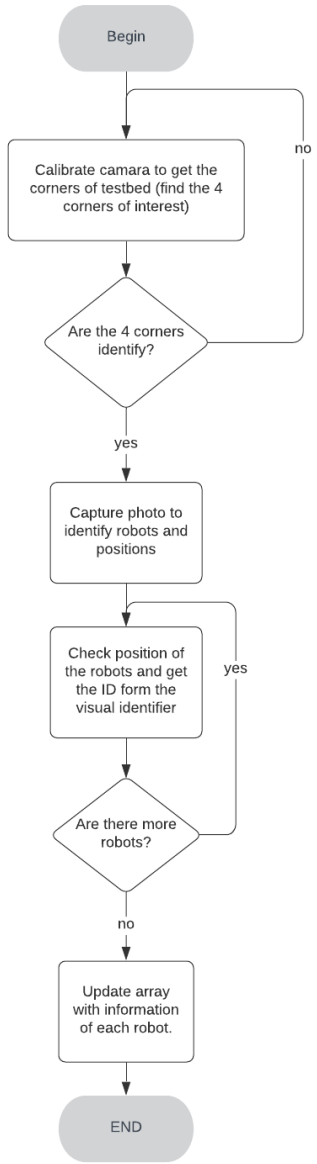

Fig. 2. Marcador reconocido en C++.

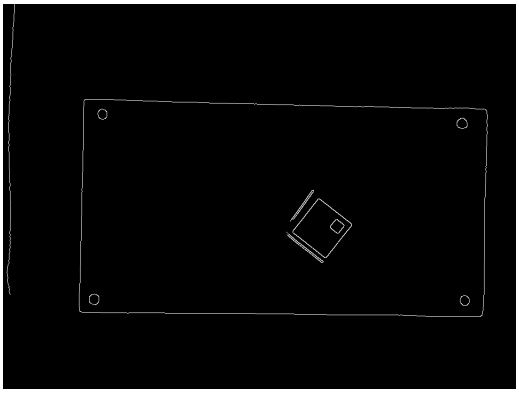

Fig. 3. Canny Edge detection example.

A. Generating the visual identifiers

The visual identifiers are images that the user may use to add and ID to every robot on the table. It is an image compound of 9 inside squares. The idea is generate a code (a visual code) that the algorithm will interpret as code (a decimal number) as Figure 4 illustrated. This image have three colors: white, gray and black. The white square is called pivot. This pivot is used as a reference to the algorithm. Because the robots could be rotated in any position, this pivot helps the algorithm to rotate the image until is located in the right upper corner.

The other two colors (gray and black) represent 1 or 0 in a binary codification. As mentioned, the image has 9 squares. The firs one is the white, the pivot. The next one represent the first position (2^0) in a 8 bits number, the next one is (2^1) so until the left bottom square that is (2^7).

For example in Figure 4, the ID is 40 (decimal number). His binary representation would be 00010100. As mentioned, the first square is the pivot. Then the next 3 squares are black, the first 0 of the binary code. Then a gray square that represent 1, then another black square, and another gray. Finally two black squares that represent the last two 0 in the binary code.

Another example is the Figure 5. In this case, the code is 170. In binary this code is 01010101. Same as the previous example, the first square is the pivot. Then it has a black square that represent 0, then a gray square that is 1, and so on until the end of the 8 squares.

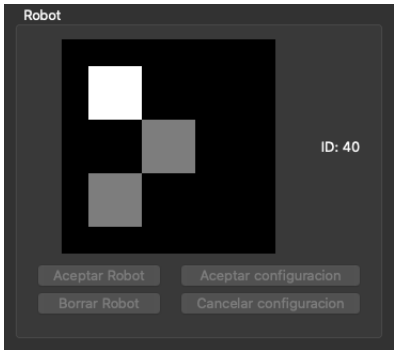

Fig. 4. Marcador reconocido en C++.

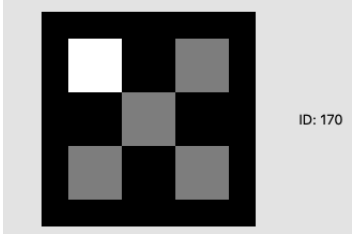

Fig. 5. ID representation for 170.

IV. Experiments and Results

A sets of experiments were carried out to validate the proposed framework. Firstly, it was validated the recognition of the robots position on the testbed. The second experiment was to run the algorithm using threads and compare execution times to to find the most efficient implementation

A. Experiment to validate position

The objective of this first experiment was to compare if the position detected by the algorithm was similar to the real position on the testbed. The algorithm was firstly implemented with an integer approximation. For this, was made 30 runs of the algorithm to obtain the position and compare with the real one, using both languages (C++ and Python). Then, the same 30 runs but without an integer approximation. This was done with the objective to check real precision of the algorithm. The Figure 6 illustrated the coordinates axes that were use in this experiment.

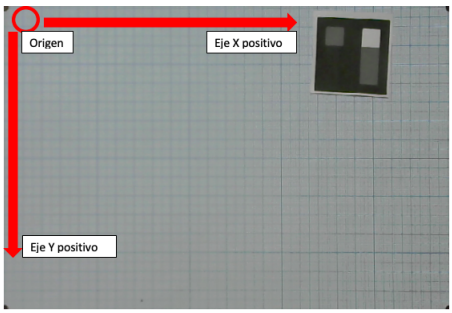

Fig. 6. Coordinated axes on the table.

Parameter	Mean	Mode	Estandar Desviation
Position (cm)	0	0	0
Angle (grades)	0.467	0	0.6399

TABLE I
MEAN AND MODE FOR POSITION WITH INTEGER APPROXIMATION BETWEEN PYTHON AND C ++ ALGORITHMS

Parameter	Mean	Mode	Estandar Desviation
Position (cm)	0.1157	0.085	0.0526
Angle (grades)	0.2	0	0.5606

TABLE II
MEAN AND MODE FOR THE POSITION WITHOUT INTEGER APPROXIMATION BETWEEN THE ALGORITHMS OF PYTHON AND C++

B. Experiment to find the most efficient implementation

After validate that the algorithm obtain the position correctly, an experiment was done to find the most efficient implementation. The original implementation (and their respective Python version) was done without threads. Then, a version with 2 threads on C++ was implemented, and a 4-thread version was implemented in Python too.

In order to have similar points of comparison, an implementation was made in Python using 2 threads, as in C++. Each of this experiments was run 45 times in similar scenarios.

Language	Mean	Mode	Estandar Desv.	T. Max(s)
Python	0.05513 s	0.085 s	0.02808	0.100
C++	0.01076 s	0.1070 s	0.0063	0.1388

TABLE III
TIME MEASUREMENTS FOR ALGORITHM WITH NO THREADS IN PYTHON AND C++

Language	Mean	Mode	Estandar Desv.	T. Max(s)
Python	0.2519 s	0.29 s	0.031	0.31
C++	0.1083 s	0.1073 s	0.00597	0.1402

TABLE IV
TIME MEASUREMENTS FOR ALGORITHMS USING 4 THREADS IN PYTHON AND 2 THREADS IN C++

Language	Mean	Mode	Estandar Desv.	T. Max(s)
Python	0.06358 s	0.047 s	0.02659	0.119

TABLE V
TIME MEASUREMENTS FOR ALGORITHM IN PYTHON USING 2 THREADS

V. DISCUSSION

The first comparison point was the position detection. The original implementation in C++ have integer approximation. The objective of this was to have a more realistic measurement of the position. Table I shows a summary of the measurements. Between Python and C++ algorithms the difference was 0 with integer approximation, with a 0.467 mean of difference on the rotation angle.

To have a better perspective of this difference, a version with no integer approximation was done. On Table II it shows this results. Between Python and C++, the position varies 0.1157 cm and the rotation angle 0.2 grades. Although now there is variation between the positions, it is still approximately a millimeter which means that there is practically no difference between position detected by the C++ algorithm and the Python algorithm.

On Table IV it shows the executions times between algorithm with a multi-thread implementation. As mentioned, multi-threading helps to do parallel task and that will impact execution time. The Python algorithm was firstly implemented with 2 processing threads, 1 threads to update the robot information and the main thread. The C++ version was implemented with 2 threads only. The objective of this fist implementation in Python was to be able to have modularity in the algorithm and to be able to divide large amounts of code into smaller parts.

Even so, as shown in the table, execution time of C++ version is better than the 4-thread Python version. This may be cause because overhead by passing trough threads, and also that they may be executing on the same core. Having this in mind, another version of the Python algorithm was developed. The results is shown in Table V. This version have 2 threads, same as the C++ version, and this implementation improves execution time.

Now, comparing this results with Table III (execution time with no threads) there is a significant difference. The execution time seems to be similar with thread that without them. This is probably because there is no real parallelism and all the threads are executing on the same core.

It is important to emphasize that, although the times do not show an improvement with more threads, the use of these helps a lot to the versatility and modularity of the code. This is because it allows it to be divided into smaller parts and each one of them performs a more specific task. Therefore, it helps a lot in the search for errors since it is easier to identify where a possible fault may be and to fix it, which is a great advantage in codes that are large. In addition, make the program not sequential, that means, that there is not waiting for a resource to be released or obtained to continue its execution.

VI. CONCLUSION

1) It was found that if the position measurements are worked in integers, statistically there is no difference between the version of C++ and the version of Python, but if it does not approach integers, the difference between algorithms is on average 1 millimeter. In both cases, accuracy and precision are denoted between the algorithms.
2) The execution time in the multi-threading implementation, in the Python language, is greater than its no thread version. This may be due to the number of threads implemented and other factors such as being processed in a single core.

REFERENCES

[1] Raúl López Briega. Visión por computadora. https://iaarbook.github.io/vision-por-computadora/.
[2] OpenCV. About. https://opencv.org/about/.
[3] Esmitt Ramirez and Francisco Moreno. Algoritmos de visión por computador para un sbc. 03 2017.
[4] Diego Aracena Pizarro, Pedro Campos, and Clésio Luis Tozzi. ComparaciÓn de tÉcnicas de calibraciÓn de cÁmaras digitales. *Universidad Tarapacá*, 13(1):58–67, 2005.
[5] Carlos Ricolfe Viala and Antonio José Sánchez Salmerón. Procedimiento completo para el calibrado de cÁmaras utilizando una plantilla plana. *Revista Iberoamericana de Automática e Información Industrial*, 5(1):93–101, 2008.
[6] Richard H Carver and Kuo-Chung Tai. *Modern multithreading: implementing, testing, and debugging multithreaded Java and C++/Pthreads/Win32 programs*. John Wiley & Sons, 2005.
[7] André Josué Rodas Hernández. *Desarrollo e implementación de algoritmo de visión por computador en una mesa de pruebas para la experimentación con micro-robots móviles en robótica de enjambre*. PhD thesis, Departamento de ingeniería electrónica, mecatrónica y biomédica, Universidad del Valle de Guatemala, 01 2019.
[8] Luis Antón Canalís. *Swarm Intelligence in Computer Vision: an Application to Object Tracking*. PhD thesis, Univesidad de las Palmas de Gran Canaria, 03 2010.
[9] H. Inoue and T. Nakatani. Performance of multi-process and multi-thread processing on multi-core smt processors. pages 1–10, 2010.