RECO-SOME: Driving with HSV

Paul Thai
College of Engineering – Computer Engineering California State Polytechnic University, Pomona California, US pthai@cpp.edu

William Tam
College of Engineering – Computer Engineering California State Polytechnic University, Pomona California, US wtam@cpp.edu

Huy Tang
College of Engineering – Computer Engineering California State Polytechnic University, Pomona California, US huytang@cpp.edu

Patrick Tejada
College of Engineering – Computer Engineering California State Polytechnic University, Pomona California, US pjtejada@cpp.edu

Mohamed El-Hadedy
Assistant Professor, ECE-department, College of Engineeering, College of Engineering – Computer Engineering California State Polytechnic University, Pomona California, US
mealy@cpp.edu

Abstract— **Automation is the future. Taking a simple task and automating it alleviates the need for humans to partake in the process which allows for better allocation of time elsewhere. There are three types of automation: Process, Integration, and AI. Process automation is handled by specially defined software and applications that are tasked with that process. Integration automation allows machines to mimic human tasks once their parameters are defined. AI automation is the process where the AI learns from its previous situation and uses the data to consider its next decision. The paper talks about implementing the use of process automation which is run by python software with the OpenCV library. The purpose is to test and see if data received from the camera input can be used to program four motors to rotate in a certain direction at a certain speed. Using HSV (Hue, Saturation, Value) the camera can detect a certain range of colors defined by the user, and depending on the parameter specified, a command is passed to the motors telling it how it should move. The paper takes the scope of self-driving cars to an extremely basic level – moving forward or reverse at a certain speed with varying colors. This will help determine how effective HSV is at detecting distinct colors and how it can be used in self-driving vehicles. The paper explains how different CPUs on different OS's will affect the performance of OpenCV by using a timer that will calculate how much time is needed for the program to run.**

Keywords—HSV, OpenCV, Process Automation, Integration Automation, AI Automation, CPU, Performance. *(Key words)*

I. Introduction

With the rise of synthetic media, the importance of image processing has become greater than ever. Image manipulation techniques like the popular deepfakes have proven the need to be able to use image processing to combat such techniques. Because of the immense amount of data, image processing can be difficult to optimize. The aim is to utilize a Raspberry Pi 4, in conjunction with OpenCV, to construct a program capable of color recognition and apply it to a simple RC car. By doing so the results can show how effective the team's image processing program is when used in a simple traffic light scenario. Although very rudimentary, it is possible to implement an abundance of diverse functionalities to the RC car simply based on the color the camera reads as an input. Using this color input, it can drastically affect various parts of the motorized car like speed and direction. Utilization of HSV - hue, saturation, and value, is a way to determine colors and use that data to produce our self-driving car [3]. Although the team applied the program to a basic color set, HSV can be accurate to a specific degree which means this program can be applied to various applications depending on the user's need. In addition to the HSV driving vehicle, the team can determine and compare the effectiveness of the Raspberry Pi 4 in comparison to other CPUs by timing how long it would take for the program to run the color detection portion of the code. [4]. Although simple, the task aims to prove the powerful capabilities of image processing

II. Related Work

There are related works of measuring multiple CPU performances and comparing their results to one another. However, other works compare the performances of compiler tasks, Geek bench performance, and video transcoding, while others correlate the performance of OpenCV using different libraries utilized by the researchers [1]. A research paper from IBM [4], compares the performance of disparate CPUs on a Linux operating system server with the number of cores and frequency rates specified. This research paper demonstrates the performance of four CPU performances running the same OpenCV script to enable a camera and identify the projected colors shown using HSV coloring. From the results, this paper is like in [4] because both record the number of cores and CPU frequencies to compare with other CPUs with a controlled variable, in IBM's case, is the Linux operating system server. Both papers also achieve the same goals by utilizing different CPU specifications to realize the complexity and usability of their variable.

In [1], it highlights the differences in performance between different OpenCV libraries, including Haar cascade and Dlib. These libraries objectively provide varying use functions for multiple scenarios to choose from, hence, performance differences naturally emerge. In this paper, it is like highlighting different libraries that have a constant variable, in this case, this variable is the script that is run within each CPU. As the variable is kept the same within each CPU, consequently, the group was able to find what variations there are between each of the processors in question.

III. Mathematical Model

The original goal of was to determine whether the Raspberry Pi 4 (Rpi4) would be able to run OpenCV on its processor. Then the scope of the paper extended further to the point where the group would test the same python code on different CPU to determine which CPU would be optimal for the program to run on. To determine how fast one CPU compares to another, the python code has an algorithm that will calculate how the program will run and then log the time into a predefined text file.

start = time.time() (1) [8]
time.time( )=*Month/Day/Year/Hour/Minute/Second* [8]

This variable *start* will hold the value at which the timer will go off. In the program, this start variable is placed right before the infinite while loop to since the goal is to start the timer right before the detection algorithm executes. The time.time() is a function in the python time library that is used to return the time in seconds with an addition of the current date.

diff = time.time() – start (2) [8]
time.time() – start = *final time – start time* [8]

The *diff* variable is responsible for storing the time that it took from the program to run the code color detection algorithm to when it stops. It is important that this variable *diff* vis placed right after loops breaks so it can immediately capture when the algorithm ends. By doing so, the time.time() returns the time at that point minus the time at that start which tells us how long the program took when it ran the color detection portion of the program.

IV. Performance and Evaluation

The image processing library, OpenCV, can use GPU-targeted functions. However, due to the scope of the paper, it will not be used. Then, it can be concluded that most of the processing load lies on the system's CPU. Therefore, benchmarking a system will also mainly reflect the performance of its CPU. Using the equations and model from the previous section, it was possible to conduct evaluations on the performance of the various systems. A testing set was developed using PowerPoint with timers that change slides with each having a colored circle. The testing program must detect each colored circle correctly and then finishes on the red circle which breaks the program after a few detections. For comparison, two Intel CPUs and an AMD CPU were used to benchmark as a reference to the Broadcom ARM processor on the Raspberry Pi. For each CPU, the testing data used the ten best points to eliminate outliers. This ensures that there is a greater degree of accuracy in the measurements.

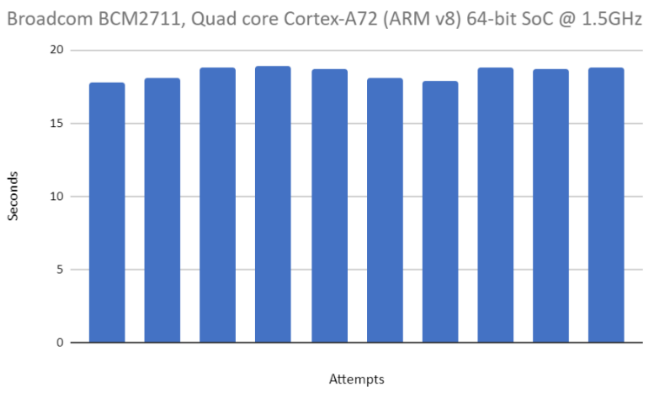

Fig. 1. Benchmarking 10 runs measuring program time on a Cortex A72 (Pi) processor

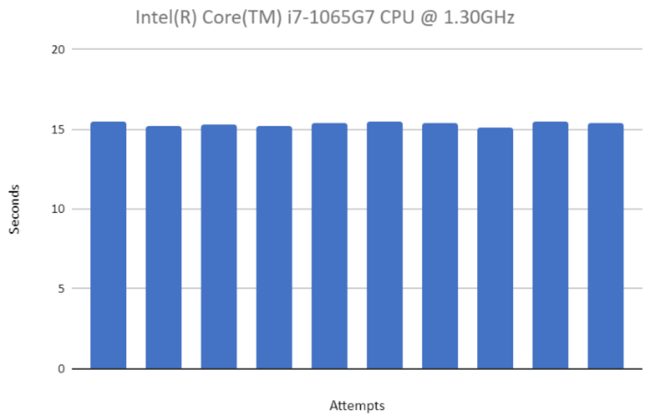

Fig. 2. Benchmarking 10 runs measuring program time on an i7 7065G7 processor

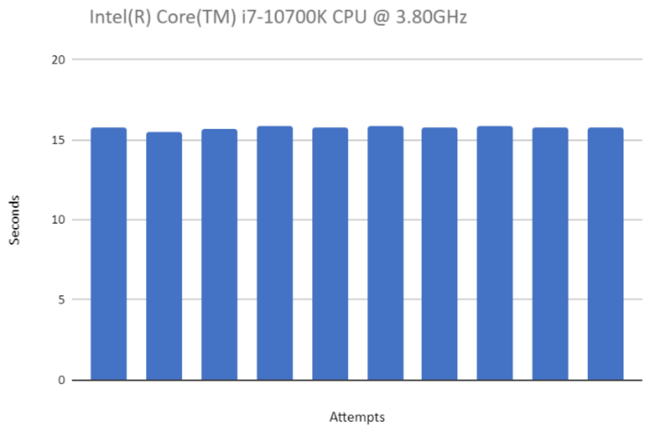

Fig. 3. Benchmarking 10 runs measuring program time on an i7 10700K processor

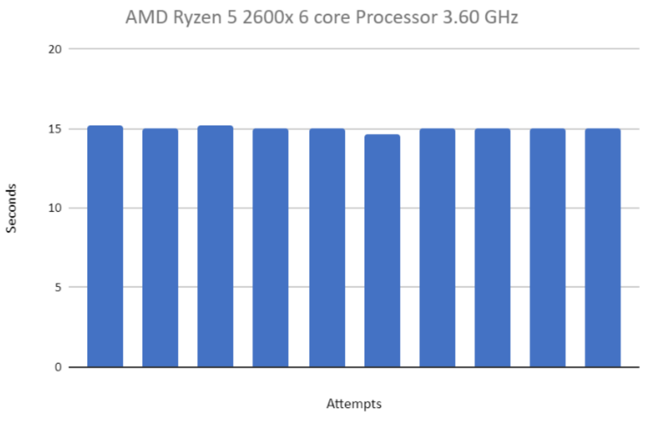

Fig. 4. Benchmarking 10 runs measuring program time on a Ryzen 5 2600X processor

The average time for the A72, i7 1065G7, i7 10700K, and 2600X was respectively 18.4293986797332 seconds, 15.335225629807 seconds, 15.7492772436142 seconds, and 15.0020660400390 seconds. These results are mostly within the expected range of values. The Raspberry Pi understandably performed the worst of the testing group. It was on average around 2 to 3 seconds slower than its competitors. The averages of the others are within a one-second margin which makes sense since the CPUs are from the newer generations from both companies--10th gen Intel and 2nd gen Ryzen. The numbers are understandably better than A72, however, common sense says higher clock frequency means better performance. The Ryzen 5 2600X has a lower program time than the i7 1065G7 which is expected since one is a desktop processor with higher clock frequency compared to a lower clock laptop processor. This would also imply a shorter program time of the i7 10700K compared to the i7 1065G7, which was not the case. Human errors during testing could be the source of this discrepancy. There are other factors that could also contribute to these numbers.

Although the main contributor to performance is the CPU, there are other components that can affect the benchmarks. One of the factors is the type of storage. The Pi operated on a micro-SD card which is much slower than modern storage options like a traditional hard drive or solid-state drive. Another factor will be the random-access memory in the system. The Pi 4B has 8GB capacity of RAM, which is on the lower end for modern computers, especially for more intensive programs. The RAM uses the LPDDR4 technology operating at 3200 MHz which is standard for modern devices, but the other testing systems could be operating at better frequencies and timings. The testing environment was varied as the Pi and i7 10700K were using Ubuntu 20.04, a Linux distribution, while the others used Windows 10. Some programs and libraries are better optimized on certain operating systems. The last potential variable for performance is the different webcams as different models can have different resolutions and frame rates. This can impact performance by increasing the processing needs for a single frame or increasing the number of frames to process.

Under more ideal situations, there are many ways to mitigate these pitfalls. For instance, modular testing systems could have been used so the CPUs could be swapped out easily while maintaining the other components. These other components should be like the specs of the Pi for a better comparison. Additionally, benchmark testing can be done on all Windows CPUs and then repeat the same for Linux for an accurate comparison between operating systems. One standard webcam model should be used to reduce any differences in how images are processed. Lastly, one person could do all the testing to keep human errors from methodology consistent. It could be possible to eliminate sources of human error by completely automating the process of starting PowerPoint and testing at a controlled time. This would also enable the collection of more data points for accuracy.

V. Conclusion

Information regarding the OpenCV library was documented and how it was used for image-processing purposes. Color-detection was the focus, and the subsequent program was designed to detect primarily four colors: red, yellow, green, and purple. This program is testing and applying functions from the library to control a basic motorized vehicle that behaves depending on the detected color. A basic test program was also designed to perform a basic benchmark and measure the performance of the various CPUs. Measuring the time of this program, evaluations could be made about the overall computing power of different processors on different OS. Some pitfalls were also identified with the testing methodologies and ways to improve them.

References

[1] Nataliya Boyko; Oleg Basystiuk; Nataliya Shakhovska, “Performance Evaluation and Comparison of Software for Face Recognition, Based on Dlib and Opencv Library,” October 2018

[2] Administrator, “Raspberry Pi L298n interface tutorial: Control a DC motor with l298n and Raspberry Pi,” Electronics Hub, 12-Feb-2018. [Online]. Available: https://www.electronicshub.org/raspberry-pi-l298n-interface-tutorial-control-dc-motor-l298n-raspberry-pi/. [Accessed: 23-Nov-2021].

[3] “Multiple color detection in real-time using python-opencv,” GeeksforGeeks, 10-May-2020. [Online]. Available: https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/. [Accessed: 23-Nov-2021].

[4] N. Congleton, “How to benchmark your linux system,” Linux Tutorials - Learn Linux Configuration, 29-May-2020. [Online]. Available: https://linuxconfig.org/how-to-benchmark-your-linux-system. [Accessed: 23-Nov-2021].

[5] Gus, “How to install visual studio code for the raspberry pi,” Pi My Life Up, 13-Oct-2020. [Online]. Available: https://pimylifeup.com/raspberry-pi-visual-studio-code/. [Accessed: 23-Nov-2021].

[6] Python Awesome, “A high-precision CPU and memory profiler for Python,” Python Awesome, 10-Jan-2020. [Online]. Available: https://pythonawesome.com/a-high-precision-cpu-and-memory-profiler-for-python/. [Accessed: 23-Nov-2021].

[7] D. A. Patterson and J. L. Hennessy, Computer Organization and Design: The hardware/software interface. Cambridge, MA: Morgan Kaufmann Publishers, an imprint of Elsevier, 2021.

[8] “Time - Time Access and conversions¶,” *time - Time access and conversions - Python 3.10.0 documentation*. [Online]. Available: https://docs.python.org/3/library/time.html. [Accessed: 24-Nov-2021].