Performance Evaluation of Basic Image Processing Algorithms in CPU, GPU, Raspberry Pi and FPGA

TAHAR ABBES Mounir[1], Selma Boumerdassi[2], Abdelhak Benhamada[1],
Abdelmadjid Mhamed Allal[1], Mohamed Kherarba[1]
[1] Hassiba Ben Bouali University, Computer Science Department, LME Chlef, Algeria
mtaharabbes@yahoo.fr
[2] Selma Boumerdassi, CNAM/CEDRIC, Paris, France.

Abstract— This article presents a comparative and experimental study (benchmark) regarding the main execution time on diverse platforms (CPU, Graphics Processing Units (GPU), Raspberry, and The Field-Programmable Gate Arrays (FPGA)). We evaluate the time performance of standard image processing algorithms used in many vision applications. To carry out this experiment, we designed a mobile explorer robot equipped with an integrated CMOS camera. This robot has been programmed to perform a route avoiding obstacles using five image processing algorithms (re-size, erode, dilate, find contours, and distance calculation). Two levels of tests were carried out: The internal level check the efficiency of the application algorithms, and the external level measures the platform's output concerning execution time. Our results highlight the following remarks: GPU is advantageous in image processing algorithms, which process data or pixels independently. The CPU shows its power on sequential data; however, GPU is slower than CPU in those algorithms. The FPGA card's performance is ten times higher than CPU and GPU; it is possible to increase performance by processing in parallel.

Keywords - Image Processing; FPGA; CPU; GPU; Raspberry Pi. (key words)

I. INTRODUCTION (HEADING 1)

Evaluating the performance of a system is essential to know under what conditions it can withstand and continue to operate without failures or faults within a determined period. This has been a topic of interest in science computer engineering for nearly half a century. Let us illustrate the problem with a simple example; if we want to implement an industrial real-time machine control system, several choices of the execution platform are available (CPU, GPU, STM32, PIC, ATMEGA ...). So we must have a prior idea on the performance of these platforms in terms of execution time, memory, and energy consumption, to be able to make an optimal decision. Our work will help engineers and technicians to make optimal choices depending on the application context, for this we have chosen to implement five standard algorithms for image processing in different platforms. The chosen application falls within the field of robotics, it consists of building an autonomous robot which traverses a terrain with obstacles without human intervention.

Improving the degree of autonomy of robotic systems is of interest to researchers and manufacturers. In this system, robots are designed to work without human intervention in often hostile environments and long periods. To reduce power consumption and CPU time, the robot tasks performed using algorithms with low complexity. One of the most used applications for robots is the autonomous navigation system. Autonomous navigation systems are used in applications such as service, monitoring, or exploration robots in which the robot simultaneously moves and perform the main task. Mobility is one of the main aspects of these robots as it is the basis on which one can combine many subsystems with different functionalities. However, motion system performance has a significant effect on task performance.

Related problems occur in applications that could have fatal consequences (e.g., robots carrying dangerous material). Mobility depends on the environmental parameters, which include sound and image. Image segmentation is the most useful image processing system process. The segmentation interest is to decompose an image into several homogeneous regions, in the sense of a criterion that is a priori set. Having homogeneous regions can offer more data that simplifies the task. Segmentation techniques are based on the variation of pixel values (edge detection) or detection of homogeneous areas in the image (extraction of regions). Region segmentation approaches work by partitioning the image into a set of regions. Mobile elements (robots) faced several problems that could be solved using different techniques. The problem of obstacle avoidance is one of the significant issues in mobile robotics.

A classic problem in this domain is to make avoidance of obstacles and the planning of trajectories for autonomous mobile robots. It defined as follows: if we know the first place of the mobile node in a known or unknown environment, that can surely contain static or dynamic obstacles, and if we have collected all the required pieces of information about the characteristics of this environment by the sensor system (camera) of the robot, how we can try if we want to move this mobile robot to a new position? In our case, we only deal with dynamic obstacles by assuming that information from the robot's environment where gathered from a camera and processed using image processing methods in a processing environment. This paper analyzes each device's trends in execution time performance. It compares their sustained performance sets of scientific applications seeking to find the right platform for a specific application. Recent literature, comparing Graphics Processing Units and CPU are usually restricted to a special set of applications in a particular domain, or only compare two of those platforms. On the other hand, FPGA (Field-Programmable Gate Arrays) has illustrated great performance in image processing applications. However, recent CPU and GPU also have a potential for high performance for those problems. We have to choose a domain that calls algorithms using all the resources available on the platform. We opted for image processing, and the subject that affects several and very complex algorithms is the avoidance moving obstacle. We aim to develop a dynamic algorithm for avoiding obstacles based on image processing techniques in different platforms (CPU, GPU, Raspberry, and FPGA).

The following steps are used in Algorithm:

1. Capturing the image
2. Detecting and identifying obstacles.
3. Distance measurement between camera and obstacles.
4. Decide for avoidance.
5. Then test the Algorithm into various platforms: CPU, GPU, Raspberry, and FPGA.

The document is structured as follows: Section two is dedicated to literature survey. Section three is devoted to the formulation problem and the proposed method, while Sections 5 presents evaluation performances and results. Section 6 concludes the study.

II. RELATED WORKS

This section focuses on the classification and discusses related studies into four parts (CPU, GPU, Raspberry Pi, and FPGA). The literature is reviewed to examine available methods that could be used to evaluate the time consumed, and study the efficiency of FPGA, CPU, and GPU with different configurations. These have been important topics of study in the literature for many years because of the constant evolution of those platforms. A number of studies have examined the impact of the hardware on the chosen solution.

The first study [1] investigated offloading of the processing part of a latency educational game to a low-cost Raspberry and GPU. The work [2] proposes a theoretical algorithm for a characterization of operations performance in an important class of applications on GPUs and Intel CPU, and GPUs from AMD and NVIDIA. Authors in [2] show that GPUs are faster and more energy-efficient than a CPU or Xeon Phi processor.

Conceptually similar work has also been carried out by [3] in which, they analyzed and focus on the specific application associating moving area separation, over images that initiated in big multimedia operations. They offer a novel Histogram-based Moving Object Segmentation algorithm that implements a pixel-oriented approach. It's providing higher performance on both quality and efficiency. Another important constraint on all the work discussed is:

- Studying how fragmentation techniques can be integrated to improve the effectiveness of this framework.

A concept has been presented in [4] to discover the possibility of implementing game theory decision making to get a win-win between autonomous vehicles. The results validate that the proposed accelerator and tests the performance by implementing the design on a Xilinx KCU116 board. The accelerator's initial speed-up is 2.4x versus performance on a CPU. Along with the same context, a technique has been proposed for designing applications for real-time image processing [5]. The authors Use the ITER CODAC Core System (CCS) software tools and the hardware configuration specified in the ITER Hardware Catalog of Fast Controllers, Camera Link FPGA-based frame-grabbers, and NVIDIA GPUs. This interface support provides complete control of camera configuration parameters, delivery of image processing functions between the FPGA and the GPU, and efficient data transfer between the various architectural components. Consequently, the GPU showed better performance than the CPU.

Likewise, [6] proposed that the GPU application is laughable, as they cannot perform the calculations with sufficient speed. The authors illustrate that FPGAs and GPUs are commonly used in computer vision applications and image processing between hardware accelerators. A detailed comparison of the FPGAs and GPU performance was thus given.

An easily reproducible benchmarking method was evaluated in [7] that use only publicly available vision libraries: OpenCV, NVidia Vision Works, and if OpenCV unique platform-specific code. The results show that the GPU achieves an energy/frame reduction ratio of 125 1.1{3.2 versus CPU and FPGA for simple kernels. However, the FPGA outperforms others with energy/frame reduction ratios. It is also observed that as the complexity of a vision kernel increases, the FPGA performs progressively better.

More recently, the author's work in [8] seeks to help a developer or company understand the trade-offs in using heterogeneous computing systems provided by cloud computing providers to deploy high-performance computing applications. Knowing the cost/performance and scalability of implementation at CPUs, FPGA, and GPUs are essential. To reduce costs, one may choose the right deployment strategy. The authors show that FPGAs can provide substantial speed-up compared to a GPU on the performance-critical kernels.

Along these, research supporting multiple encryption algorithms for password recovery based on hybrid CPU-FPGA systems that can benefit from both the versatility of the CPU and the energy efficiency of the FPGA [9]. The proposed hardware accelerator architecture is found to be 12.5 and 3.1 times more 140 power effective than the TrueCrypt and WPA-2, respectively, pure FPGA-based password recovery accelerators.

The contributions in [10] can be summed up as follows: Firstly, a heterogeneous architecture of computing systems is built, consisting of computing nodes with a local queue. It proposes the parallel application job model based on the execution size of the CPU and GPU. Secondly, a model for using a system computing node CPU GPU is specified. It analyzes the relationship between computing node power consumption and CPU-GPU utilization, and a deduction of job execution energy consumption and optimal CPU-GPU utilization.

The research study in [11] examines the extent to which GPU has been used for the processing of radar signals and radar data. Several studies have used GPU for GPU implementation of radar signal and data processing algorithms compared to the usual Central Processing Unit (CPU). The result of the comparison shows that the performance of the GPU is much better than the CPU.

III. Architecture of the Proposed solution

In order to evaluate the five processing image algorithms in various platforms CPU, GPU, Raspberry, and FPGA we implemented on our robot (contour detection, segmentation, erosion, dilate and distance calculation) which allow the avoidance of dynamic obstacles in a specified area.

As shown in figure 1, the proposed system is divided into two parts:

- Hardware part: contains the electronic components used.
- Software part: Represents the used algorithms to perform all the tasks (information processing).

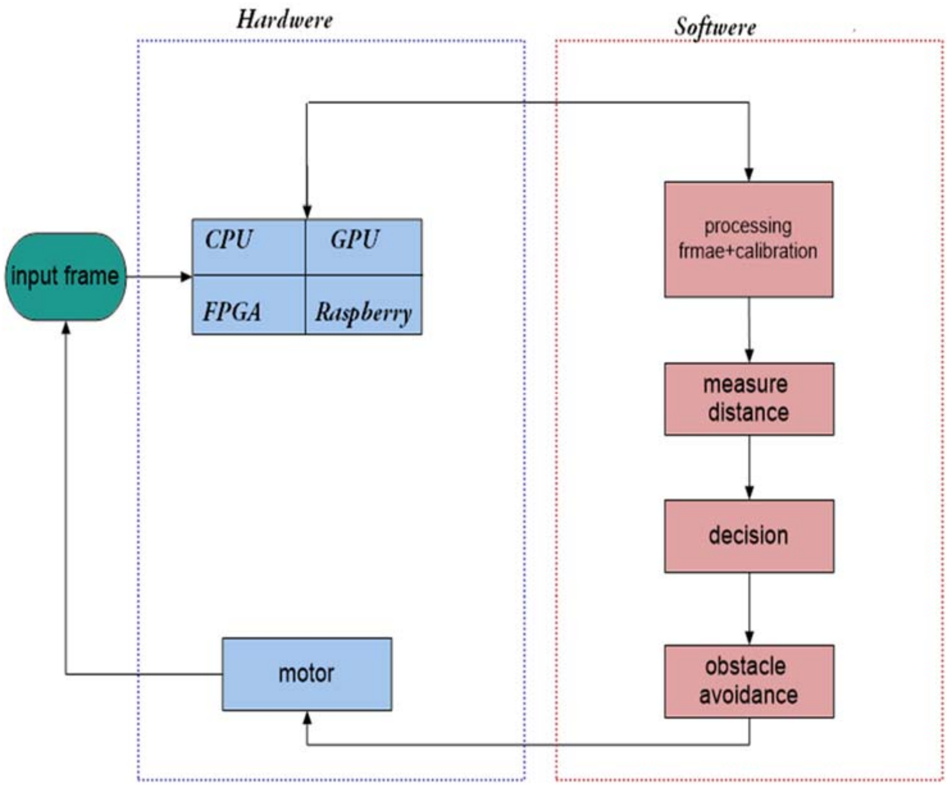


Figure 1. The architecture of the proposed solution.

We have implemented an image processing algorithms (Calibration, Contour detection, Distance measurement) integrated into a development platform, the latter communicates with a camera which makes it possible to capture frames and send them as digital images, each image is processed using an algorithm. At the end, the processing results obtained will be used to make a decision (check the motors) as illustrated in figure 2 the system overview is presented in figure where:

d: the distance measured between the obstacle and the camera.

Threshold 1: specify for the initial state whose distance is very close to an object to avoid.

Threshold 2: the ideal distance to avoid the obstacle by the decision "turn right" or "turn left".

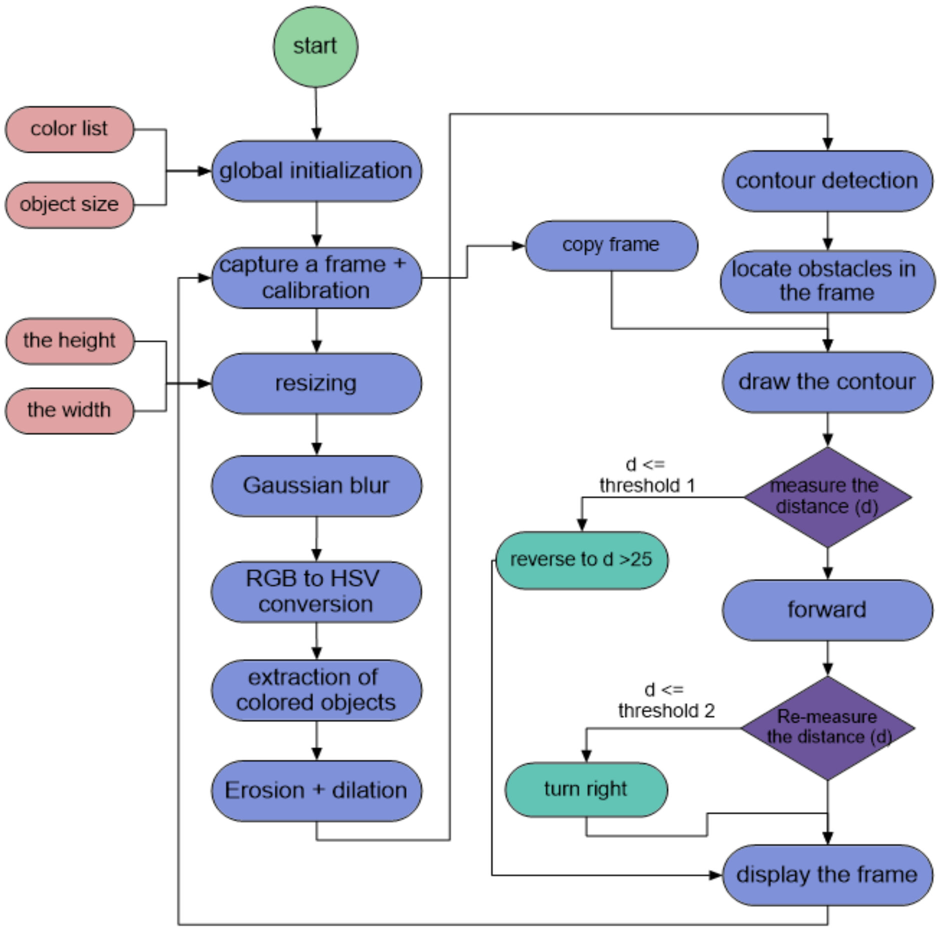


Figure 2. .The diagram of proposed system.

Several standard image processing algorithms have been used in our system; these have been modified so that they are less complex. Among these algorithms is: Re-seize which will be the first processing to be applied in the image in order to minimize the number of pixels to be processed.

A. The resizing algorithm

During the resizing process, keep in mind the original picture size, aspect ratio, and proportional relationship of the image's width and height. If we ignore these last elements, our resizing will give unsatisfactory results. This step (resizing) objective minimizes the computation time and the memory space of the processing platform.

B. Gaussian Blur

Gaussian Blur [12] Filtering is a treatment that applies globally to the entire image. For each pixel in the image, the filter calculates its new value taking into account the pixel's vicinity. Therefore, we attach ourselves to filters intended for improving the image (noising, smoothing, etc.).

A filter is therefore characterized by:

- The shape of the neighborhood (usually a square centered on the pixel).
- The size (or radius) of the neighborhood.
- The algorithm for calculating the final value.

Blur filters [12] consist of modifying pixel value so that it approaches neighboring pixel values. Therefore, the differences between neighboring pixels are reduced; noise, contours, and details are reduced, so the image is "smoothed." The blur is usually due to a convolution of the original image by a kernel:

- x : image source.
- u(x) : pixel value (without filter).
- j(x) : kernel.
- w(x) : pixel value (with filter).

$$w(u) = ju = uj = \int_R j(x-y)\,u(y)dy. \qquad (1)$$

The advantage of a Gaussian filter is to minimize the loss of information while reducing overall noise.

C. RGB to HSV

The TSL or HSV [13] model (Hue Saturation Value) is a color management system representing color not by its classic red, green and blue representation, such as for display on the screen, but defined by the hue saturation and luminosity offering a more perceptual visualization of the color. The principle of this method is as follows:\\

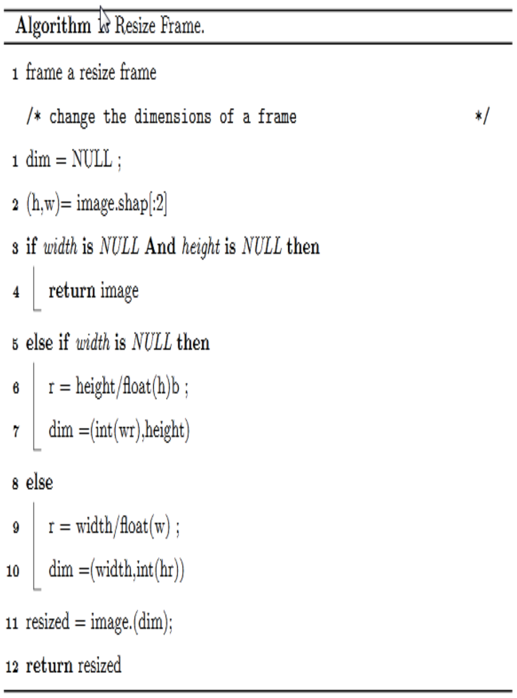
for each pixel of the image, if its HSV value is not included between the lower and upper limit, then it becomes 0, otherwise we will not change.

D. Erode and Dilate

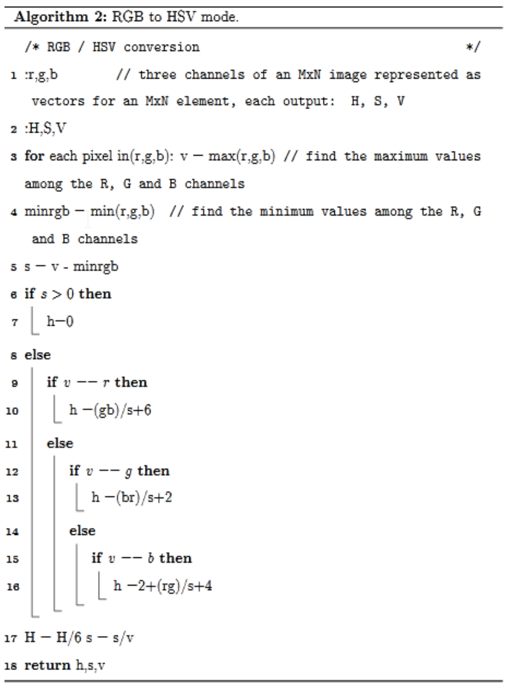
Dilation [14] is a basic morphological operation. It's expanding the image pixels. The successive application of erosion and dilation produces the elimination of details smaller than the structuring element without there being a significant distortion of the characteristics that have been retained. By using gray scale images, dilation increases the brightness of objects, and with binary images, it also connects the separated areas by space smaller than the structuring element. Erosion [14] is the dual function of dilation; it's also one of the fundamental morphological operations. It usually used for probing and reducing the shapes contained in the input image. It decreases the size of the objects and removes the small anomalies.

E. Contour Detection

As described in [15], a contour is defined as an outline representing or bounding the shape or form of an object. Contour detection attempts to extract curves which represent object shapes from images. In fact, the concept of contour is based on human's common experience, which does not have a formal mathematical definition. Contour is closely related to two additional concepts, i.e., edge [16] and boundary [17]. Another consideration is to introduce prior shape information about the contours, so that so as to improve contours will be detected. This is important for some usefully tasks such as tracking objects. The contour detection algorithm must be able to reject unwanted information for the contours undesired and detect objects even in the presence of noise and occlusion. A related approach is presented in [18], which aims to detect semantic contours. Moreover, the accuracy in this method is not satisfactory, which presents a particularly attractive area [19].



```
Algorithm 1: Resize Frame.
1  frame a resize frame
   /* change the dimensions of a frame                    */
1  dim = NULL ;
2  (h,w)= image.shap[:2]
3  if width is NULL And height is NULL then
4      return image
5  else if width is NULL then
6      r = height/float(h)b ;
7      dim =(int(wr),height)
8  else
9      r = width/float(w) ;
10     dim =(width,int(hr))
11 resized = image.(dim);
12 return resized
```



```
Algorithm 2: RGB to HSV mode.
   /* RGB / HSV conversion                                  */
1  :r,g,b      // three channels of an MxN image represented as
    vectors for an MxN element, each output:  H, S, V
2  :H,S,V
3  for each pixel in(r,g,b): v − max(r,g,b) // find the maximum values
   among the R, G and B channels
4  minrgb − min(r,g,b)  // find the minimum values among the R, G
    and B channels
5  s − v - minrgb
6  if s > 0 then
7      h−0
8  else
9      if v −− r then
10         h −(gb)/s+6
11     else
12         if v −− g then
13             h −(br)/s+2
14         else
15             if v −− b then
16                 h −2+(rg)/s+4
17 H − H/6 s − s/v
18 return h,s,v
```

F. Camera calibration

Camera calibration is one of the important problems in image processing algorithm because many applications need the exact calculation of metric information from the input image. Calibrating a camera consists of determining the transformation that projects the 3D points of a scene to their 2D image plane correspondences [20].This transformation depends on two types of parameters: external parameters which define the pose of the camera and the internal parameters that describe the internal geometry of the camera, that is, the process of building an image through the optical system. We used Zhang's [20] method to calibrate the used camera in our application. The calibration steps as detailed by Zhang is as follow:

- Print an example pattern and attach it to a planar surface.
- Read a set of images for calibration.
- Process the image for corner detection.
- Calculation of the internal and external matrix;
- Refine all parameters.

IV. IMPLEMENTATION AND RESULTS

The previous sections presented the adopted solution, the obstacle avoidance algorithm's design, and the used methods. This section describes the implementation of the algorithms using the python language (version 3.8) and the Open Source Computer Vision Library (OpenCV) in the following platforms:

- Raspberry Pi 3 B +.
- GPU
- FPGA
- CPU

The input video is the same in GPU, CPU, FPGA, and Raspberry Pi 3 B + captured from the camera used in Raspberry to ensure that all the platforms use similar video frames.

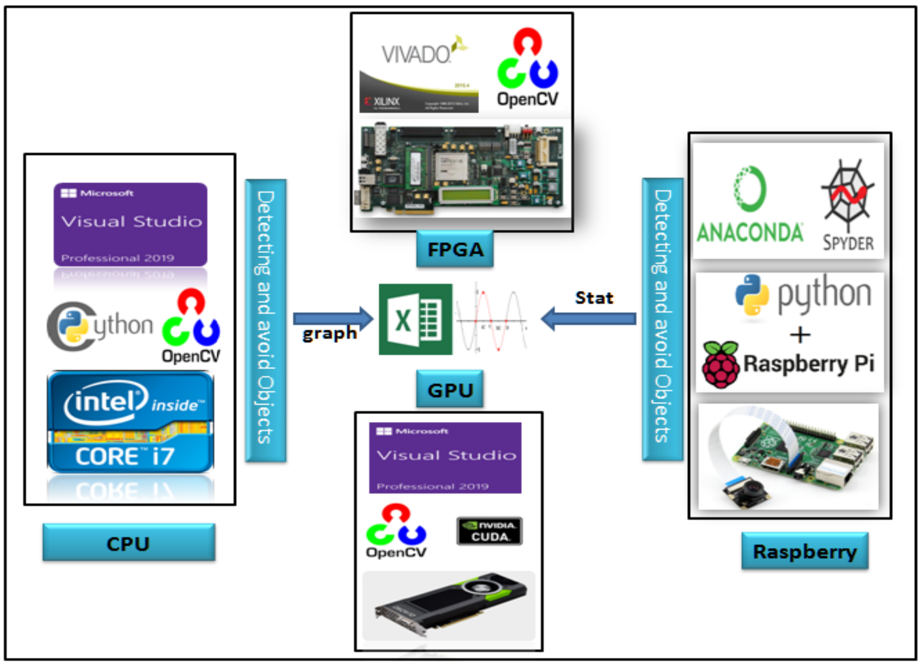


Figure 3. Architecture of the solution.

OpenCV is a Library Officially launched in 1999. The OpenCV project was initially developed by Intel to optimize applications consuming a lot of processor time. It was part of a series of projects, such as displaying a 3-dimensional object [21]. This library is distributed under the BSD license. The main contributors to the project are the library development team at Intel and several optimization experts. This library is written in C and C ++ and can be used on Linux, Windows and Mac OS X. The OpenCV library provides many diversified functionality allowing creating programs starting from raw data to go as far as the creation of basic graphical interfaces. It offers most of the classic operations in low-level processing of images and videos [22].

A. Implementation in RASPBERRY and CPU

Officially released on February 29, 2012, the Raspberry Pi is developed by the Raspberry Pi Foundation. It's a computer that has the size of a credit card. The original purpose of the Raspberry Pi was to provide a low-cost tool for encouraging programming. We use the Raspberry card to build a mobile robot field explorer, which will avoid obstacles inside this field; in this way all the algorithms proposed in section 3 will be used so intensive.

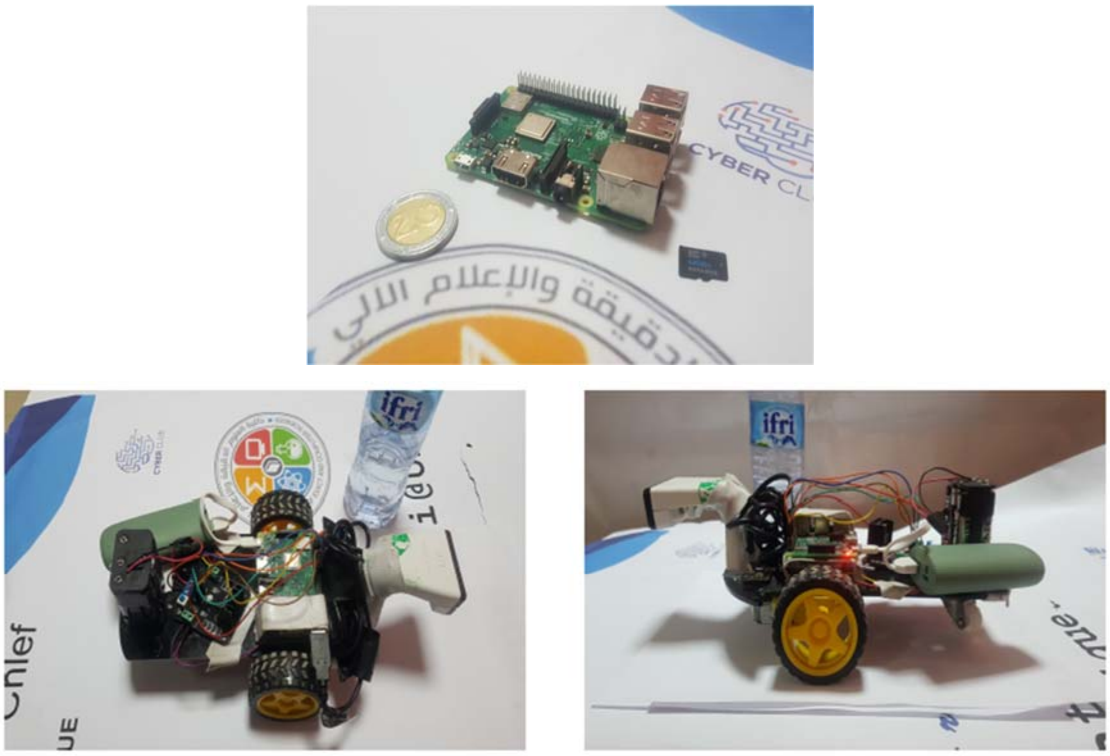


Figure 4. Rasbery board and the mobile robot.

During the experiment we collect the statistics and after that we do the evaluation

In order to evaluate the performance of the proposed solution we have conducted many real world experiments in a given area (figure 4). In all experiments we placed objects with different size at approximately. In the first experiment the mobile robot covers an area of approximately 245cm * 160cm. The duration of each of the experiments was approximately 4 minutes.

A mobile robot is a mechanical, electronic and computer system that acts physically on its environment in order to achieve a goal that has been assigned to it. This machine is versatile and able to adapt to certain variations. It has functions of perception, decision and action. Thus, the robot should be able to perform various tasks, in several ways, and perform its task correctly, even if it encounters unexpected new situations. A robot is said to be autonomous:

Our robot is composed with the following components:

- DC motor
- L298 shield
- Battery 12v for motors, 3, 7 for raspberry
- Camera

Those components are interconnected as indicated in the following figure.

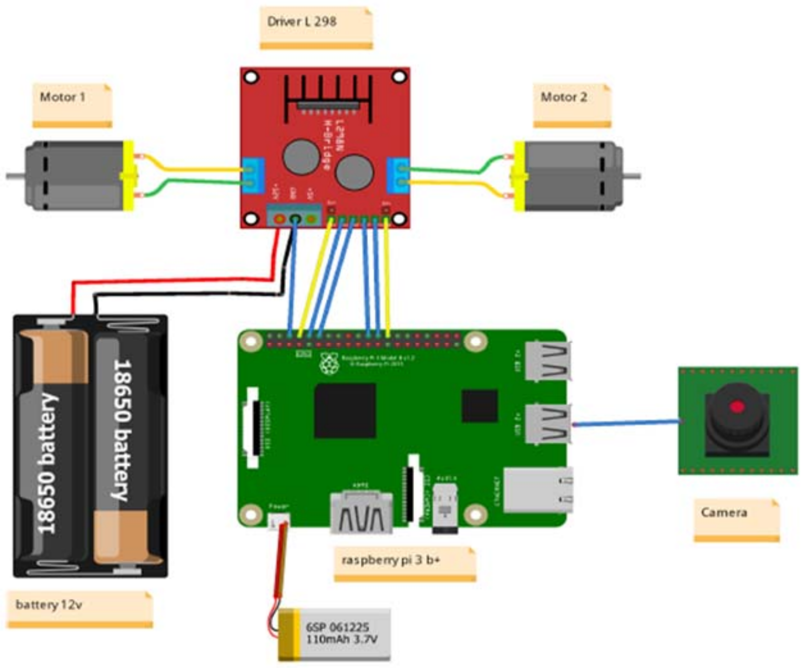


Figure 5. Logical connection of mobile robot.

After having carried out the calibration, we begin to calculate the distances separating the robot and the objects encountered according to the algorithm presented previously, as illustrated in figure 6.

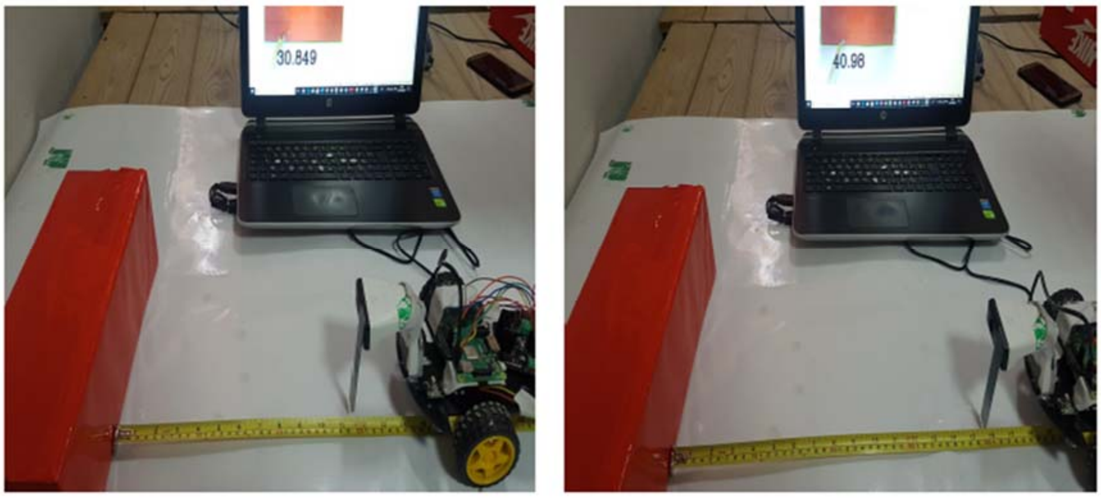
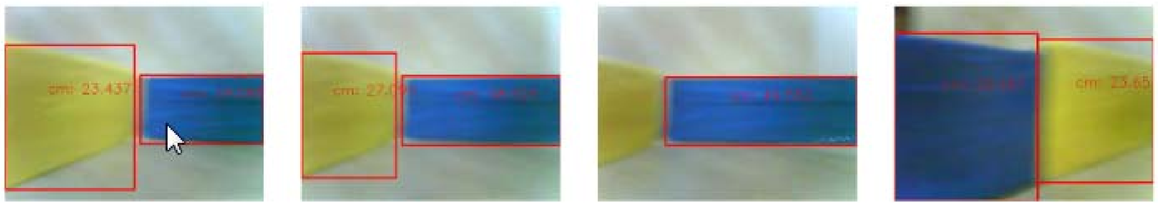


Figure 6. distance calculation.

After checking all parameters and before starting the experimentation we should calibrate our camera, according to Zhang method.



(a) Detecting Objects (b) Calculating distance (c) Taking decision avoid (d) Detecting Objects

Figure 7. Frames during experimentations.

For this experiment, we fixed the distance over which the robot must avoid the obstacle at 25 Cm and the evaluation time is set to 4 minutes, after observation (figure 8) may have said that the robot avoids all obstacles until reaching 55 % of the engine speed, at this point we start to have collisions because the Raspberry CPU is limited in a clock frequency as well as the size of the memory, to process the frames in real time.

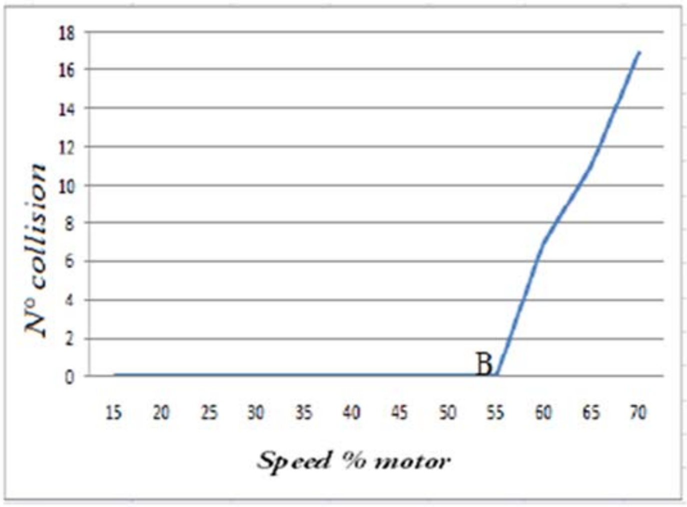


Figure 8. Speed Vs Number of collisions.

After finalizing all parameters we compare and evaluate the fundamental functions (InRange, Gaussian Blur, RGB to HSV and Contour detection) in the chosen platform (CPU and Raspberry), and after that we make a deep comparison between them.

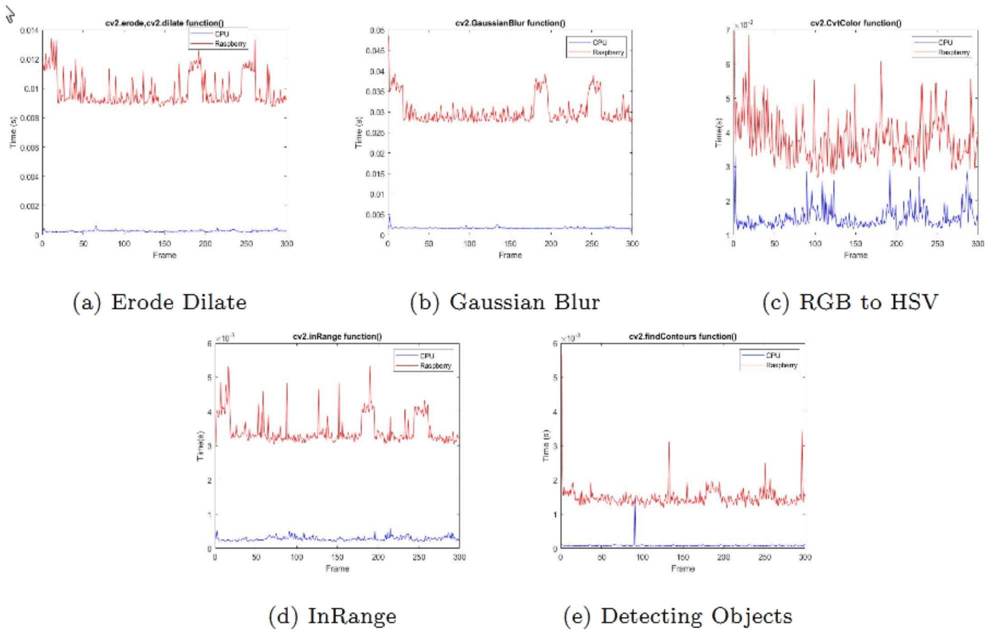


Figure 9. CPU VS Raspberry.

All these steps are carried out in the same way in the CPU platform with the following technical characteristics:

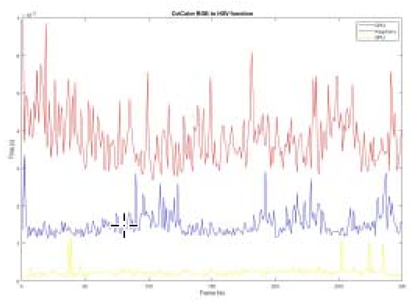
- Intel(R) Core(TM) i5-4210 U CPU@1.70GHz 2.40 GHz
- (RAM):6 Go
- Windows 10.

According to Figure 9 (a, b), processing time using Raspberry pi 3 ranges from 0.009s to 0.013s for Gaussian Blur and erode and dilate functions. Compared to CPU range from 0.0009s to 0.001s, CPU takes less time and has higher stability. And regarding Figure 9 (c), CPU takes less time and has lower stability, this due to variety of pixel nature in our frames, and also for the limited memory in Raspberry board.

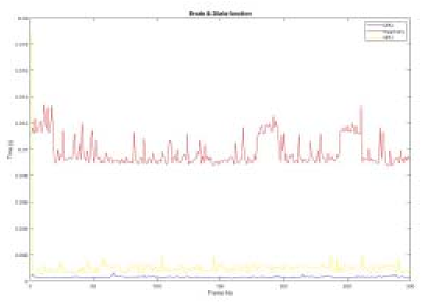
B. *Implementation on GPU*

Graphics Processing Units (GPUs) are increasingly being adopted by real time vision application due to their immense processing power [23]. In order to make the evaluations of the previous algorithms on GPU, we used Opencv: 4.3.0, with the NVIDIA Quadro p5000 graphics card and CUDA Toolkit: 10.1.

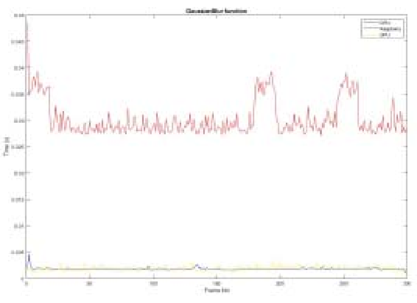
CUDA is a higher level interface, provides a C-like syntax for achieving on the GPU and compiles offline. CUDA exposes two levels of parallelism, data parallel and multi-threading[24]. The development environment used is Microsoft Visual studio 2019 with C ++.



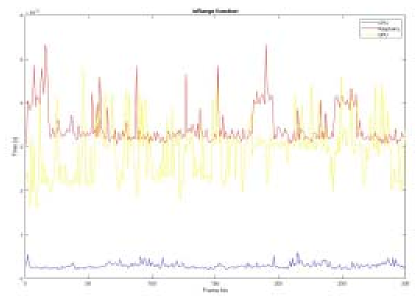
(a) RGB to HSV



(b) Erode and Dilate



(c) Gaussian BLur



(d) InRange

Figure 10. GPU VS CPU VS Raspberry.

Graphs for "Time Vs. Frame input" is shown in figure 10, in which the lines with yellow, blue, and red colors are used to specify the GPU, CPU, and raspberry, respectively. We can see a fluctuation and high variation in GPU and raspberry graphs because of the underlying hardware. Up to a certain amount of memory, the processor can fetch in one clock cycle. In Gaussian and erode operations, both the CPU and GPU are fighting for performance for the most part; this is because the combination of throughput and latency of both devices are similar. In conversion and re-seize operation, CPU performs better than GPU; this is because of these devices' architecture. For intensive processing and manipulating data, the GPU significantly improves the execution time only on the filter system (figure 10.a).

C. Implementation on FPGA

FPGA circuits (Field Programmable Gate Arrays) have changed designing and building systems in the form of electronic circuits. One of the reasons for this upheaval is that the FPGA offers an exceptional compromise between software and hardware. Vivado HLS software (high-level synthesis) is among the best-known software and tools used for programmable logic design. It is also used to generate the IP core and could considerably reduce the development cycle for FPGA cards. Utilizing Vivado HLS can shorten 1/3 of the RTL simulation time and increase the algorithm verification speed by more than ten times [25]. Using Vivado HLS, users can use the reliably existing software code, and they will not need advanced hardware knowledge or HDL programming experience.

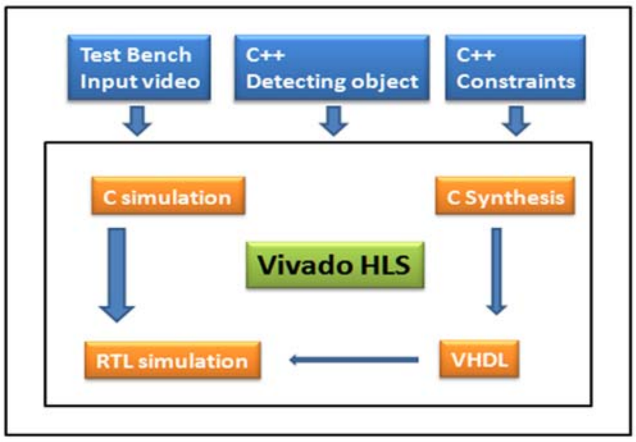


Figure 11. FPGA Xilinc Architecture.

In this experiment we will use the 2017.3 version of Vivado HLS, and as a card we will choose the zynq product family whose target device is xc7z020clg484-1.

The implementation costs can be obtained in synthesis report, which is calculated by Vivado HLS. The results were shown in Table I and II, contains the following elements:

- BRAM: the number of Block RAM
- FF: flip-flops
- DSP: digital signal processors
- LUT: look-up-tables

The resource utilization rate of FPGA was illustrated in the last row of table 2.

TABLE I. FPGA PERFORMANCE ESTIMATES

FPGA PERFORMANCE ESTIMATES		
Clock	***Target***	***Estimated (ns)***
ap-clk	100.00	85.31

TABLE II. UTILIZATION ESTIMATES SUMMARY.

Name	**BRAM 18K**	**DSP48E**	**FF**	**LUT**
DSP	-	-	-	-
Expression	-	-	0	84
FIFO	48	-	1004	4312
Instance	59	140	15018	31961
Total	107	140	16028	36393
Available	280	220	106400	53200
Utilization %	**38**	**63**	**15**	**68**

From the results in Table 1, the average time for processing all frames in Vivado HLS and CPU are 0.08531s and 1.131157 seconds, respectively. It can be shown that using Vivado HLS to execute our system is 13.2 times faster than the CPU.

V. SYNTHESIZES

This section discusses the results of the five algorithms chosen (Gaussian Blur, Erode, dilate, In-range, and RGB to HSV) regarding time-performance. Using these results, we analyze the performance of our system. The first important note that we noticed is that the GPU can compete and do the same performance as the CPU in the Gaussian blur filter. The GPU is also slower than CPU in RGB to HSV algorithm because of the limitation of memory access due to its architecture. So we can say that the nature of the data handled plays a decisive role in determining these platforms performance.

From the results in Table 3 (designed from all the past experimentations), the total average time for processing all frames in FPGA and CPU, GPU is 0.08531s, 1.131157, and 1.78509 seconds, respectively. It's clear that using FPGA in our system is 14 times faster than the CPU and GPU. The performance of FPGA is limited by memory and bandwidth. Note that it is possible to double the performance by processing twice the number of pixels in parallel.

TABLE III. EXECUTION TIME IN SECONDS FOR FPGA,RASPBERRY, CPU AND GPU.

Function	FPGA	Raspberry Pi	CPU	GPU
Gaussian blur	0.0315	9.071972	0.51344	0.55172
Erode Dilate	0.0268	2.940461	0.08203	0.30152
Inrange	0.0160	1.033809	0.08483	0.86374
Rgb to HSV	0.0121	0.003812	0.4512	0.06810
Total	**0.085s**	**13.050054s**	**1.13157s**	**1.78509s**

VI. CONCLUSION

In this paper, we have compared the GPU performance (execution time) with CPU, FPGA, and raspberry Pi 3 using five standard algorithms in image processing. Many studies show that performance using GPU is better compared to using CPU. To our current knowledge, this topic has not been previously investigated in the literature; that compares the GPU performance with CPU, raspberry pi 3, and FPGA.

This result advocates GPU devices potential usage in innovative software due to their flexibility; it means that the program can be changed easily during the development process. And on top of that, the nature of the data handled and the algorithm used are decisive for performance. For example, uniform images with filter usage are better in GPUs than CPUs. However, FPGA gives better results in massive pixel processing, with algorithms that run in parallel.

We have the following issues which have to be considered. We have compared the performance using only five problems. In our work, power consumption is not considered.

REFERENCES

[1] Marwa K. Elteir, Shaimaa Lazem, Mohamed Azab, ‘‘ Unleashing the hidden powers of low-cost IoT boards: GPU-based edutainment case study’’; Journal of King Saud University –Computer and Information Sciences. https://doi.org/10.1016/j.jksuci.2020.02.001

[2] B. Veenboer ∗, J.W. Romein, "Radio-astronomical imaging on graphics processors", Astronomy and Computing 32 (2020) 100386. https://doi.org/10.1016/j.ascom.2020.100386

[3] Alfredo Cuzzocrea ∗, Enzo Mumolo, "A novel GPU-aware Histogram-based algorithm for supporting moving object segmentation in big-data-based IoT application scenarios". Information Sciences 496 (2019) 592–612. https://doi.org/10.1016/j.ins.2019.03.029

[4] Sen Du, Tian Huang, Junjie Hou, Shijin Song, Yuefeng Song, “FPGA based acceleration of game theory algorithm in edge computing for autonomous driving”. Journal of Systems Architecture 93 (2019) 33–39. https://doi.org/10.1016/j.sysarc.2018.12.009

[5] S. Esquembri∗, J. Nieto, M. Ruiz, A. de Gracia, G. de Arcas. ‘‘ Methodology for the implementation of real-time image processing systems using FPGAs and GPUs and their integration in EPICS using Nominal Device Support’’. Fusion Engineering and Design 130 (2018) 26–31. doi.org/10.1016/j.fusengdes.2018.02.051

[6] Amir HajiRassouliha *, Andrew J. Taberner, Martyn P. Nash, Poul M.F. Nielsen, ‘‘ Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms ‘’. Signal Processing: Image Communication 68 (2018) 101–119. doi.org/10.1016/j.image.2018.07.007

[7] Murad Qasaimeh, Joseph Zambreno and Phillip H. Jones, Kristof Denolf, Jack Lo and Kees Vissers, “Analyzing the Energy-Efficiency of Vision Kernels on Embedded CPU, GPU and FPGA Platforms”, IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2019. DOI 10.1109/FCCM.2019.00077

[8] Maxim Shepovalov, Venkatesh Akella, FPGA and GPU-based acceleration of ML workloads on Amazon cloud - A case study using gradient boosted decision tree library, Integration, Volume 70, January 2020, Pages 1-9. doi.org/10.1016/j.vlsi.2019.09.007

[9] P. Liu, S. Li and Q. Ding, "An Energy-Efficient Accelerator Based on Hybrid CPU-FPGA Devices for Password Recovery," in IEEE Transactions on Computers, vol. 68, no. 2, pp. 170-181, 1 Feb. 2019

[10] X. Tang and Z. Fu, "CPU–GPU Utilization Aware Energy-Efficient Scheduling Algorithm on Heterogeneous Computing Systems," in IEEE Access, vol. 8, pp. 58948-58958, 2020, DOI: 10.1109/ACCESS.2020.2982956.

[11] Riza Satria Perdana, Benhard Sitohang, Andriyan B. Suksmono, A Survey of Graphics Processing Unit (GPU) Utilization. for Radar Signal and Data Processing System, 6th International Conference on Electrical Engineering and Informatics (ICEEI), 2017

[12] Getreuer, Pascal (17 December 2013). "ASurvey of Gaussian Convolution Algorithms". Image Processing on Line. 3: 286–310. doi:10.5201/ipol.2013.87

[13] Michael W. Schwarz; William B. Cowan; John C. Beatty (April 1987). "An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models". ACM Transactions on Graphics. 6 (2): 123–158. doi:10.1145/31336.31338

[14] Dila tation et l’érosion[Digital Image Processing, 3rd Ed., chapter 9 ”Morphological Image processing”,Rafael C. Gonzalez and Richard E. Woods, Prentice Hall, 2008]

[15] Y. S. Ming, H. D. Li, X. M. He. Contour completion without region segmentation. IEEE Transactions on Image Processing, vol.25, no.9, pp.3597-3611, 2016. DOI:10.1109/TIP.2016.2564646

[16] D. Ziou, S. Tabbone. Edge detection techniques-an overview. Pattern Recognition \& Image Analysis, vol.8, no.4, pp.537-559, 1998.

[17] D. R. Martin, C. C. Fowlkes, J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.26, no.5, pp.530-549, 2004. DOI:10.1109/TPAMI.2004.1273918

[18] B. Hariharan, P. Arbelaez, L. Bourdev. Semantic contours from inverse detectors. In Proceedings of IEEE Conference on Computer Vision, IEEE, Barcelona, Spain, pp. 991–998, 2011

[19] Xin-Yi Gong, Hu Su, De Xu, Zheng-Tao Zhang, Fei Shen, Hua-Bin Yang. An Overview of Contour Detection Approaches[J]. International Journal of Automation and Computing, 2018, 15(6): 656-672

[20] Z. Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 22 , Issue: 11 , Nov 2000),Page(s): 1330 - 1334
[21] Adrian Kaehler; Gary Bradski (14 December 2016). Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media. pp. 26ff. ISBN 978-1-4919-3800-3.
[22] OpenCV C interface: http://docs.opencv.org
[23] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, "GPU Computing," in Proceedings of the IEEE, vol. 96, no. 5, pp. 879-899, May 2008, doi: 10.1109/JPROC.2008.917757.
[24] Dumitrel Loghin, Lavanya Ramapantulu, Oana Barbu, Yong Meng Teo, A time–energy performance analysis of MapReduce on heterogeneous systems with GPUs, Performance Evaluation, Volume 91, 2015, Pages 255-269, ISSN 0166-5316, https://doi.org /10.1016/ j. peva.2015.06.015.
[25] Hui Gao, Houde Dai, Yadan Zeng, High-Speed Image Processing and Data Transmission Based on Vivado HLS and AXI4-Stream Interface, Proceeding of the IEEE International Conference on Information and Automation Wuyi Mountain, China, August 2018