Three Video Applications using an FPGA based pyramid implementation: Tracking, Mosaics and Stabilization

Marco Aurelio Nuño-Maganda, Miguel O. Arias-Estrada, Claudia Feregrino-Uribe
Instituto Nacional de Astrofísica, Óptica y Electrónica
Luis Enrique Erro No 1. Sta María Tonanzintla, Puebla, México.
nmaganda@cseg.inaoep.mx, ariasm@inaoep.mx, cferegrino@inaoep.mx

Abstract

In this paper we present a set of hardware modules which form the basis for three vision applications: Target Tracking, Image Stabilization and Image Mosaicking. The two main modules are: the pyramidal module and the multiresolution correlation module. They were implemented using the Handel-C language, and tested in the Celoxica RC1000 development platform, which has an Virtex-E FPGA. We show the performance statistics for tracking more than one target using the basic modules, and present results of the applications implemented based on these basic modules.

1. Introduction

Computer Vision is one of the main branches of Artificial Intelligence [1]. The Computer Vision applications obtain a set of images from certain source, they process them and generate as result images or data that later are analyzed depending on the application. Throughout history many alternatives for digital image processing have been used. One of this alternatives is the use of supercomputers or dedicated workstations. A different alternative is the use of FPGAs (Field Gate Programmable Array), which are programmable digital devices, that unlike the microprocessors, what is programmed is not an instruction set, but the description of an architecture based on basic components[4]. Currently, FPGA devices with capacity equivalent to millions of digital gates are available.

One of the image representation used for video processing is the pyramid, because it separates the total resolution of the original image in a set of images that represent it as a set of different spatial resolutions.

2. Background

The strength of the pyramid comes from the increase in processing speed on the image operations [3]. This simplification permits to work at coarser resolutions where there are less pixels to be processed. Each level of the pyramid is ¼ smaller that the preceding level, allowing this a reduction in the number of pixels to be processed by a factor of 4, 16, 64, 256 and so on.

Many algorithms that work at pyramidal level are called coarse-to-fine algorithms. They process the image in a very coarse resolution and obtain a first result of the processing, which is vague, because it works with low resolution images. In order to obtain better results, they are refined repeating the processing to high resolutions but, using as a reference the results obtained from lower resolutions.

2.1. Gaussian Pyramids

In order to generate a gaussian pyramid, we suppose that the image is represented by an array g_0 of C columns by R rows [4]. Each value represents the intensity of the corresponding point in the image. This image is the level 0 of the gaussian pyramid. The Level 1 of the pyramid consist of an image g_1 which is a reduced version of the image g_0 to which a low-pass filter was applied. Each value within level 1 is calculated like a weighed average of the values in level 0 within a 5x5 window. Each value of level 2, represented by g_2, is obtained from the values of level 1 applying the same target. The fig 1 shows the pyramid of an image following this scheme.
The filter process of level at level is implemented by the function *Reduce*:

$$g_k = \mathit{Reduce}\,(g_k\text{-}1)$$

Which means that for levels $0 < l < N$ and nodes i, j, $0 < i < C_l$, $0 < j < R_l$

$$g_l(i,j) = \sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n)\, g_{l-1}(2i+m, 2j+n).$$

where N is the number of levels of the pyramid whereas C_l and R_l are the dimensions of the l-th level, and m and n are the subindex of the convolution mask. The density of the nodes is reduced by half in a dimension or four in two dimensions from level to level. The pattern of weights w must fulfill the following properties:

a) Generally, its size is 5 x 5 pixels

b) Must be separable

$$w(m,n) = \hat{w}(m)\hat{w}(n)$$

c) The components of the pattern must be standardized

$$\sum_{m=-2}^{2} \hat{w}(m) = 1$$

d) In addition, these components must be symmetrical

$$\hat{w}(i) = \hat{w}(-1) \quad for\ i = 0,1,2$$

For example, in a 512 x 512 image, level 1 will be 256 x 256 pixels , level 2 128 x 128 pixels, and level 3 64 x 64 pixels.

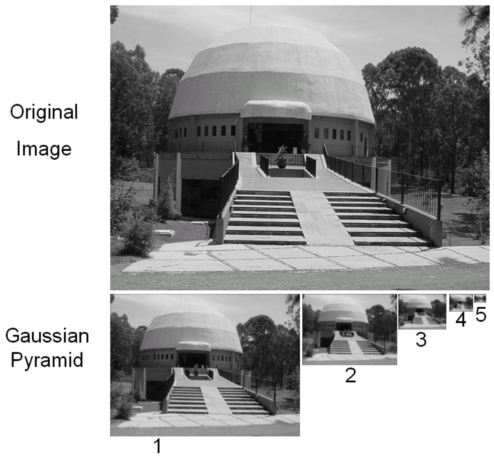


Fig 1. Gaussian Pyramid of the Image 'INAOE'

2.2 Correlation Measures

There are several correlation measures in the literature. The three more used are:

a) Sum of Squared Differences [5]:

$$SSD = \sum_{ij} \left(I_1(x+i, y+j) - I_2(x+i+d, y+j) \right)^2$$

b) Sum of Absolutes Differences [5]:

$$SAD = \sum_{i,j} \left| I_1(x+i, y+j) - I_2(x+i+d, y+j) \right|$$

c) Normalized Cross Correlation [6]:

$$NCC = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \frac{fr(i,j) fp(i,j)}{\sqrt{fr(i,j)^2} \sqrt{fp(i,j)^2}}$$

3. Basic Modules

In this section we present the two basic architectures developed, that are integrated to obtain a multiresolution correlation architecture, which is fundamental for the implemented applications. Instead of showing the architectures in terms of functional blocks, we show them in terms of processes. This is an advantage that allows us to the use a hardware description language based on the standard ANSI-C. These processes are mapped into hardware blocks by the place and route tools of the FPGA vendor, thus the user must not worry any longer about the connection between the different control lines.

3.1. Pyramidal Architecture

In fig 2 are shown the modules that compose the pyramidal architecture. These modules are:

a) Data Memory. This memory keeps the image to be processed.
b) Convolution Modules. These modules multiply the image values with the convolution mask coefficients.
c) Control Generators. These modules access to the Coefficients Memory and supply them to the Convolution Modules.
d) Coefficients Memory. This is a read-only memory that stores the convolution mask coefficients.
e) Address Generator. This is a small control unit that supplies the control signals and the synchronization of the different modules to the architecture.
f) Multiplexor. This element isolates the architecture bus of the memory data line, with the purpose of controlling the access to the bus.

The interaction of the parts previously mentioned is carried out like:

1. The original image is read from data memory and stored in the image registers.
2. The Control Generator access the Coefficients Memory and pass them to the Convolution Modules depending on the mask and the image row being processed.
3. The Convolution Module reads the data contained in the Image Registers and carry out the multiplications of the image values by the coefficients of the convolution mask.
4. Whenever the Convolution Module generates a result, it is stored in the Registers Bank.
5. Once the Registers Bank has been filled, an interruption to the architecture is generated to keep its content in the Data Memory.
6. Steps 1-4 are repeated until finishing with the original image

The association of two convolution modules with two control generators is denominated *processor of pyramidal* *convolution*. The more basic implementation of the pyramidal architecture contains one processor of pyramidal convolution. If additional parallelism is required, more processors of pyramidal convolution can be implemented.

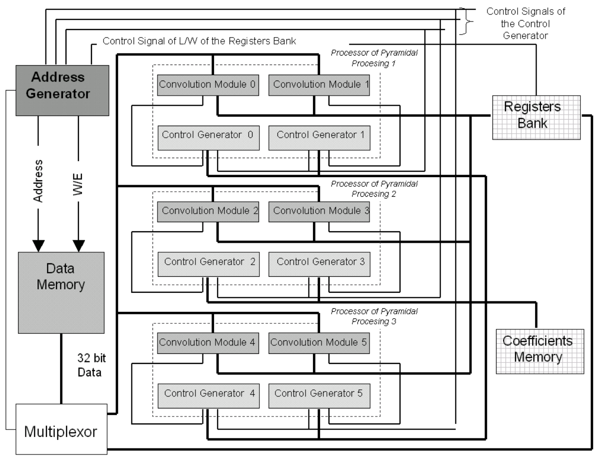


Fig 2. Main Modules of the Pyramidal Architecture

3.2. Correlation Architecture

In Fig 3 the modules that compose the correlation architecture are shown. These modules are:

a) Image Memory. This memory stores the image.
b) Target Memory. This memory stores the target to be tracked.
c) Image Registers. These registers stores some values of the image to be processed.
d) Target Registers. These registers store some values of the target to be processed.
e) Correlation Functions. These blocks implements the correlation function to be used in the architecture. There are four blocks, because it is possible to process four correlation functions in parallel with this scheme. The possible functions to be implemented in this block are the mentioned in section 2.2.
f) Correlation Registers. These registers store the accumulated values of the correlation functions being processed.
g) Local Comparator. This block evaluates which of the four correlation registers generate the best result.
h) Best Global Register. The best result from the Local Comparator is compared with this value to find the best global value according with the correlation function.
i) Global Comparator. This block compares the Best Global Register with the Best Local obtained from the Local Comparator.
j) Current Row and Current Column Registers. These registers store the current Row and Column values.
k) Best Row and Best Column. When the global comparator find a better value than the stored, these values are updated depending on the values of the Current Row and Current Column Registers.

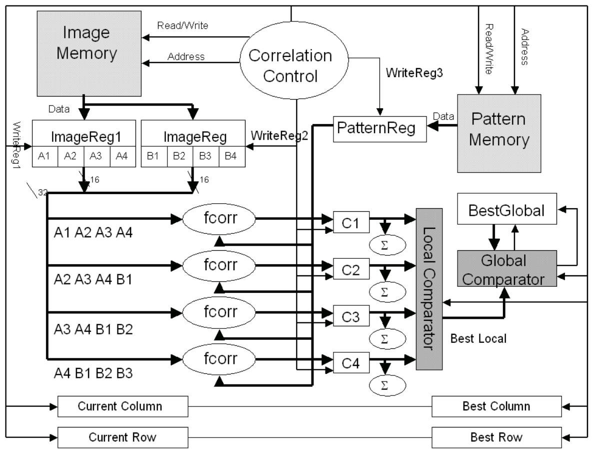


Fig 3. Main Modules of the Correlation Architecture.

The interaction of the mentioned parts previously is:

1. The image registers and the target register are initialized with some values of the image and the target respectively.
2. The correlation functions calculate the correlation index between the image values and the target points and store and accumulate them in the Correlation Registers.
3. In parallel with the step 2, the Local Comparator obtains the best value stored in any of the Correlation Registers.
4. In parallel with steps 2 and 3, the Global Comparator compares its value with the one obtained from the Local Comparator. When the value obtained from Local Comparator is the best, this value is stored in the Best Global Register, and the registers of Best Row and Best Column registers are updated depending on the values stored in the Current Column and Current Row registers.
5. This process is repeated until all the possible parts of the image with the same size that the target are processed.

3.3. Multiresolution Correlation

Once the correlation and pyramid modules have been implemented, it is necessary to explain how these architectures are combined to carry out the multiresolution correlation.

Several ways of finding an object within an image have been developed. One of these consists of carrying out the correlation of all the possible regions within the image of the same size of the object and finding the maximum or the minimum value of correlation that indicates the object position in the image.

In order to carry out the correlation process, the pyramid is obtained from the image in where the object of interest will be searched. Also, the pyramid must be obtained from the target to be searched. The number of levels of both pyramids must be the same.

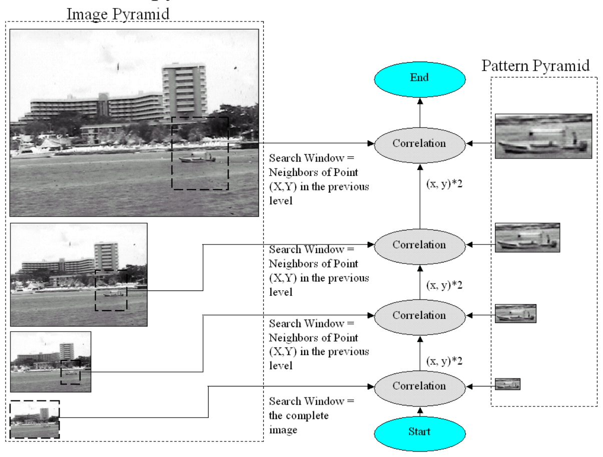


Image Pyramid
Pattern Pyramid
End
Correlation
Search Window = Neighbors of Point (X,Y) in the previous level
(x, y)*2
Correlation
Search Window = Neighbors of Point (X,Y) in the previous level
(x, y)*2
Correlation
Search Window = Neighbors of Point (X,Y) in the previous level
(x, y)*2
Correlation
Search Window = the complete image
Start

Fig 4. Multiresolution Correlation Process

The correlation process starts in the lowest level of both pyramids. In this level, the search window consist of the whole image in the last level of the pyramid. Because there are a few pixels to process, this stage is fast. The process obtains a tentative position, that is a vague result, but it serves as basis for future refinement.

Once the first result has been obtained, the process goes to the next upper level in both pyramids. The correlation process starts but the correlation window is limited depending on the results obtained from the lowest level. With this limitation, we reduce the number of pixels to be processed, but the result is refined. This process is repeated until the level 0 of both pyramids, and then the process gives the position as result (row and column) where the target is located (tentatively). This process is shown in fig 4.

3.4. Multiresolution Multiple Target Tracking

The process mentioned in the previous section can be applied to follow more than one target. Due to the complexity of the architecture, we preferred to explain it in terms of processes and not in terms of functional blocks.

In the first frame, the architecture must take its parameters from certainly memory region. Later, it must calculate a memory addresses, according to the number of targets and their sizes.

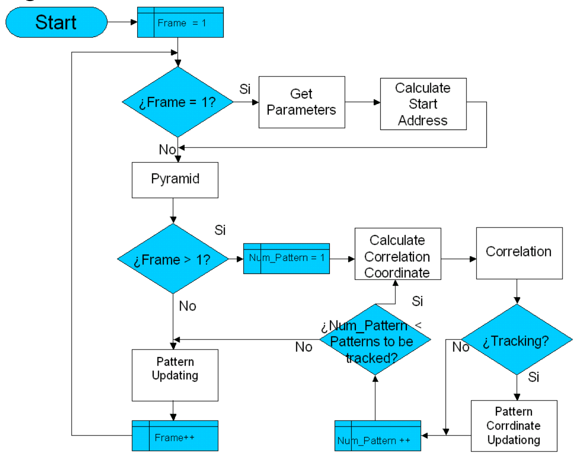


Fig 5. Multiple Multiresolution Target Tracking Process

In all the frames, the pyramid module must process the image and obtain the pyramid from the original image. With respect to the correlation process, in the first frame this process remains inactive, only the updating targets process is carried out, depending on the coordinate of each one of the targets. After the first frame, the correlation process is performed depending of the target processed. This process starts in the low level of the pyramids of target and image, and finish when the process has obtained the tentative coordinate of the target in the image. Later, this coordinate is updated depending of the application using the architecture. This process is repeated for each target to be tracked. Finally, the target updating process is activated. This process is shown in fig 5.

4. Real-Time Applications

4.1. Tracking

The tracking benefits from the multiresolution pyramid because in each level of the pyramid the amount of data to be processed is reduced. The change made with respect to the multiresolution correlation implementation consists on storing the target and later updating it with the target located in each frame. This causes that the tracking obtains certain robustness, because it will not be affected by rotation or changes of the object form. This process is shown in fig 6.

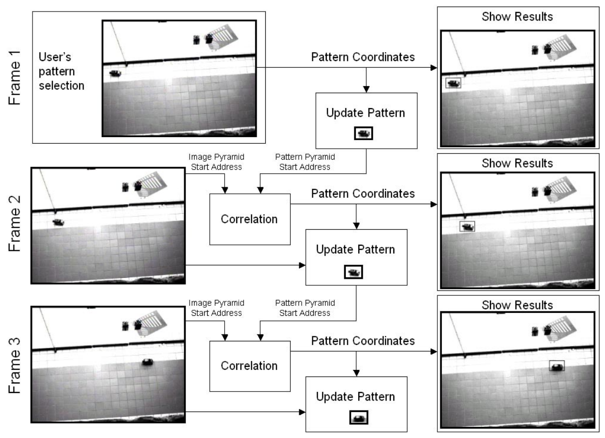


Fig 6. Iterative Target Tracking Process

4.2. Mosaics

In this application, there are several targets that are randomly located within the image. A set of targets from the central column of the image is taken as a reference; its movement and the direction of the movement with respect to the next image will help to determine the parts of these images that will be considered for the mosaic construction. This process is shown in fig 7.

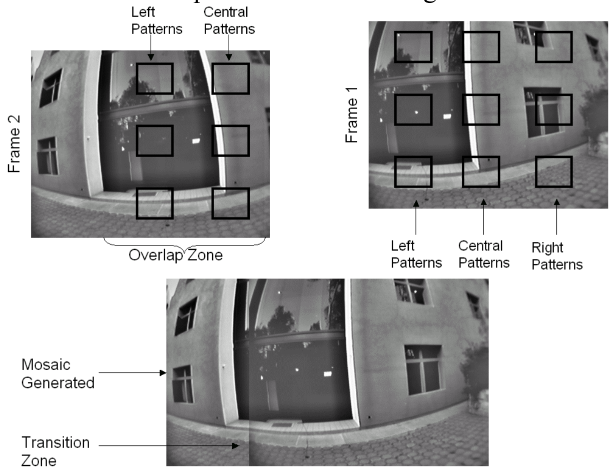


Fig 7. Mosaics Generation Process

4.3. Image Stabilization

In this application, there are several targets located in the central part of the image. A set of targets from this part of the image is taken as a reference; its movement and the direction of the movement with respect to the next image will help to determine the parts of these images that will be considered for the image stabilization. This process is shown in fig 8.

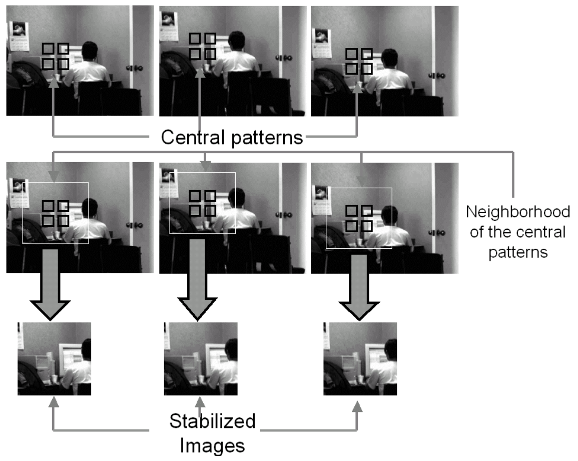


Fig 8. Stabilization Process

4.4. Voting system for resolving movement vector

When we combine the movement of one or more targets, we can assume that always ideal conditions will occur. But ideal conditions rarely occur. As an extra work, we developed a small voting system for solving inconsistencies when the targets do not follow the next conditions:

- The target must be differentiable of others objects in the scene.
- The patter must not be part of the background.
- The target never must exit of the vision field of the camera.

The voting system groups the targets according to the movements of the majority of the targets. With this voting system, if one of the tracked target produces an incorrect result, this does not affect the global movement vector.

5. Implementation and Results

The implementation of the proposed architectures was made on the Celoxica RC1000 development platform. We show results of the pyramidal architecture, tracking performance and also show results of the applications implemented.

5.1. Performace Results

In this section the operation statistics of the pyramidal architecture are shown. The architecture was implemented using different number of processors of pyramidal convolution. We tested for 1, 2 and 3 processors of pyramidal convolution. The resources used by each one of these applications are shown in table 1.

In order to obtain the performance statistics, the clock cycles that the architecture needs for producing the result were counted. Once the number of cycles were obtained, they were multiplied by the inverse of the frequency (in this case, 25 MHz), in this way we obtain the image processing time and the number of images processed per second. These results are shown in table 2.

Processors	Slices	Flip-Flops	4 input LUTS	Block RAMs	Equivalent Gates
1	*2,197 of 19,200 (11%)*	*251 of 38,400 (1%)*	*3,787 of 38,400 (9%)*	*22 of 160 (13%)*	*397,138*
2	*3,640 of 19,200 (18%)*	*289 of 38,400 (1%)*	*6,333 of 38,400 (16%)*	*42 of 160 (35%)*	*748,209*
3	*5,139 of 19,200 (26%)*	*329 of 38,400 (1%)*	*8,975 of 38,400 (23%)*	*62 of 160 (38%)*	*1,100,581*

Table 1. Resources of the FPGA used by the Pyramidal Architecture**.**

Pyramidal Convolution Processors	***Number of Cycles***	***Processing Time (ms)***	***Images Processed by Second***
1	*696,613*	*27.86*	*35.88*
2	*464,444*	*18.57*	*53.82*
3	*386,923*	*15.47*	*64.61*
4	*348,568*	*13.94*	*71.72*

Table 2. Pyramidal Architecture Performance

5.2. Pyramid Results

The results shown for the pyramidal architecture were obtained directly from the implemented architecture. We developed an interface in Visual C++ for reading the images from files, passing them to RC1000 memory and later obtaining the processing result. The results are compared with the ones obtained from the Matlab simulation. We also show the difference between software simulation and hardware implementation pyramids, and then we evaluate the quality of FPGA processing. These results are shown in fig 9.

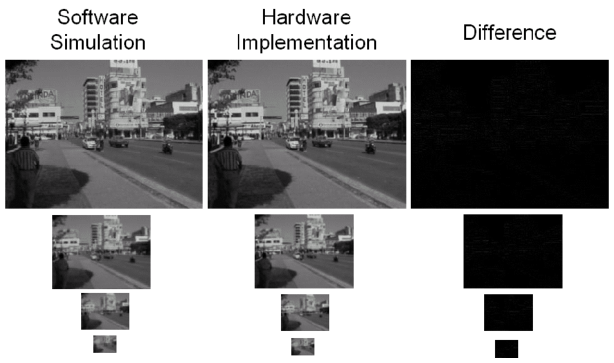


Fig 9. Comparison between Software Simulation and Hardware Implementation of Pyramids

5.3. Tracking Performance Results

The architecture is flexible with respect to the target size. We carried out tests with different target sizes. These results are shown in tables 3, 4 and 5.

Target size = 40 x 40				
Targets	Latency Cycles	Throughput Cycles	Processing Time	Images processed per Second
1	383,548	414,928	16.59 ms	60.25
2	386,041	448,895	17.95 ms	55.69
3	388,534	482,862	19.31 ms	51.77
4	391,024	516,829	20.67 ms	48.37
5	393,520	550,796	22.03 ms	45.38

Table 3. Tracking Performance for a 40 x 40 pixels Target

Target size = 60 x 120				
Targets	Latency Cycles	Throughput Cycles	Processing Time	Images processed per Second
1	391,090	477,498	19.09 ms	52.35
2	401,135	574,035	22.96 ms	43.55
3	411,180	670,574	26.82 ms	37.28
4	421,230	411,180	30.68 ms	32.58
5	431,280	767,109	34.54 ms	29.94

Table 4. Tracking Performance for a 60 x 120 pixels Target

Target size = 80 x 80				
Targets	Latency Cycles	Throughput Cycles	Processing Time	Images processed per Second
1	390,228	491,608	19.66 ms	50.85
2	399,401	602,255	24.09 ms	41.51
3	408,574	712,902	28.51 ms	35.06
4	417,747	823,549	32.94 ms	30.35
5	426,920	934,196	37.36 ms	26.76

Table 5. Tracking Performance for a 80 x 80 pixels Target

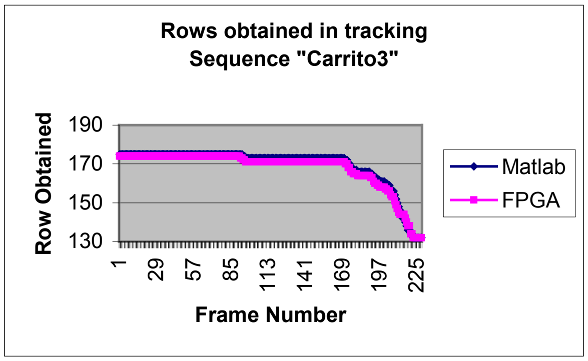
5.4. Applications Results

In this section we show the results of the different interfaces for the applications implemented.

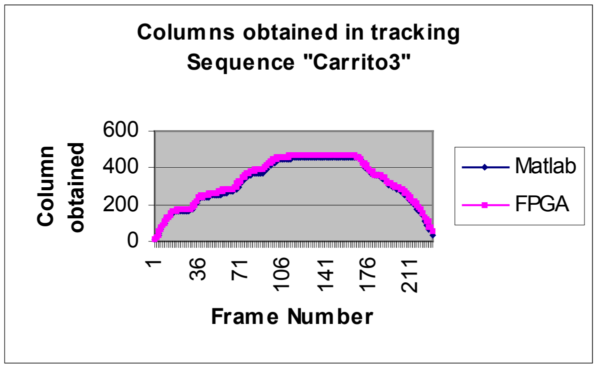
5.4.1. Tracking

The results shown are divided in two parts: the first one compares the results between the hardware architecture and the tracking simulation in Matlab. The second one consist on showing the tracking of an object along an image sequence.

In the first part we show two graphs, where the row and the column obtained by the tracking process are compared. We can remark that the difference between the positions is minimal and the object of interest is always tracked. In fig 10 we show the result of tracking a sequence. In graph 1 and 2, we show the comparison between the FPGA tracking and the Matlab tracking applied to the same sequences.



Graph 1. Comparison between Matlab Tracking and Tracking Architecture



Graph 2. Comparison between Matlab Column Tracking and Tracking Architecture

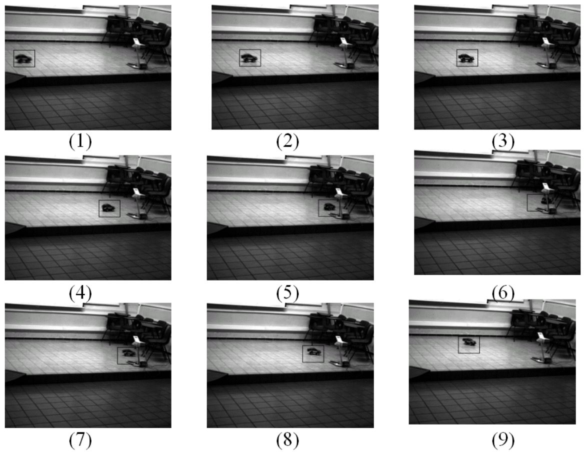


Fig 10. Tracking of an object along of a sequence

5.4.2. Mosaics

With respect to the mosaics generation, we carried out tests according to the methodology proposed in this work. In fig 11 we show both mosaics generated and the original images. The images were digitalized directly with an application made in Visual C++, each image was passed to the FPGA for its processing. The Visual C++ Interface has the main task of generating the mosaic depending of the overlaping degree between the images.

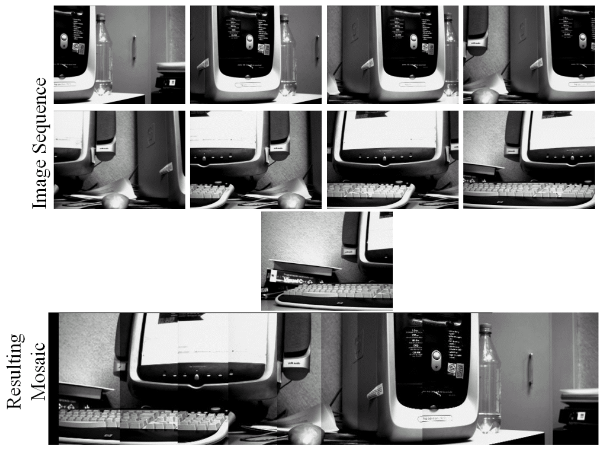


Fig 11. Mosaic Generated

5.4.3. Stabilization.

In fig 12 we show the results of the stabilization process. We show the images taken without processing, which were digitalized with an interface implemented in Visual C++. Each image was passed to the FPGA for its processing, and this interface showed us the resulting subimage depending of the movement vector obtained from the target predefined. The images are trimmed 1/10 of the original image, and this is the reason of this lower resolution.

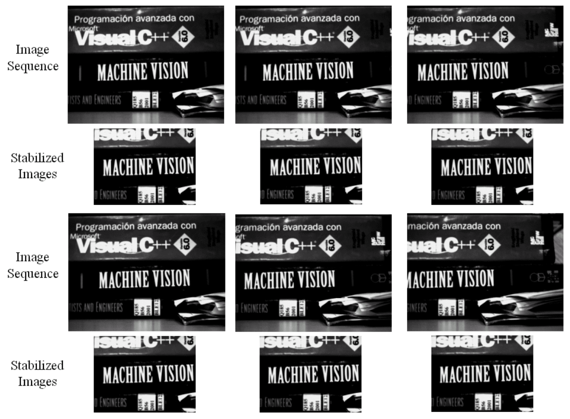


Fig 12. **Stabilization of an image sequence**

6. Conclusions

In this work a hardware architecture for tracking objects was presented. The basic characteristics are the flexibility with respect to the parallelism degree that is desired to implement. The degree of parallelism depends on the available resources in the FPGA used for its implementation, on the desired speed and on the tracking application.

Also it is possible to confirm that Handel-C is a language that shortens the design and implementation times of complex hardware architectures. It also allows to make a greater number of tests and verifications before implementing a final design, which can be used in applications with lower space requirements or power consumption. In addition, high performance applications can be designed with the purpose of reaching real time performance.

7. Future Work

In this work a multiresolution correlation hardware architecture was implemented with the applications mentioned in this work. As future work we propose a hardware implementation of some technique of machine learning to improve the quality of the tracking, like Radial Basis Neural Network or Holographic Neural Networks. Also we propose the use of other techniques of pyramidal decomposition, like the Wavelet transform. Within the proposed code, it is necessary to make some improvements to implement the architecture in smaller capacity FPGAs, with the aim of making some prototype of mobile application, that requires circuits of low power consumption and less area.

8. References

[1]. G. Pajares and J. De la Cruz, "*Computer Vision"*. Ed. Ra-ma, México, 1995

[2]. P.J. Burt and E.H Adelson, "*The Laplacian pyramid as a compact image code*". IEEE Trans. Commun. COM-31, pp 532-540, 1983.

[3]. M. W. Hansen, M. R. Piacentino, and G. S. van der Wal. "*The Arcadia Vision Procesor"*. IEEE International Workshop on Computer Architectures for Machine Perception, pp 31-41, 2002.

[4]. S. Brown, and Jonathan. Rose, "*Architecture of FPGAs and CPLDs: A Tutorial*", IEEE Design and Test of Computer, Vol. 13, No. 2, pp. 42-57, 1996.

[5]. Juan Manuel. Xicoténcatl, *"Hardware Architecture for Stereo Vision in Real-Time"*. MSc. Thesis, INAOE, México, 2000

[6]. Cesar Darío, Peregrina, *"Object Tracking using Active Vision"*. MSc. Thesis, INAOE, México, 2002.