# FPGA Implementation of Camus Correlation Optical Flow Algorithm for Real Time Images

Pedro Cobos Arribas

*Dpto. de Sistemas Electrónicos y de Control. E.U.I.T. Telecomunicación. Univ. Politécnica de Madrid*

*pcobos@sec.upm.es*

Félix Monasterio Huelin Maciá.

*Dpto. de Tecnologías Aplicadas a la Telecomunicación. E.T.S.I. Telecomunicación. Univ. Politécnica de Madrid*

*felix@robot.tat.upm.es*

## Abstract

*This document describes a FPGA implementation of an optical flow correlation based detection method which is very fast (with low accuracy) and allows near real time applications (6 frames/second) on standard computing hardware. It manages the motion estimation problem with a speed restriction, obtained from an image down sample process, that yields image speeds less than one pixel per frame. Then a patch correlation image method is applied and a temporal search can be made at a lineal time versus the spatial search at a quadratic time of other classic matching methods. The method fulfils all the requirements of a robotic vision system, being robust, fast and accurate enough for real time applications like time to contact detection and robot navigation tasks and like the global active vision system that is being developed, which operates from the two dimensional images coming from a CCD camera and a frame buffer, makes the logarithm-polar transform, the optical flow calculation and the image invariants extraction*

*The objective of this job is to evaluate the skills of this algorithm for obtaining optical flow at high speed, to allow real time image processing. The result is a realisation on a FPGA of ALTERA that allows the processing of the optical flow on 100x100 pixels images in real time (25 images/second).*

## 1. Introduction

The first part of this document describes the theoretical study of an optical flow algorithm, which is a part of a visual processing system, fixed in a global active vision system. In the second part, the hardware system design, which makes this task at real time, is also described.

First of all, the environment where this application has to work is described, with the vision system and the frame-grabber used. In the following section, the theoretical bases of the optical flow algorithm are explained, as well as their characteristics, properties, advantages, troubles and possible applications.

The next part contains the experimental description of a hardware application (with an Altera FPGA), that has been designed and simulated.

Finally, in section 4 the results and conclusions are presented.

### 1.1. Vision system

The active vision system can be divided in the following blocks (Figure 1):

- CCD camera: it contains the CCD sensor that provides monochrome images of 480 x 500 pixels.
- Frame grabber: This block makes the analog to digital conversion of the sensor 's signals to a data collection (three frames on three dual port memories).
- Log-Polar: it carries out a transformation on the image , that imitates the human vision system, to reduce its size (and process time) without the lose of properties.
- Optical flow: It calculates the optical flow (with a minimum of two consecutive images).
- Actions calculus: It processes the received data and decides where to focus the camera.
- Control camera: It carries out the necessary motions to focus the camera in a concrete point.

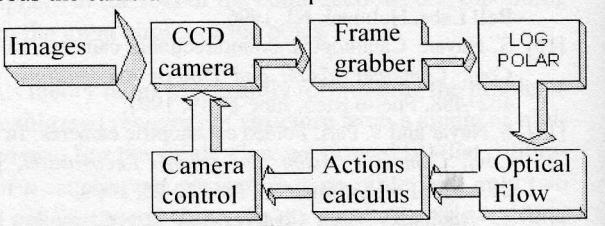



**Figure 1**

Some implementations of these blocks are described in [1], [2]

## 2. Theoretical bases

### 2.1. Optical flow obtaining techniques

Optical flow is a concept with many definitions: the apparent motion that an observer notices in an image, a two dimensional vector that indicates the motion of objects, or features, in a sequence of images, etc. The definition is lees important that the fact that it is the best option to calculate three-dimensional properties of the environment, and to obtain other useful motion information, starting from the luminance changes of the image plane points.
The optical flow algorithms have many applications, where stands out the autonomous mobile robot navigation in an unknown environment. These techniques are detailed in [3] and are generally very difficult to support real-time performance (only with strong restrictions on the environment). They can be divided on four types:
· Differential techniques.
· Matching or regions correlation.
· Frequency based methods (based on energy signal).
· Phase based methods.

Many of this algorithms are not suitable for practical applications actually, due to the hardware requirements (parallel computers, or ASIC´s) of real time applications, or to the excessive duration (hours) of other approaches with high precision, on a workstation.

The algorithm that will be described next uses correlation (matching) regions techniques, but it modifies them in two aspects to avoid the huge amount of time used in the homologous search at successive images, seeking that the time used in the process doesn't grow in a quadratic way with the size of the search area in the image. The first modification implies the use of a time variable sampling of the image that allows to exchange the space with the time and it drives to an algorithm that is lineal at worst case and constant at the best, with the size looked for, or otherwise, with the objects in motion speed inside the image. This technique detects even the fastest motion in an image. So, it is focused for robotics applications like objects pursuit, where the most important thing for the robot is usually to detect the objects that move faster in its near environment (possibility to avoid them, to catch them, etc.). The second modification allows to extend the application of the algorithm to create a flow field with multiple speeds that allow to transform the quadratic space searches in other lineal in time steps. This space-time inversion has the effect of searching for faster moving objects in each image instead of looking for the slower motion ones that only are searched when it is necessary. The theoretical ideas, base of the present article and its hardware implementation, have been progressively developed in [4], [5], [6], [7], [8], [9], [10] and [11].

### 2.2. Robust correlation_based matching method

An object moving in the three-dimensional space has a speed vector field $W(x, y, z)$ associated, which is projected on an observer 's retina to form a two-dimensional field $W_p(x. y)$. The retina really observes changes in the image points intensity values, for a discreet interval of time. The optical flow is a vector field that describes this change, by the indication of the motion of characteristic points from one image to another. It is assumed that the maximum possible displacement for any pixel is limited to a value $k$ in any address, being this value related to the expected speeds. In general some restrictions are required to calculate the optical flow field in a correct way starting from two successive image frames, but if the flow belongs to rigid bodied objects, the problem becomes well posed and it can be solved in a satisfactory way in most of the cases. In this situation, it is usually acceptable that a certain pixel has the same speed that the one in its neighbours. Supposing that these pixels are grouped in a squared neighbourhood (or window), of a size $v$, centred on a certain pixel, the pixel motion at $(x, y)$ is defined as the same of a window with of a $v \times v$ size, centred in $(x, y)$, out of $(2\delta+1).(2\delta+1)$ possible displacements.
The referred method determines also the correct motion of the pixels group by simulating it with each possible displacement of $(x, y)$ and calculating a measure of coincidence for each displacement. In Figure 2 we can see in an intuitive way, how the area correlation calculation process will take place in its original position with all its possible situations in the following image. We have one case with very low similarities and another with a very high correlation.

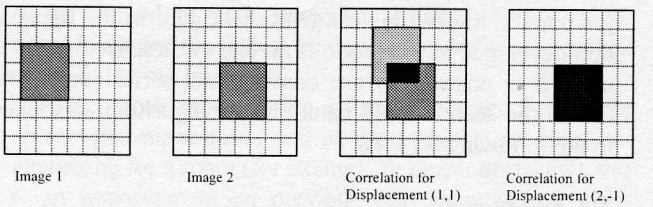



**Figure 2**

The measure of likeness for the displacements, or error function $(\phi)$, obtains a proportional value to the matching of two characteristics in the image. The coincidence degree $M(x, y; u, w)$ and a displacement $(u, w)$ can by calculated for a point, by means of the sum of $\phi$ values between each pixel in the displaced region in the first image (search area, $P_v$) and the corresponding pixel in the current region on the second image (reference image):

$$\forall u, w : M(x, y, u, w) = \sum \phi \, (E_1(i, j) - E_2(i + u, j + w) \quad , (i, j) \in P_v$$

The following step is the election of the error function $(\phi)$, between the different available options. The most often

used, due to the simplicity of its calculation process, is the absolute value of the differences between the values of both pixels luminance. Another possibility for $(\phi)$ is the squared difference of their luminance values. In this two cases that differs slightly, low values of $\phi$ imply a better likeness, being the precision and calculation process complexity greater in the one mentioned in the second place. It is also possible to measure the coincidence between two images using other primitive different to the values of the intensity. For instance characteristic in the image, as the borders, could be searched; and additional similarity cues could also be decided based on the intensity gradient in its borders, for example with the slope of the zero crosses of the Laplacian operator on the image.

In an ideal situation, with only front or parallel motion, no noise in the process, constant environment illumination and a translation of a integer number of pixels (inside a given range), we can expect that the pixels of both areas will match perfectly, being the motion of the pixel the one out $(2\delta+1) \times (2\delta+1)$ possible displacements, with the maximal coincidence degree. If the size of the neighbourhood is large, ties are rare and it is possible to decide one arbitrarily (local errors could take place). If the chosen displacement is the right one, all pixel values will match and the difference between both regions will be minimal and it will produce the maximal coincidence degree. This process is repeated for all the pixels of the image, regardless of the other pixels, with the exception that motion is not calculated in a $\delta + v$ border, since the process would have to use pixels that don't exist in the image. The algorithm produces a dense and smooth output, which does not require an additional smoothing or interpolation stage.

Parallel implementations [12], [5] are possible due to the fact that the displacements chosen for a pixel are independent of the rest of the pixels. It is also possible to perform the calculation process in dedicated chips (ASIC´s) or programmable logical devices (FPGA´s), like in the application developed in this work. If no one of these techniques is used, it is possible to use conventional serial computers, due to the search based nature of the algorithm, described in the following sections.

## 2.3. Problems

First of all, the reference regions will have different size (Table 1) depending on the search ratio ($v = 2 \cdot r + 1$).

| Reference region size ($v_r \times v_r$) | Search ratio ($r$) |
|---|---|
| 5 x 5 | 2 |
| 7 x 7 | 3 |
| 9 x 9 | 4 |
| 11 x 11 | 5 |

**Table 1**

Moreover, if $d$ is the maximal pixel displacement, the size of the search area size will be: $v_b = v_r + 2d$, and the number of possible situations at this region is: $(2d+1)^2$.

| Maximal pixel displacement ($d$) | Possible situations in search region | Search zone size $v_b$ (pixels) |
|---|---|---|
| 1 | 9 | $v_r + 2$ |
| 2 | 25 | $v_r + 4$ |
| 5 | 121 | $v_r + 10$ |
| 10 | 441 | $v_r + 20$ |

**Table 2**

The search and reference zone sizes are shown ($d=1$) in next figure (Figure 3)

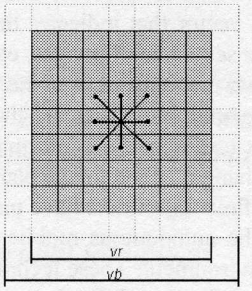
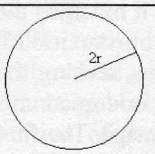



**Figure 3**

One limitation to this algorithm is that its complexity time grows in a quadratic way with the maximum possible displacement allowed for the pixel (Figure 4). Intuitively, as the tracked object speed doubles, the time used for its motion search quadruples, because the search area is equal to a circle centred at the pixel, with a radius equal to the maximal speed.

| Search zone | Search zone area | Explore time |
|---|---|---|
|   | $A = \pi \cdot r^2$ | $t_A$ |
|   | $B = \pi \cdot (2 \cdot r)^2 = 4 \cdot A$ | $t_B = 4 \cdot t_A$ |

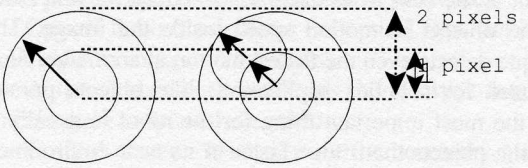



**Figure 4**

The search circle is usually approached by a square array of pixels of $(2r+1)$ side, for search radius r. If the image rate is $\Delta t$, both images in each motion to be tracked are taken at $t$ and $t+\Delta t$, and a two pixels radio search would need a 5x5 search zone and 25 evaluations. If we had the chance to take an additional frame in between the both original ones (at the time $t + \Delta t/2$), the original search would break

down in two consecutive searches of one pixel, one from the first image in the time $t$ to the second in $t+\Delta\ t/2$ and another from this to the third image in $t+\Delta t$. In this way a single 2-pixel search (the equivalent of four 1-pixel searches in search area) is reduced to two 1-pixel search. In a similar way, a search on an area of 3 pixels (9 times 1-pixel search) could decrease to three successive searches of 1 pixel, if extra images were available. It seems a good option to do the searches in a large number of small displacements (using multiple images, one for each search), rather than in a smaller number of large displacement searches. If this reasoning were taken to the limit, it would imply that the multiple combination of one pixel search is the best election, in terms of calculation process. Using this technique, the search for a given speed v requires a number of k pixels searches (being k a constant, usually 1) linear in v, in an opposite way to the original matching algorithm which requires a single, quadratic time v pixel search. This fact reduces the problem from a quadratic time to a lineal time.

In spite of every problems seems to be solved, the previously exposed is not the only problem. When a matching method is used to obtain the optical flow, as opposed to a gradient method [12], the smallest displacement that can be searched for is a single pixel, and usually is necessary to down sample the input image to a coarser resolution to be able to process it in real time. This means that the minimal distance that can be searched for is a single pixel, but with a relatively large size. Thus small motions can be omitted, and a large motion can also be missed if it was composed of many small displacements, each less than a pixel.

If the variable $d$ is fixed to the value of a pixel and the search of the speed motion is necessary, the only option is to allow the $t$ variable to change. Thus, instead of searching over distances to calculate the speed, with a fixed $\Delta t$, it will be necessary to search over time to discover the actual speed of the pixel, as it is shown at next section.

## 2.4. Tracking applications solution

In a basic approach, optical flow is calculated on a sequence of images as the one shown in the figure 5.

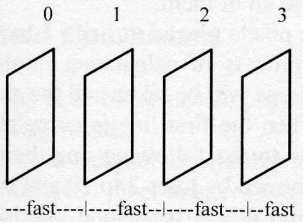



**Figure 5**

There is not any explicit delay between the images of the sequence (only the time needed to capture an image, which is a fixed quantity) and the motion detected, if any, it will correspond to a very fast motion. The noted image as "0" corresponds to the last one being acquired, "1" correspond to the previous one and so on. Since the search algorithm only operates on a fixed distance of a pixel, with this approach a very small range of speeds will be detected (either zero or a pixel/frame). This range of detection can be extent to other wider, varying the parameter $\Delta t$.

Therefore, the fastest motion that can be detected would be of one pixel in the plane image, between each images pair. If this motion is not limited and an object is moving faster than one pixel/frame, taken at 25 frames/seg (video rate), then it would be necessary to search for motion greater that one pixel/frame. However, this step is not necessary, for most practical applications, since the image is usually sub sampled to a coarser level for faster processing, and under these conditions each pixel is larger and it would correspond to a very fast motion at video rate.

If there is no problem to determine the fastest motion, it can be considered how to find smaller motions, using longer delays ($\Delta t$) between images. If no motion is detected between frame "0" and "1", the calculation process would be made between the frames "0" and "2" (previously stored in memory); if in this scale motion is detected, the tracking operation will be done an if it is not, the process would be repeated on frames "0" and "3", "0" and "4", etc., up to some limit for the search of the slowest speed.

It is necessary to mention that the motion detected at a slower scale could be motivated by a sudden, rapid acceleration, or more generally by any motion faster than the one that is currently being examined. This temporary aliasing can be detected proving that the slowest component in the motion is significant. For example, a search in the following scale (slower) to the mentioned scale doesn't indicate any motion. In this case, when a temporary aliasing or otherwise is not detected in that scale, the original slow motion is real. It can be useful for practical applications, to ignore this aliasing confirmation, since any fast motion will be determined correctly immediately later, when the algorithm looks for the fastest motion again, before the arrival of the following image.

This technique produces the effect of a better find of the objects that move faster and as these objects are the most interesting for a robot (for example to detect obstacles), this is an improvement on previous movement search algorithms.

The method is optimised to detect and track the fastest motion in the image, but it can also operate on a wide range of speeds. A pyramidal technique [4], [13], can be used to separate the smaller motion components from the faster motion. However, these techniques are only useful up to a point, because at the highest levels in the analysis the pixels are quite large, implying that there is a significant quantification error for both angular and magnitude resolution.

To generate a full optical flow field, not only for tracking applications, it would be necessary to search for various speeds, but it is not the propose of this physical implementation that searches for maximum speed.

## 3. Experimental description

It has been implemented in this job the initial version of the referred algorithm for searching the maximum speed at the image. It makes the following stages, to determine this speed, or displacement of a pixel, between a number of $(2d+1)^2$ possible displacements where $d$ is an integer value that depends on the speed:

1. Displacement: this stage obtains the pixel values of the displaced regions in the searched image (last one). The number of regions obtained is equal to the number of possible pixel displacements.
2. Comparison: the subtraction of the pixel values in the displaced regions from the reference image patch is made here.
3. Excitement: This stage is implemented with a " box-filter", which is an operator that performs a mean value of the comparison stage.
4. Election: The most resemblance displaced region with the reference one will correspond with that with smaller excitement stage values. Therefore this stage will find the smallest value of the excitement one.

### 3.1. FPGA implementation

Next figure (Figure 6) describes the logical design of the FPGA that implements the algorithm of optical flow calculation. The meaning of its blocks is as follows:
- The two RAMDP_8kb blocks are double port RAM memories (8 KB), used to store the data images temporally. They are necessary, since the frame grabber gives the information of the images in a sequential way and loading it is necessary to store some lines to carry out the process. Their size is of 16 lines of 96 pixels, out of 480 possible ones, for the temporal restriction of the time of calculation (see point 4). The device chosen for it is the CY7C09159V.

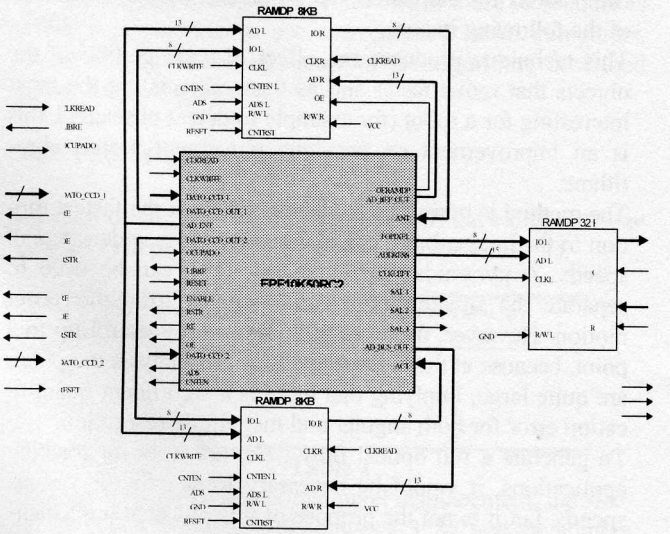



**Figure 6**

- The RAMDP block (32 KB) is the dual port RAM memory where the optical flow calculated vectors are stored on. The used integrated circuit is a CY7C09079V.
- The used FPGA contains all the necessary modules required for the optical flow calculation. They have been dismissed at figure 7 and they are the following ones:

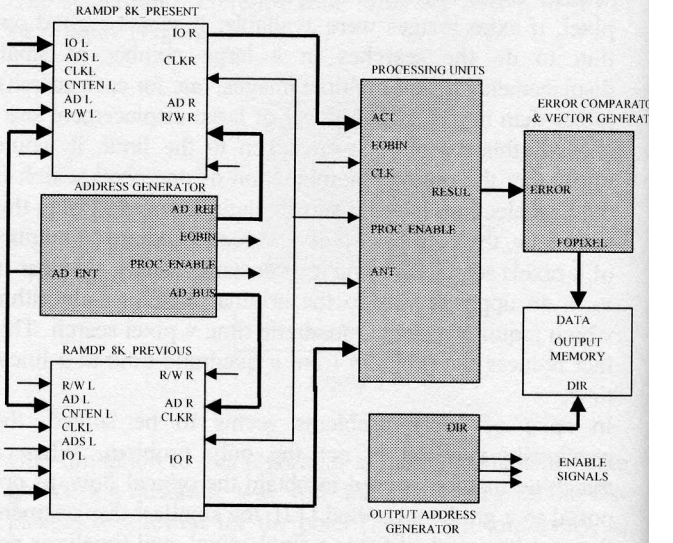



**Figure 7**

· *Address generator* of the positions where the values of the pixels are stored on (of the current and previous images), for the processor to be able to read them (displacement Stage).

. *Processing units* in charge of carrying out all the operations required by the algorithm, to obtain the coefficient of likeness (comparison and excitement stages). This block actives a signal to indicate other units when it has finished its job, to allow another line processing. It is formed by 9 blocks of elementary processors that calculate the error coefficients of the displaced regions. They are 9, coinciding with the number of possible positions that a pixel can be placed on with a 7x7 pixels reference block (2r+1 = 7) with maximal possible displacement of one pixel at every directions (d=±1 pixel). They are placed this way in order to the input data from the reference block can be carried over them while those of the search block arrive simultaneously to all of them.

First of all, the pixels placed at lines 1 to 9 are addressed and when the process of calculating minimal error for a pixel ends, next one will be taken and the same process will be repeated. When the first line is completed, the address generator points to the following one, being then the addressed block formed by lines 2 to 10 and the line 1 remain free for new data. When the second line is completed, the addressed block is formed by lines 3 to 11, etc.

With this approach, the data of line 17 will be hold at line 1, the line 18 at line 2 and so on, being the read and write process over the dual port memory simultaneous. When the last data on line 16 is being addressed on, the following lines are hold at the upper part of the memory, and the ad-

dress mode is made at a continually way, as it is shown at the following table.

| ADDRESSED LINES | FREE LINES |
|---|---|
| 9-10-11-12-13-14-15-16-1 | 8 |
| 10-11-12-13-14-15-16-1-2 | 8 and 9 |
| 11-12-13-14-15-16-1-2-3 | From 8 to 10 |
| 12-13-14-15-16-1-2-3-4 | From 8 to 11 |
| 13-14-15-16-1-2-3-4-5 | From 8 to 12 |
| 14-15-16-1-2-3-4-5-6 | From 8 to 13 |
| 15-16-1-2-3-4-5-6-7 | From 8 to 14 |
| 16-1-2-3-4-5-6-7-8 | From 8 to 15 |

**Table 3**

. *Error compare unit* that analyses all the coefficients of likeness of a region, detects the smaller from them and generates the optical flow vector for each pixel (election Stage).

The FPGA that has been used for the design is an ALTERA EPF10K50RC240-3, using 37 inputs and 75 outputs from it.
Two clocks are used in the system. One, named "CLKWRITE", is given by the interface with the camera CCD and has a 10 MHz frequency. The other one, "CLKREAD", is generated internally and its value is of 15.384615 MHz (or the maximal possible for the correct operation of all the processing units).

The output of the system is placed on the right port pins of the of the dual port memory RAMDP (32 KB) that holds the optical flow vectors that have been obtained, together with the three necessary enable signals ("out 1", "out 2" and "out 3") that are used by the later prosecution stages to read safely from this memory without writing interference. The optical flow of three images is stored, and the enable signals indicate if the memory is ready for being read ("out1" enable the reading of the image 1, and so on.). The format of the flow of each pixel is a byte that holds the *(u, v)* coordinated (flow in *x*-direction and *y*-direction, respectively), with a two complement format number of 4 bits, that expresses the possible results (+1, 0, -1) in each address.

## 4. Results and Conclusions

This process has been implemented physically on a FPGA, with the following results:
· An interface with a digital camera that stores two or more monochrome 480x480 pixel images (two in the beginning) is used. More frame memories should be included in order to implement the detection of several speeds. The access to these memories is sequential, beginning for the first pixel of the image. A protocol has been used to prevent the reading of an area that is being updated for the acquisition system.
· Looking for real time operation of the algorithm, the initial size of the images has decreased in a relationship 5:1 (Figure 8). In a practical system this approach can be solved with a image sub sample (with the log-polar transformation for instance) or with parallel process of several optical flow operators in each region of the image, or with the positioning of the camera, on the part of an active vision system, in the area where motion presence has been detected previously, to improve the details.

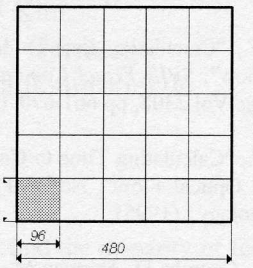



**Figure 8**

· The logical programmable device used (Altera FPGA) can work with a 65 nseg clock. (15,384615 MHz) that is able to operate on images of the size mentioned before up to a of 22,56 frames/seg frequency, very near the 25 imperative frames/seg of a real time application. Light modifications on the implementation architecture (or the use of a faster FPGA) are pretended, to be able to arrive to that frequency, without many problems.

Some results of flow fields generated at real time, will be shown very soon, when a development vision system, that is actually been in construction, was finished. This system has a hardware structure very similar to the exposed in this work, with memory blocks at input and output and a FPGA to hold the calculus. The reasons to use this hardware elements are two, first the speed of the process, bigger than the one produce for a DSP solution (and of course for a software implementations.) and second for the flexibility of the FPGA system, that can be programmed iteratively at an easy way to obtain the correct results.

## 5. References

[1] Cobos P., Monasterio F., "Fpga implementation of the Horn & Shunk Optical Flow Algorithm for Motion Detection in real time Images". *Dcis '98 Proceedings, XIII Design of circuits and integrated systems conference*, pp: 616-621. (1998).

[2] Cobos P., Monasterio F., "Fpga implementation of a Log-polar Algorithm for real time Applications". *Dcis '99 Proceedings, XIV Design of circuits and integrated systems conference*, pp: 63-68. (1999).

[3] Barron, J.L., Fleet, D.J., Beuachemin, S. S., "Systems and Experiment Performance of optical flow techniques". *International Journal of Computer Vision*, Kluwer Academic Publishers. Vol. 12, N° 1, pp: 43-77. (1994).

[4] Camus, T., "Applications of Pyramid Structures to Multiescale Optical Flow". *Brown Technical report CS-90-09.* (1990).

[5] Camus, T., Bülthoff, H., "Space-Time Tradeoffs for Adaptive Real-Time Tracking". *SPIE Mobile Robots VI.* (1991).

[6] Camus,T., "Real-Time Optical Flow". *PhD Thesis*, (1994).

[7] Camus, T., "Calculating time to collision with real-time Optical Flow". *SPIE Visual Communications and IMAGE Processing.* Vol 2308, pp 661-670. (1994).

[8] Camus, T., "Calculating Time to Contact Using Real Time Quantized Optical Flow". National Institute Of Standards and Technology. (1995).

[9] Camus, T., Coombs D., Herman M. and Hong T.H., "Real-Time Single Workstation Obstacle Avoidance Using Only Wide Field Flow Divergence", *Proceedings of the International Conference on Pattern Recognition*, Vol. 3, pp: 323-330. (1996).

[10] Camus, T., Szabo S., Coombs D., Herman M. and Hongchi L., "A real-time computer vision platform for mobile robot applications". *Real-Time- Imaging*, Vol.2, N °.5, pp: 315-27. (1996).

[11] Camus, T., "Real-Time Quantized Optical Flow". *The Journal of Real-Time Imaging (special issue on Real-Time Motion Analysis), Academic Press,* Vol 3, pp: 71-86. (1997).

[12] Horn, B., and Schunck, P., "Determining Optical Flow". *Artificial Intelligence,* Vol 17, pp: 185-203. (1981).

[13] Perrone, J., "Simple technique for optical flow estimation". *Journal of the Optical Society of America*, Vol. 7, N° 2, pp: 264-278. (1990).

[14] Altera. *Data Book.* (2000).