Real-time Optical-flow Computation for Motion Estimation under Varying Illumination Conditions

Julio C. Sosa[1], Roberto Rodríguez[2], Víctor H. García Ortega[1] and Rubén Hernández[1]
[1]Department of Post-degree, Superior School of Computation, National Polytechnic Institute (IPN), Mexico
[2]Digital Signal Processing Group, Institute of Cybernetics, Mathematics & Physics (ICIMAF), Cuba
Email: [jcsosa, vgarciao, rhtovar]@ipn.mx, rrm@icmf.inf.cu

Article Info

Article history:

Received Jan 28, 2012
Revised Mar 21, 2012
Accepted Mar 26, 2012

Keyword:

estimating object motion
FPGA
image sequences
optical flow approach
varying illumination conditions

ABSTRACT

The optical flow approach has emerged as a major technique for estimating object motion in image sequences. However, the obtained results by most optical flow techniques are poor because they are strongly affected by large illumination changes and by motion discontinuities. On the other hand, there have been two thrusts in the development of optical flow algorithms. One has emphasized higher accuracy; the other faster implementation. These two thrusts have been independently pursed, without addressing the accuracy vs. efficiency trade-offs. The optical flow computation requires high computing resources and is highly affected by changes in the illumination conditions in most of the existing techniques. In this paper, a new strategy for image sequence processing is proposed. The data reduction achieved with this strategy allows a faster optical flow computation. In addition, the proposed architecture is a hardware custom implementation in EP1S60F1020 FPGA showing the achieved performance.

Corresponding Author:

Julio C. Sosa
Department of Post-degree, Superior School of Computation,
National Polytechnic Institute (IPN),
Mexico
Email: jcsosa, vgarciao, rhtovar]@ipn.mx

1. INTRODUCTION

The optical flow approach has emerged as a major technique for estimating object motion in image sequences [1]. Additionally, novel theoretical analysis of motion and optical flow estimation had appeared [2], [4] and [3]. However, the results obtained by most optical flow techniques are poor because they are adversely affected by large illumination changes and by motion discontinuities. Recently however, there have been two trends in the development of optical flow algorithms. One has emphasized higher accuracy; the other faster implementation. These two trends have been independently pursed, without addressing the accuracy vs. efficiency trade-off [5].

The optical-flow computation consists on the estimation of the apparent 2D movement field in the image sequence, as introduced by Horn and Schunck [6]. In this way, each pixel has an associated velocity vector. This technique can be combined with several segmentation techniques in order to improve its accuracy or implement object tracking. Many strategies for optical-flow computation have been published [7]. Among these methods, the gradient-based and the correlation-based approaches are the two most commonly used techniques.

Most of the recent advances in optical-flow computation have focused their improvements on achieving a high accuracy. Examples of these research fields can be found in [8] and [9]. Moreover, there are few works that focus on the computation speed aspects, like the achievement of a faster speed with real time constrains. Additionally, innovative theoretical analysis of motion and optical flow estimation encourage the use of custom electronic devices [10], [11] and [12].

The classical approach for image sequence analysis usually involves full image processing. In the optical flow computation the spatial and temporal derivatives are calculated for all pixels on all images, despite the fact that images could have suffered minor changes from one frame to the next.

The iterative form for optical flow computation can be implemented by software on a general purpose microprocessor. However, in order to process image sequences in real time it is interesting to use programmable devices, ASIC, analog integrated circuit VLSI [13], clusters of processors [14] or special processors [15], though these last solutions are not practical as they are very expensive.

FPGAs have been chosen to develop a cheaper an easier prototype. A custom architecture of this type was proposed by Arribas & Monasterio [16]. They processed 50×50 pixels, images at 19 fps using only three iteration cycles per each pair of images. Initially, the default result is assumed as a zero value for all optic flow field vectors per iteration cycle.

In a more recent work, Martin, et al., [17] show a new optical flow computing technique. In this work, the iterations are made among a group of successive images in an image sequence, not only between a pair of two consecutive images. In other words, a single iteration per image pair is performed and the results are used as input for a second iteration, but now with a new pair of images. This idea is based on the assumption that changes between two consecutive images are negligible and the results obtained are useful for an iterative process. Martin et al. present results after processing 64 images, but using only one iteration per pair of images.

Finally, Sosa [18] shows a new optical flow computation architecture, with change-driven data flow processing. In that the work it was shown the systems effectiveness depends on the percentage of static pixels whose variation intensity is below the threshold, which was a fixed threshold, processing 256×256 pixels, images at 80 fps using ten iteration cycles per pair of images.

It is important to know that in real scenes, there will be inevitable noise that can randomly change pixel intensity values. For this reason it is necessary to use dynamically threshold adjustment that will significantly reduce the problem. Robustness to varying illumination conditions is achieved by an innovative technique that combines a gradient-based optical flow method, change-driven processing and an adaptive threshold.

This paper uses the same principles of the three works described before in [10], [16] and [17]: image changes between two consecutive images are minor and the image intensity change appears due to local pixel movement. In this way, if there are not intensity changes in any consecutive images pair pixels, then there is no movement, and these images will not be processed.

The system has been developed for implementation on an Altera development board, which includes a Stratix EP1S60F1020 FPGA and 256 MBytes of DDR SDRAM memory. The Overall system was designed in VHDL and has been simulated using ModelSim.

2. THEORETIC ANALYSIS

In this section, the original problem for optical-flow computation according to Horn & Schunck's [6] algorithm is considered, and an innovative technique for optical flow computation is introduced.

2.1 Horn & Schunck Optical Flow Algorithm

According to [6], and discussed in [18], the Optical Flow computation is obtained by:

$$u^{n+1} = \hat{u}^n - \frac{E_x(E_x\hat{u}^n + E_y\hat{v}^n + E_t)}{(\alpha^2 + E_x{}^2 + E_y{}^2)}, \qquad (3)$$

$$v^{n+1} = \hat{v}^n - \frac{E_y(E_x\hat{u}^n + E_y\hat{v}^n + E_t)}{(\alpha^2 + E_x{}^2 + E_y{}^2)}, \qquad (4)$$

where $\hat{u}$, $\hat{v}$ are the Laplacian of velocities *u*, *v* and α is an adjust parameter and *Ex, Ey* and *Et* are the partial derivates of image intensity related to *x*, *y* and *t* respectively. These equations have an iterative form because *u* and *v* are function of $\hat{u}$ and $\hat{v}$. If the iteration number is much bigger, the result will be more accurate, but it will also increase the calculation time. The equations (3) and (4) show an iterative dependence, thus the result depends on the previous calculation.

The method used, by Horn and Schunck [6], to determine a measurement of *Ex, Ey* and *Et* is:

$$E_x = \frac{1}{4}\{E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+}$$
5)

$$E_y = \frac{1}{4}\{E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k} + E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+}$$
6)

$$E_t = \frac{1}{4}\{E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k} + E_{i,j+1,k+1} - E_{i,j+1,k} + E_{i+1,j+1,k+}$$
7)

where the column index j corresponds to the x direction in the image, the row index i to the y direction, while k lies in the time direction.

On the other hand, the approach used for the calculation of the Laplacians of u and v is:

$$\hat{u}_{i,j,k} = \frac{1}{6}\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\} + \frac{1}{12}\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+}$$
8)

$$\hat{v}_{i,j,k} = \frac{1}{6}\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\} + \frac{1}{12}\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+}$$
9)

It is necessary to obtain the spatial and temporal gradient values for computing optical-flow. To do this, at least two consecutive images of the sequence are required.

2.2 Change-Driven Image Processing

The change-driven data flow processing has been applied before as part of another image processing algorithm, by software [19] and by hardware [18]. The principle is based on two considerations; an image pixel, representing a point on an object, does not change its brightness value from an instant of time *t* to the next instant of time *t+δt*, and the images usually change a little from frame to frame, especially if the acquisition time is short. This justifies the existence of a module that locates only the pixels that have changed. The idea of such a module is biologically inspired. The eye can work over a large range of luminance levels [20]; it must also be able to handle the different rates of change in luminance. In the spatial domain, spatial vision can be characterized by the contrast sensitivity function (*CSF*).

In this work the parameter Change Sensitivity Threshold (*CST*) is used. This condition can be expressed as follows: when the pixel intensity level difference between two consecutive images is less than a threshold, then this pixel will not be processed. This restriction can be written with the equation:

$$Ch = \{ \begin{matrix} 1 & mag > CST \\ 0 & mag \leq CST, \end{matrix} \qquad (10)$$

where *mag* is the grey level difference, *CST* is the threshold, an integer value that is always small. Detected change *ch* will be 1 if there is a change in the pixel or 0 if not. With small values for the threshold many pixels will be detected as moving pixels and there will not be a big time reduction. On the other hand, with higher values for the threshold a major speed-up will be achieved, but with a loss of accuracy.

The change-driven image processing theoretical speed-up was estimated by software before the hardware implementation. The formal optical flow and change-driven optical flow algorithms have been implemented using C++.

The algorithm is significantly different, and subsequently, the computing resources needed (and therefore the hardware required) are reduced when change-driven processing is applied. A look-up table (LUT) is necessary to compare two consecutive images and detect which pixels have changed on the images. Only pixels that have changed (above *CST*) initiate the corresponding processing instructions.

The number of clock cycles needed to implement the original optical flow algorithm can be expressed as:

$$cycles = 8(m\times n) + k(16(m\times n)+5(m\times n)), \tag{11}$$

where m×n is the image size and k is the number of iterations. The first term indicates the cycles need to estimate the partial derivatives, the second term indicate the cycles needed to estimate the velocities and Laplacian. The optical flow modified algorithm computation cost is represented as:

$$cycles = 3(m\times n)+ 8\,\rho\,(m\times n) + 2k\rho(16(m\times n)+ 5(m\times n)), \tag{12}$$

where ρ is defined by:

$$\rho = 1 - \frac{\eta}{(m\times n)} \tag{13}$$

and where η is the number of pixels that does not change. There will be a few pixels that change and speed-up the optical flow computation if the *CST* is high. The *CST* selection is a very important step. It is possible to conclude that the *CST* is a critical value in change-driven processing algorithms, a too low value may include spurious changes, while a too high value will erase significant scene changes.

3. CHANGE SENSITIVITY THRESHOLD SELECTION

The Change Sensitivity Threshold (*CST*) selection is a very important step. The focus in this section is to determine what *CST* value is the best, and the strategy for *CST* computation on line. In [18] there is an example of the change-driven image processing policy applied to the optical flow computation where an empirical fixed CST was used. The experiments performed, with constant brightness, recommend a *CST* less than five for obtaining useful results in the optical flow computation. It was shown that the system effectiveness depends on the percentage of static pixels whose variation intensity is below the *CST*.

It is very important to make out, that: the Horn & Schunck's formulation assumes that an image pixel, representing a point on an object, does not change its brightness value from an instant of time t to the next instant of time $t+\delta t$. However, in a realistic scene this is almost never the case. A pixel can change its brightness value because an object moves or because the illumination of the scene changes. As a result, this noise will give a number of false positives that will reduce system accuracy and performance. However, dynamically adapting the *CST* significantly reduces the problem, and keeps the Horn & Schunck formulation.

Some image sequences are used for motion estimation under varying illumination. The utilized sequences are commonly used as a test bench for optical flow algorithm evaluation purposes. They have been downloaded from the Computer Science Department of the University of Western Ontario[1] and from the Computer Vision Research Group at the University of Otago[2]. Each sequence used here is representative of three common situations in image processing: the *taxi* sequence has been acquired in an outdoor environment without controlled illumination conditions; the *Rubic* sequence has been taken from an indoor environment with controlled illumination, and *sphere* sequence has been generated artificially, without the noise introduced by most cameras (synthetic sequence). All sequences are in B/W, with 256 grey levels, as shown in figure 1. The pixel intensity level is given as an 8-bit binary number. The minimum intensity difference is 1 in this case. If the *CST=0*, then all the pixels are processed and there are no benefits in using change-driven processing. The last case will never be used because it indicates that the two consecutive images are completely different.

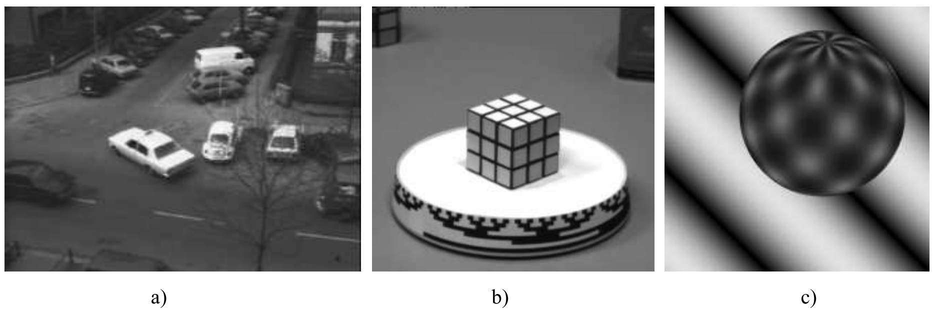
A small study of the three sequences showed that there are changes. Considering 11 image pairs (of each sequence), about 70% of pixels do not change. The *sphere* sequence (synthetic type) showed that the percent of pixels without change is constant. It is possible to see this in figure 1. An opposite behavior is presented by the sequence *taxi*. It is a natural sequence, which showed the smallest percent of pixels without change and has an erratic behavior. The last sequence (*Rubic*), is an intermediate point between the two previous sequences.

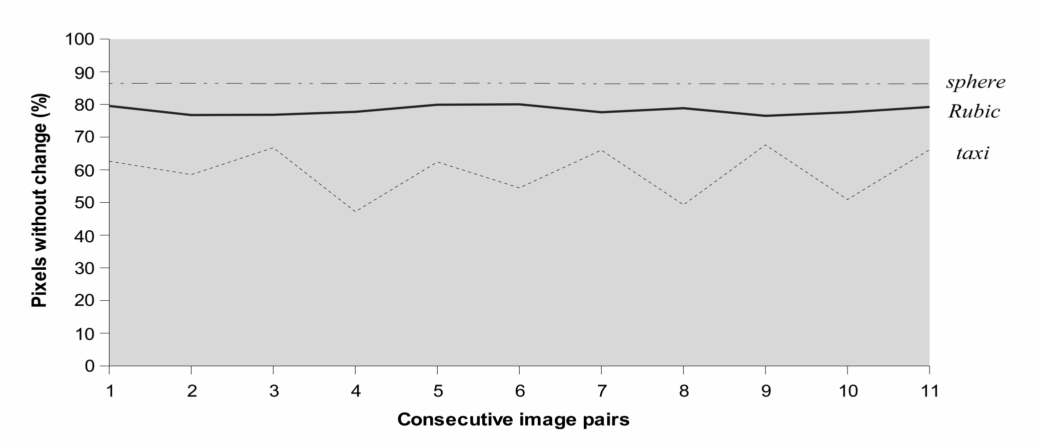
The studies indicate that the synthetic sequences, such as: sphere, square2, translating and diverging tree have a percent of pixels without change near to the constant. Then it is necessary to use only a CST = 1. The

[1] ftp.csd.uwo.ca/pub/vision.

[2] http://www.cs.otago.ac.nz/research/vision/.

natural sequences are more complex. The *CST* selection depends on environment. It was necessary to develop a strategy to dynamically adapt the *CST*.





d)

Figure 1. Sequences used and their changes in 11 image pairs.

The selection of an appropriate algorithm is not an easy choice since each algorithm makes different assumptions about the image content or environment characteristics.

Our approach takes the average â of gray level pixel set. Ideally, the whole image should be an average in order to compute the image average grey level, but this is not necessary. The principle is simple. If there are not changes between two consecutive images, that is, the brightness is constant and there is not movement, then the grey level average must be the same for each image. However, if there is not movement but there are illumination or brightness changes then the grey average is different between two consecutive images. This difference will determine the *CST* value used to implement the change-driven image processing.

Then, it is possible to take several pixels (not all image) to compute the image average grey level. Therefore, the difference between two consecutive averages (of two consecutive images), will indicate the appropriate *CST*.

In [20] the grey level average was computed by:

$$\hat{a} = \sum_{i}^{m}\sum_{j}^{n}\frac{p(i,j)}{m \times n}, \qquad (14)$$

where $m \times n$ denote the grid dimension, i and j are the pixel coordinates (they are not consecutive pixels, $i, j = 1k, 2k, \ldots k \geq 1$, if $k = 1$ all the image is average). The grey level average was computed off-line.

In this work the grey level average was computed on line and by experimentation with different image sequences in order to determine the total amount of pixels needed. Then, it was necessary to consider: 1) the memory accesses are in bursts of 32 bytes, 2) an accumulator register was used, 3) the pixel set must be distributed in each image window, 8 windows, of 32 × 32 pixels were used, in each image and 4) the easiest way to compute the value was to obtain a pixel set with a power of two numbers of pixel. Then, the integer division consists of only right shifts.

4. ARCHITECTURE

The diagram of the hardware platform using a Stratix PCI development board, which uses a Stratix EP1S60F1020 FPGA to implementing the optical flow computation strategy discussed in the previous sections, is shown in Figure 2.

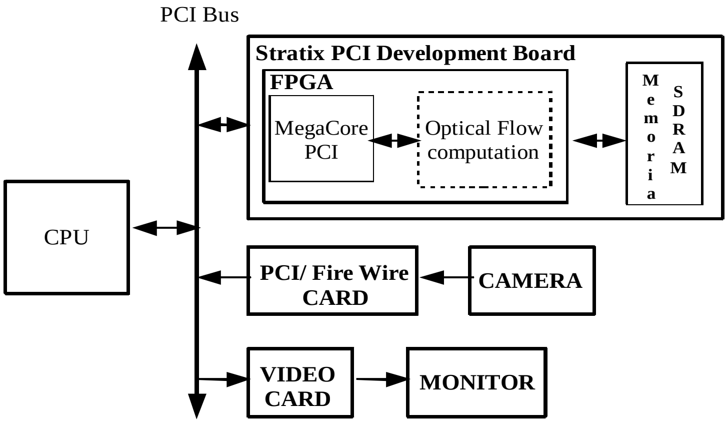


Figure 2. Diagram of the hardware platform using a Stratix PCI development board and other components used by the system.

The FPGA architecture has been divided into five modules: threshold selection module, Gradient/LUT module, data selection module, velocities module and finally the Laplacian module. The modules have been described in [18]. The main changes, in this work, were the Treshold Selection Module and Data Selection Module.

The optical flow algorithm requires a large FIFO memory that can be implemented in the FPGA. The EP1S60F1020C6 has 5,215,104 RAM bits.

It must be noted that the calculations made with the hardware modules are implemented using integer arithmetic, since it requires less resources than a floating point approach. The divisions have been normalized to the power of two and implemented as a right shift. The idea is to optimize the use of resources and to increase the speed of the system.

4.1 Threshold Selection Module

The threshold selection module computes the average pixel set grey level. The pixel amount is a power of two. Thus it is not necessary to implementing a division, which is a time occupying operation. The pixel set was fixed as a grid of 8×8 pixels uniformly distributed along the image. That allocation gives a total of 64 pixels per image. This module architecture is very simple. A fast-carry adder and a right shifter have been employed.

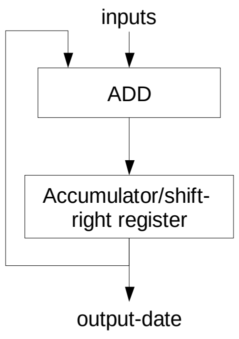
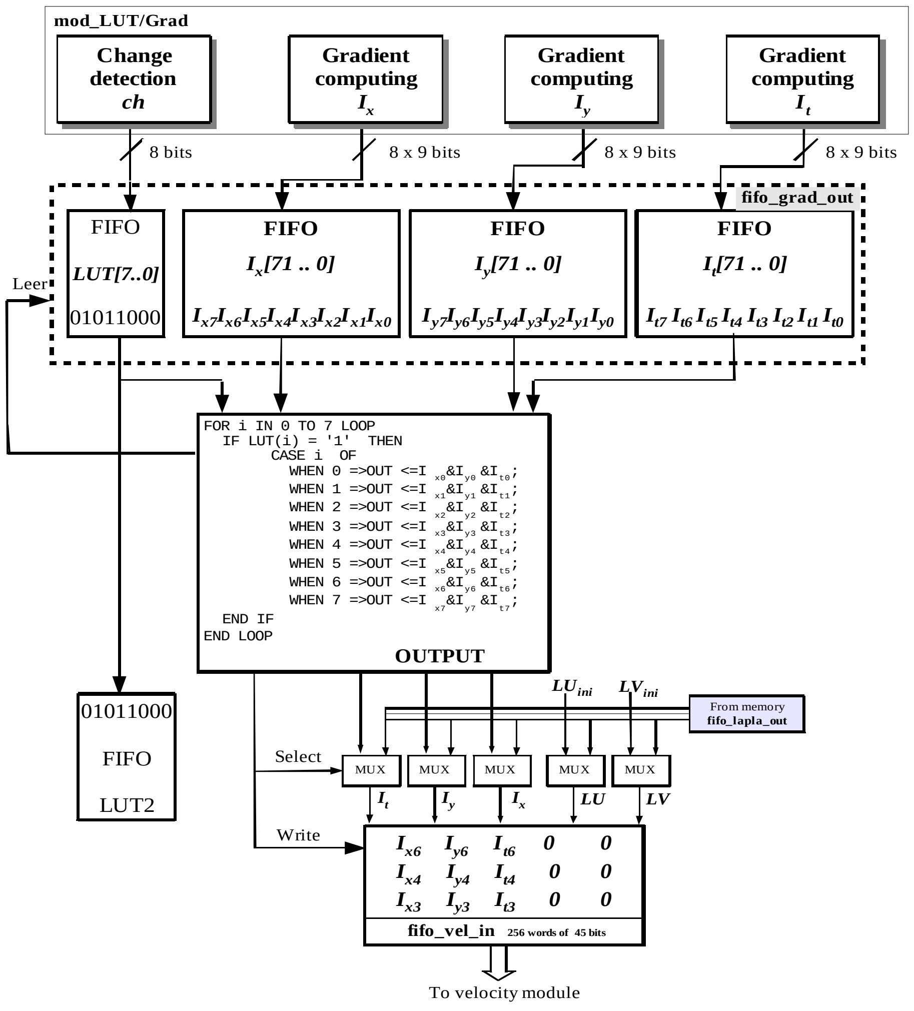


Figure 3. Threshold module block diagram.

The latency of this module is 5,130 clock cycles at 166 MHz. This time is necessary to read the pixels from memory. The access memory frequency is 133MHz. The *CST* is computed and sent to the Gradient/LUT module so the change table (LUT) can be built. It is necessary an accumulator register to add the read pixels, in this stage, see figure 3. At the end of this process a shift-right implemented division must be executed. Because of this, it is necessary that the pixels number be a power of 2.

4.2 Data Selection Module

The outputs of gradient *Ex, Ey* and *Et* must be 72 bits, to allow a signed word assuming each pixel is 9 bit (2 complement), as shown in figure 6. Three FIFO memories with 32 words of 216 bits each, are used for storing the outputs. At the same time the output LUT is stored in a 32 × 8 FIFO memory. The LUT access and data selection are driven by the selection controller module. The data selection module reads the LUT in order to know which of the pixels has changed so the selection of *Ex, Ey* and *Et* components can be started. Then, the data processing begins when the first LUT word is read. An example of this is shown in figure 4 when the first word is LUT = [01011000], note that only three pixels have changed LUT(6), LUT(4) and LUT(3). This indicates that *(6,4,3)* components from *Ex, Ey* and *Et* were selected, also that the initial laplacian values (LU and LV) were introduced in order to compute the speed, all the results are stored in a new FIFO with 256 words of 45 bits. When the reading of each word from the LUT is completed, each word is stored in a new FIFO memory called LUT2.


mod_LUT/Grad
Change detection ch
Gradient computing I_x
Gradient computing I_y
Gradient computing I_t
8 bits
8 x 9 bits
8 x 9 bits
8 x 9 bits
fifo_grad_out
FIFO LUT[7..0] 01011000
FIFO $I_x[71 .. 0]$ $I_{x7}I_{x6}I_{x5}I_{x4}I_{x3}I_{x2}I_{x1}I_{x0}$
FIFO $I_y[71 .. 0]$ $I_{y7}I_{y6}I_{y5}I_{y4}I_{y3}I_{y2}I_{y1}I_{y0}$
FIFO $I_t[71 .. 0]$ $I_{t7}I_{t6}I_{t5}I_{t4}I_{t3}I_{t2}I_{t1}I_{t0}$
Leer
OUTPUT
LU_{ini}
LV_{ini}
From memory fifo_lapla_out
01011000 FIFO LUT2
Select
MUX
MUX
MUX
MUX
MUX
I_t
I_y
I_x
LU
LV
Write
I_{x6} I_{y6} I_{t6} 0 0
I_{x4} I_{y4} I_{t4} 0 0
I_{x3} I_{y3} I_{t3} 0 0
fifo_vel_in 256 words of 45 bits
To velocity module

```
FOR i IN 0 TO 7 LOOP
  IF LUT(i) = '1'  THEN
       CASE i  OF
         WHEN 0 =>OUT <=I x0&I y0 &I t0;
         WHEN 1 =>OUT <=I x1&I y1 &I t1;
         WHEN 2 =>OUT <=I x2&I y2 &I t2;
         WHEN 3 =>OUT <=I x3&I y3 &I t3;
         WHEN 4 =>OUT <=I x4&I y4 &I t4;
         WHEN 5 =>OUT <=I x5&I y5 &I t5;
         WHEN 6 =>OUT <=I x6&I y6 &I t6;
         WHEN 7 =>OUT <=I x7&I y7 &I t7;
  END IF
END LOOP
```

Figure 4. Data selection module.

One additional function of the selection controller module is to read the essential components and store them in the *FIFO_vel_in* memory so the next iterations can be performed.

5. RESULTS

In this section the optical flow computation performed by the hardware system is compared to a floating-point software implementation. The hardware modules are implemented using integer arithmetic. Additionally, a study was made to analyze the accuracy of the model proposed in relation to the original Horn & Schunck model (without the change-driven policy) using 10 iterations. The algorithm and images sequences are available from (ftp://ftp.csd.uwo.ca/pub/v3ision/) and from http://www.cs.otago.ac.nz/research/vision.

5.1 Hardware Platform

This design was implemented on the Stratix PCI development board, with a FPGA (EP1S60F1020). The device has 57,120 Logic Elements (LE), 144 digital signal processing (DSP) block with (9-bit × 9-bit) embedded multipliers and 5,215,104 memory bits. Additionally, the development board has 256-MByte of external PC333 DDR SDRAM. It is very important because when using for computer vision task, high-speed memory is critical.

All the architecture has been developed in VHDL. Several tests were performed before synthesis to validate the design. Different tools were used depending on the part of the implemented system being considered.

First the processing modules included into FPGA were tested employing Quartus II software. Next the system was simulated including the SDRAM modules and the pci_mt32 MegaCore function [AUG 04]. In this stage synthetic images were used as well as real image sequences. Finally the overall system was synthesized with the PCI and SDRAM controller restrictions.

The whole design utilized 17, 650 Logic Elements (31% of the total 57,120), 3,199,874 RAM block bits (61% of the total 5,215,104) and 8 DSP block 9-bits elements (6% of the total 144).

5.2 Synthetic Image Sequences

The main advantages of synthetic inputs are that 2-D motion field and scene properties can be controlled and tested in a methodical fashion. In particular, it is possible to access the true 2-D motion field and quantifying performance.

This design was tested on two synthetic sequences to show its effectiveness. The use of synthetic sequences indicates that *CST* must be minimum (that is 1), as explain in section 3. Both sequences (*square2* and *treed*) have an established ground truth and are commonly used for benchmarking. A bit level simulation coded in MATLAB was programmed to evaluate the algorithm’s accuracy. In all cases 10 iterations was used.

The first sequence tested was the *square2* sequence; the used frames were 6 and 7. The angular error of the *square2* sequence was 2.3º, in hardware implementation. But, in the standard deviation was 9.9º, is best respect to full processing software implementation, as can be seen in table 1.

A more complex synthetic sequence was tested, the *Diverging Tree* sequence. In this sequence, the camera moves along its line of sight; the expansion focus is at the centre of the image scene, and image speeds vary from 1.29 pixels/frame on left side to 1.86 pixels/frame on the right; the used frames 20 and 21. In this sequence the relative angular error was 8.06º and the relative St. Dev was 2.0º. It is not a best accurate, the reason is that only 10 iterations were executed and the operations were performed using integer arithmetic.

Table 1. Software and hardware implementation error comparison. Optical Flow Full Processing (OFFP), Change Driven Hardware Optical Flow.

Sequence	OFFP		CDHOF		
	A. Error	St. Dev.	A. Error	Relative Error	St. Dev.
Square2	55.22º	11.19º	57.60º	2.37º	9.90º
Treed	13.56º	11.79º	21.62º	8.06º	13.80º

5.3 Real Image Sequences

Three real sequences were also used to test the performance of the proposed algorithm. There are two types of tests: a) movement and illumination changes and b) no movement but with illumination changes.

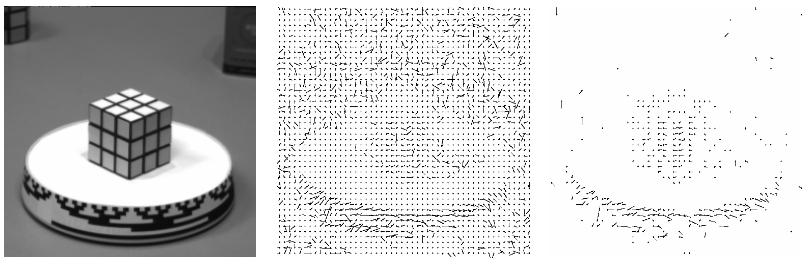
There is movement and there are illumination changes.

Figure 5 shows: 5a) *Rubic* sequence; that used the frames 6 and 7, 5b) *tapec* sequence; that used frames 40 and 41. In each sequence is computed the OFFP (by software) and CDHOF (by hardware).

Traditionally, in order to obtain the calculated optical flow vectors by software, it is necessary to process all the pixels from two consecutive images. That is why the results show a great amount of optical flow vectors. Nevertheless, many of these vectors do not mean real movement and its existence is the result of illumination or brightness changes. In this work, change driven image processing (CDIP) is used in conjunction with an innovative hardware architecture, so the amount of optical flow vectors representing real object movement is a major proportion.

It is possible to see that the optical flow computation by software is denser per image than the optical flow computation by hardware. Many of those vectors from the software computation are the result of noise and interference, and their presence puzzles the clear identification of moving objects within the scene.

a)



b)

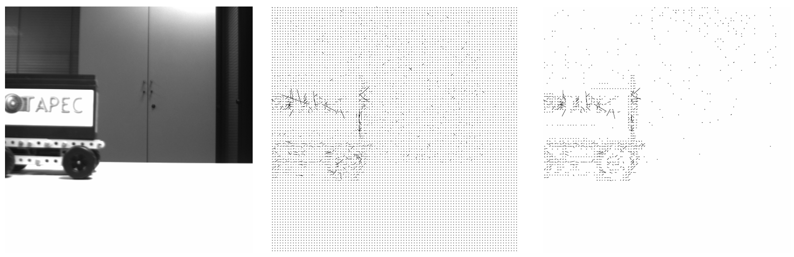


Figure 5: Shows a frames of the Rubic sequence and tapec sequence, and its calculated optical flow vector, left to right, by software and hardware computation.

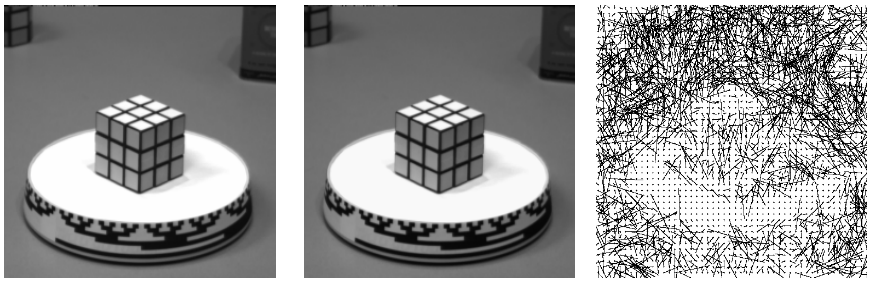
On the other way, there are a few optical flow vectors resulted from the hardware architecture computation, and the major portion of them are produced by the moving objects. Additionally, it is possible to observe that the achieved image is best for distinguishing the moving objects.

As with the above two sequences, the density of optical flow vectors is greater in the results obtained by software than those obtained by hardware. However, many from the software computation do not represent real movement but apparent movement because of illumination changes. The results from hardware computation show few optical flow vectors and the moving object are easily detected.

There is no movement and there are illumination changes.

The *rubic* sequence was used for the first testing developed in order to evaluate such a situation based on a known image sequence. The sequence is made of two versions of the same image, one raw and the other modified with a different level of intensity, with the same amount of pixels. So, the sequence has no movement but illumination changes as can be seen in figure 6. This modification was made reducing each intensity pixel from the raw image by 5, achieving a second image with apparent illumination changes but without movement. This testing was also useful for evaluating the performance of the hardware architecture developed for processing change driven optical flow, as well as the behavior of the module for detecting illumination changes and adjusting the *CST*.

Figure 6a) *Rubic* sequence, shows the result from the optical flow computation by software. The result obtained by hardware is a white image because the processing was not performed according to the working of the module for detecting illumination changes, correctly adjusting the threshold. This module worked with high efficiency because the characteristics of the test. As the illumination changes were applied in the same quantity for all pixels, the module generates an optimum mask for the adequate calculation of the threshold.

a) 

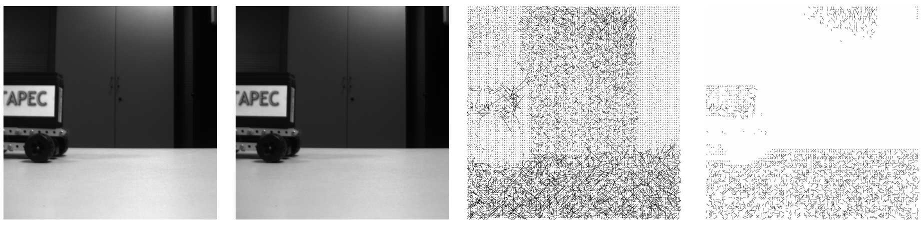
b) 

Figure 6: a) There is an illumination variation but no movement. Left to right: original image, modified image and its calculated optical flow vectors by software and b) Shows two consecutive images, of tapec sequence, and their calculated optical flow vector, left to right, by software and hardware computation.

More complex testing was implemented in order to evaluate the performance of the proposed system. Multiple sequences were captured and selected with notorious on purpose illumination changes but without movement at all. Figure 6b) shows two images (15 and 16) from sequence *TAPEC*. This figure also shows the optical flow vectors obtained by software and hardware computation. It can be seen that the optical flow obtained by hardware computation has a fewer number of optical flow vectors and the deployments are minor, considering its magnitude, than it has obtained through traditional implementation (by software computation)

5.4 Speed-Up Comparison

In this section a computational cost comparison is made. In order to evaluate the profits of using change driven image processing instead of full processing, three sequences were analyzed (*rubic, sphere and taxi*). The most important result is that the processing time is dramatically reduced. The Table 2 shows the speed-up comparison between the original model OFFP (100% temporal cost) and the proposed model.

Table 2.Speed-Up comparison, Optical Flow Full Processing (OFFP) and Change Driven Hardware Optical Flow (CDHOF).

Sequence	Rubic	Sphere	Taxi
OFFP	100 %	100 %	100 %
CDHOF	9 %	14 %	12 %

It can be seen that the processing time is reduced considerably. Even in the case of the sphere sequence the processing time is reduced by 14 % of the time needed for the optical flow full processing, by using the original model which processes all the pixels in the images. It is possible to note that this processing time reduction is a conjunct result considering as change driven processing as hardware computation.

According to obtained processing times considering several image pairs, and calculating time average, it is possible to say that the change driven optical flow processing in conjunction with the innovative hardware architecture can process up to 71 fps taking into account the taxi sequence, and up to 73 fps taking the other sequences. This is shown in Table 3.

Table 3. Processed frames per second.

Sequence	Image size	fps	CST ≥
Taxi	190 × 256	71.94	3
Rubic	240 × 256	73.42	2
Sphere	200 × 200	77.33	1

The table 4 shows the performance comparison among some architectures published in recent years including the present proposal. The four architectures are using the same H & S algorithm [4]. It can be seen here that the proposed architecture has a reduction of the number of processed frames per second which is small compared with the fastest of all [18]. Nevertheless, the architecture proposed in this work has the advantage of adaptivity to changes in illumination intensity.

Table 4. Comparison with other works

Works	Image size	fps	Iteration number
Arribas et al.	50 × 50	19	3
Martín et al.	256 × 256	60	1
Sosa et al.	256 × 256	≈82	10
Proposed here	256 x 256	≈74	10

It is important to note that the number of iterations performed in the three other works is different. When using a greater number of iterations the error diminishes. Cobos et al. [16] reports the use of three iterations. Martin et al. [17] reports only one iteration and storage of the resulting optical flow so it can be used by the next Laplacian computation. This is performed for 60 frames before showing systems results. The processing of frame number 61 requires the calculation of a new set of optical flow vectors. Finally [18] reports 10 iterations but, uses a change driven processing with a fixed threshold.

6. CONCLUSION

The change-driven image processing strategy presented in this work allows the implementation of a new architecture for speeding-up the optical-flow computation under varying illumination. This method is based on pixel change instead of full image processing and it shows an improved speed-up.

In order to achieve the correct adaptation to illumination changes the analysis of each consecutive image pair acquired while determination of changes in illumination intensity is necessary. This is performed by introducing a module capable of detecting changes in illumination intensity between consecutive images in a pair, this module employs a threshold so it can distinguish a real movement from an apparent movement. The threshold is calculated considering a set of pixels from the consecutive image pair.

The performance of the system has a direct dependence on the number of pixels changing for consecutive images. And this number of pixels depends on several factors, such as: the sequence characteristics, the image size, the environment of the place where images were acquired, the camera sensitivity, the capture image frequency, the algorithm for *CST* determining and the *CST* value set. All these factors affect the number of pixels changing and avoid the possibility of processing a constant amount of images.

The architecture has been developed and implemented on an FPGA platform. And has been evaluated using classical optical flow test sequences and full-custom sequences. The average error is similar to other full-processing implementations, but the system proposed here is faster. The number of fps that can be processed depends on the image changes. In this way, it is not possible to predict a fixed number of images processed. The results obtained with the optical flow image test bench have determined that the system is able to process 90 frames of 256×256 grey level image data per second, under varying illumination.

ACKNOWLEDGMENTS
This work has been supported by the projects SIP: 20091789, SIP: 20100537 and COFAA-IPN.

REFERENCES

[1] Kim Y., Martinez A. M., and Kak A. C.: Robust motion estimation under varying illumination, Image and Vision Computing, vol. 23, pp. 365-375, 2005.

[2] Brinkworth RSA, O'Carroll DC (2009).: Robust Models for Optic Flow Coding in Natural Scenes Inspired by Insect Biology. PLoS Comput Biol 5(11):e1000555. doi:10.1371/journal.pcbi.1000555.

[3] Wedel, A. and Cremers, D. and Pock, T. and Bischof, H.: Structure- and motion-adaptive regularization for high accuracy optic flow. ICCV09. 2009, pages: 1663-1668.

[4] Shen, X.H. and Wu, Y.: Sparsity model for robust optical flow estimation at motion discontinuities. CVPR10. 2010, pages: 2456-2463.

[5] Liu H., Hong T., Herman M., Camus T. and Chellappa R.: Accuracy vs. Efficiency Trade-offs in Optical Flow Algorithms, Computer Vision and Image Understanding, Vol. 72, No. 3, pp. 271-286, 1998.

[6] Horn B. K. P. and Schunck B. G.: Determining Optical Flow, Artificial Intelligence, vol. 17, pp. 185–203, 1981.

[7] Barron J. L., Fleet D. J., and Beauchemin S. S.: Performance of optical flow techniques, International Journal of Computer Vision, vol. 12 no. 1, pp. 43-77, 1994.

[8] Teng C. H., Lai S. H., and Chen Y. S.: Accurate optical flow computation under non-uniform brightness varations, Computer Vision and Image Understanding, vol. 97, 315-346, 2005.

[9] Brox T., Bruhn A., Papenberg N., and Weickert J.: High Accuracy Optical Flow Estimation Based on a Theory for Warping, Proc. 8th European Conference on Computer Vision, Springer LNCS 3024, vol. 4, pp. 25–36, May. 2004.

[10] Sosa J. C.: Sistema de visión basado en procesado guiado por cambios y lógica reconforgurable para el análisis de movimiento a alta velocidad". Ph.D. Thesis. Departamento de Informática, ETSE, Valencia University, Spain. November 2007.

[11] Díaz J., Ros E., Rodriguez G. and Del Pino B.: Real-time Architecture for Robust Motion Estimation under Varying Illumination Conditions, Jornal of Universal Computer Science, vol. 13, No. 3, pp. 363-376, 2007.

[12] Zhaoyi Wei, Dah-Jye Lee and Brent E. Nelson.: FPGA-based Real-time OpticalFlow Algorithm Design and Implementation, Journal of Multimedia, Vol.2, No. 5, p.p. 38-45, 2007.

[13] Stocker A. and Douglas R.: Analog Integrated 2-D Optical Flow Sensor, Analog Integrated Circuits and Signal Processing, Springer Netherlands. Vol. 46, p.p. 121–138, 2006.

[14] Dopico A. G., Correia M. V., Santos J. A., and Nunes L. M.: Distributed Computation of Optical Flow, International Conference on Computational Science, pp. 380-387, 2004.

[15] Correia M. V., and Campilho A.: A pipelined Real-Time Optical Flow Algorithm, Analog Integrated Circuits and Siganal Processing, Springer, vol. 46, no. 2, p.p. 121-138, Junuary 2004.

[16] Arribas P. C. and Monasterio F.: FPGA implementation of the Horn&Schunck Optical Flow Algorithm for Motion detection in real time Images, Proceeding XIII Design of circuits and integrated systems conference, pp. 616-621, 1998.

[17] Martin J. L., Zuloaga A., Cuadrado C., Lazaro J., and Bidarte U.: Hardware implementation of optical flow constraint equation using FPGAs, Computer Vision and Image Understanding, vol. 98, pp. 462-490, 2005.

[18] Sosa J. C., Boluda J.A., Pardo F., Gómez-Fabela R.: Change-Driven data flow image processing architecture for optical flow computation, Journal of Real-Time Image Processing, SpringerLink. ISSN: 1861-8200. Vol. 2, Number 4. December 2007. pp: 259-270.

[19] Pardo F., Boluda J. A., Sosa J. C., Benavent X. and Domingo J.: Circle detection and tracking speed-up based on change-driven image processing, ICGST International Conference on Graphics, Vision and Image Processing. GVIP'05. pp. 131-136. Cairo, Egypt, December 2005.

[20] Roberto Rodríguez, Teresa E. Alarcón and Oriana Pacheco.: A New Strategy to Obtain Robust Markers for Blood Vessels Segmentation by using the Watersheds Method, Journal of Computers in Biology and Medicine (Elsevier), Vol. 35/8, pp. 665-686, 2005.