5

High-Speed Architecture Based on FPGA for a Stereo-Vision Algorithm

M.-A. Ibarra-Manzano[1] and D.-L. Almanza-Ojeda[2]
[1]*Digital Signal Processing Laboratory, Electronics Department; DICIS*
University of Guanajuato
[2]*Mechatronic Department, Campus Loma Bonita*
University of Papaloapan
[1]*Salamanca, Guanajuato, Mexico*
[2]*Loma Bonita, Oaxaca, Mexico*

1. Introduction

Stereo vision is used to reconstruct the 3D (depth) information of a scene from two images, called left and right. This information is acquired from two cameras separated by a previously established distance. Stereo vision is a very popular technique used for applications such as mobile robotics, autoguided vehicles and 3D model acquisition. However, the real-time performance of these applications cannot be achieved by conventional computers, because the processing is computationally expensive. For this reason, other solutions like reconfigurable architectures have been proposed to execute dense computational algorithms.

In the last decade, several works have proposed the development of high-performance architectures to solve the stereo-vision problem i.e. digital signal processing (DSP), field programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC). The ASIC devices are one of the most complicated and expensive solutions, however they afford the best condition for developing a final commercial system (Woodfill et al., 2006). On the other hand, FPGA have allowed the creation of hardware designs in standard, high-volume parts, thereby amortizing the cost of mask sets and significantly reducing time-to-market for hardware solutions. However, engineering cost and design time for FPGA-based solutions still remain significantly higher than software-based solutions. Designers must frequently iterate the design process in order to achieve system performance requirements and simultaneously minimize the required size of the FPGA. Each iteration of this process takes hours or days to be completed (Schmit et al., 2000). Even if designing with FPGAs is faster than designing ASICs, it has a finite resource capacity which demands clever strategies for adapting versatile real-time systems (Masrani & MacLean, 2006).

In this chapter, we present a high-speed reconfigurable architecture of the Census Transform algorithm (Zabih & Woodfill, 1994) for calculating the disparity map from a dense stereo-vision system. The reuse of operations and the integer/binary nature of these operations were carefully adapted on the FPGA for obtaining a final architecture that generates up to 325 dense disparity maps of 640×480 pixels, even though most of the vision-based systems do not require high video-frame rates. In this context, we propose a stereo-vision system that can be adapted to the real-time application requirements. An analysis of the four essential architectural parameters (such as the size of the window of the arithmetic mean and median filters, the maximal disparity and the window size for the Census Transform), is carried out to obtain the best trade off between consumed resources and the disparity map accuracy. We vary these parameters and show a graphical representation of the consumed resources versus the desired performance for different extended architectures. From these curves, we can easily select the most appropriate architecture for our application. Furthermore, we develop a practical application of the obtained disparity map to tackle the problem of 3D environment reconstruction using the back-projection technique. Experimental performance results are compared to those of related architectures.

2. Overview of passive stereo vision

In computer vision, stereo vision intends to recover depth information from two images of the same scene. A pixel in one image corresponds to a pixel in the other, if both pixels are projections of the same physical scene element. Also, if the two images are spatially separated but simultaneous, then computing correspondence determines stereo depth (Zabih & Woodfill, 1994). There are two main approaches to process the stereo correlation: feature-based and area-based. In this work, we are more interested in area-based approaches, because they propose a dense solution for calculating high-density disparity maps. Furthermore, these approaches have a regular algorithmic structure which is suitable for a convenient hardware architecture. The global dense stereo vision algorithm used in this work is based on the Census Transform. This algorithm was first introduced by Zabih and Woodfill (Zabih & Woodfill, 1994). Figure 1 shows the block diagram of the global algorithm.

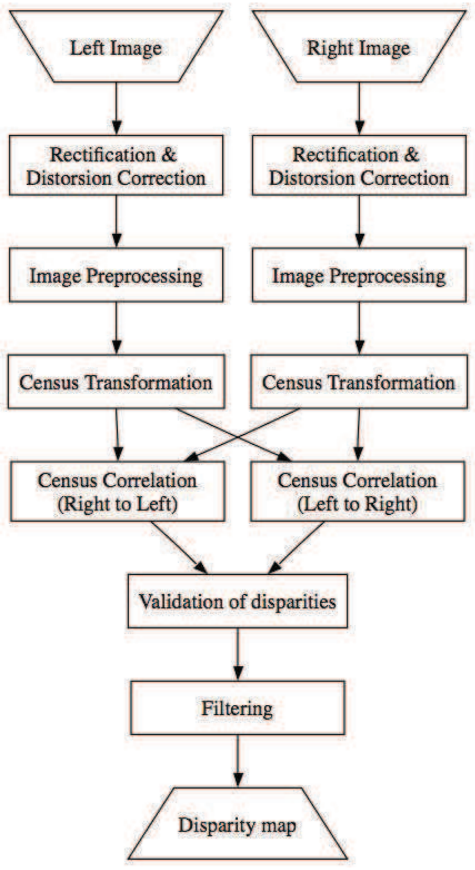

Left Image
Right Image
Rectification & Distorsion Correction
Rectification & Distorsion Correction
Image Preprocessing
Image Preprocessing
Census Transformation
Census Transformation
Census Correlation (Right to Left)
Census Correlation (Left to Right)
Validation of disparities
Filtering
Disparity map

Fig. 1. Stereo vision algorithm

First of all, the algorithm processes in parallel and independently each of the images (right and left). The process begins with the rectification and correction of the distortion for each image. This process allows us to reduce the size of the search of points for the calculation of the disparity to a single dimension. In order to reduce the complexity and size of the required architecture, this algorithm uses the epipolar restriction. In this restriction, the main axes of the cameras should be aligned in parallel, so that the epipolar lines between the two cameras correspond to the displacement of the position between the two pixels (one per camera). Under this condition, an object location in the scene is reduced to a horizontal translation. If any pair of pixels is visible in both cameras and assuming they are the projection of a single point in the scene, then both pixels must be aligned on the same epipolar line (Ibarra-Manzano, Almanza-Ojeda, Devy, Boizard & Fourniols, 2009).

2.1 Image preprocessing

The Census Transform requires that left and right input images be pre-processed. During image pre-processing, we use an arithmetic mean filter that requires a rectangular window of size $m \times n$ pixels. S_{uv} represents a set of image coordinates inside of the rectangular window centered on the point (u, v). The arithmetic mean filter calculates the mean value in the noisy image $I(u, v)$ at each rectangular window defined by S_{uv}. The corrected image value $\hat{I}$ takes this arithmetic mean value at each point (u, v) of subset S_{uv} (see Equation 1).

$$\hat{I}(u,v) = \frac{1}{m \times n} \sum_{(i,j) \in S_{uv}} I(i,j) \tag{1}$$

This filter could be implemented without using the scale factor $1/(m \times n)$ because the size of the window is constant during the filtering process. The arithmetic mean filter smooths local variations in the image, at the same time, the noise produced by camera motions is notably reduced.

2.2 Census Transform

Once input images have been filtered, they are used to calculate the Census Transform. This transform is a non-parametric measure used during the matching process for measuring similarities and obtaining the correspondence between the points into the left and right images. A neighborhood of pixels is used for establishing the relationships among them (see Equation 2),

$$I_C(u,v) = \bigotimes_{(i,j) \in D_{uv}} \xi\left(\hat{I}(u,v), \hat{I}(i,j)\right) \tag{2}$$

where D_{uv} represents the set of coordinates into the square window of size $n \times n$ pixels (being n an odd number) and centered at the point (u, v). The function ξ is the comparison of the intensity level among the center pixel (u, v) with all the pixels in D_{uv}. This function returns '1' if the intensity of the pixel (i, j) is lower than the intensity of the centering pixel (u, v), otherwise the function returns '0'. The operator $\otimes$ represents the concatenation function among each bit calculated by the function ξ. I_C represents the Census Transform of the point (u, v) which is a bit chain.

2.3 Census correlation

The two pixels (one for each image) obtained from the Census Transform are compared using the Hamming distance. This comparison which is called the correlation process allows us to obtain a disparity measure. The similarity evaluation is based on the binary comparison between two bit chains given by the Census Transform. The disparity measure from left to right D_{H1} in the point (u, v) is calculated by the equation 3, where I_{Cl} and I_{Cr} represent the left and right images of the Census Transform, respectively. This disparity measure comes from the similarity maximization function in the same epipolar line v for the two images. In this same equation, D represents the maximal displacement value on the epipolar line of the right image. The function $\bar{\otimes}$ represents the binary operator $XNOR$.

$$D_{H1}(u,v) = \max_{d \in [0,D]} \left(\frac{1}{N} \sum_{i=1}^{N} I_{Cl}(u,v)_i \bar{\otimes} I_{Cr}(u-d,v)_i \right) \tag{3}$$

The correlation process is carried out two times, (left to right then right to left) with the aim of reducing the disparity error. The equation 4 is for that case in which the right to left disparity measure is calculated. This measure was added for complementing the process. Contrary to the previous disparity measure shown in equation 3, the equation 4 uses the following pixels with respect to the current pixel in the search process.

$$D_{H2}(u,v) = \max_{d \in [0,D]} \left(\frac{1}{N} \sum_{i=1}^{N} I_{Cl}(u+d,v)_i \bar{\otimes} I_{Cr}(u,v)_i \right) \tag{4}$$

2.4 Disparity validation

Once both disparity measures have been obtained, the validating task is straightforward. The disparity measure validation (right to left and left to right) consists of comparing both disparity values and obtaining the absolute difference between them. In the case that this difference is lower than a predefined threshold δ, then the disparity value is accepted. Otherwise, the disparity value is labeled as undefined. The equation 5 represents the validation of the disparity measures, D_H being the validation result.

$$D_H = \begin{cases} D_{H1} & |D_{H1} - D_{H2}| < \delta \\ ind & |D_{H1} - D_{H2}| \geq \delta \end{cases} \tag{5}$$

2.5 Disparity filtering

A novel filtering process is needed in order to improve the quality of the final disparity image. M_{uv} is the set of coordinates in a $m \times n$ rectangular window centered on the point (u, v). First, the set of disparity values $D_H(i, j)$ in the region defined by M_{uv} are ordered. After that, the median filtering process selects the centered value at the ordered list. This value is set into the region defined by an $M \times N$ rectangular window $M_{u,v}$ and the same process is carried out for all the image pixels (i, j) in order to obtain the filtered image $\tilde{D}_H$. Hence, this filtered image calculated by the median filter, when expressed in terms of the central pixel (u, v), would be written as in equation 6.

$$\tilde{D}_H(u,v) = \text{median}\left(D_H(i,j), (i,j) \in M_{uv}\right) \tag{6}$$

Whereas, for the image preprocessing (described above), an arithmetic mean filter is used, here for the pre-filtering process a median spatial filter is used, because the median filter allows the selection of one value among all the disparity values for representing the disparity in the search window. This means that a new value does not need to be obtained, as in the arithmetic filter.

3. Hardware implementation

We have implemented an architecture for FPGA that implements several image processing tasks with high performance. During the architecture design, we try to minimize the consumed resources in the FPGA for maximizing the system performance. In previous subsections, we have explained the 4 essential tasks of our architecture: the image processing, the Census Transform, the disparity validation and the filtering of the disparity image. In this section, we describe the hardware implementation, that is, how these architectures are implemented into the FPGA.

3.1 The arithmetic mean filter module

Usually the image acquired from the camera is not so noisy, thus a fast smoothing function could be used for obtaining a good quality image. For this reason, we use the arithmetic mean filter, which is faster and easy to implement. We are often interested in the minimization of required resources, so that, after several tests, we choose a window size of 3×3 pixels that is also enough for achieving good results. Indeed, this size allows us to save resources in the final architecture.

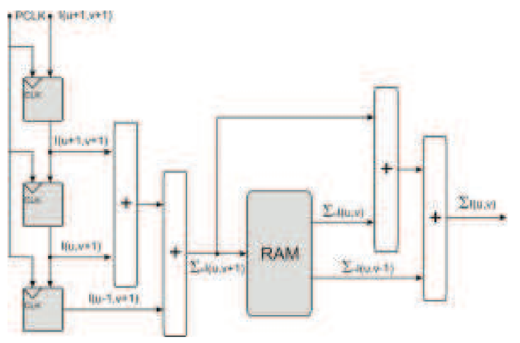


Fig. 2. Module architecture to calculate the arithmetic mean filter.

The arithmetic mean calculation is carried out in two stages: in horizontal and vertical ways. The block diagram of this architecture, in accordance with the process described in subsection 2.1, is shown in figure 2. The three input registers (left side of the diagram) are used for the horizontal addition. These registers are connected to two 8-bit parallel adders, although the result is coded in 10 bits. The result of this operation is stored in the memory that is twice the length image size. For obtaining the sum of all the elements in the window, a vertical addition is carried out. This addition uses the current horizontal addition result plus the two previous horizontal additions stored in the memory. This is shown on the right side of the diagram. Finally the arithmetic mean of the nine pixels are codified in a 12 bit-chain. In this stage, the delay only depends on the operation that involves the last line plus one value.

3.2 The Census Transform module

The arithmetic mean of the left and right images are used as inputs in the Census Transform stage. This transformation codifies all the intensity values inside the search window with respect to the central intensity value.
The block diagram of the Census Transform architecture is shown in the figure 3. The performance in this module depends on the size of the search window. The size of this window directly increases the resources and the time of processing. So the best trade off between the consumed resources and the optimal size of the window has to be selected. After several tests, the best processing time and hardware saving resources is reached for a 7×7

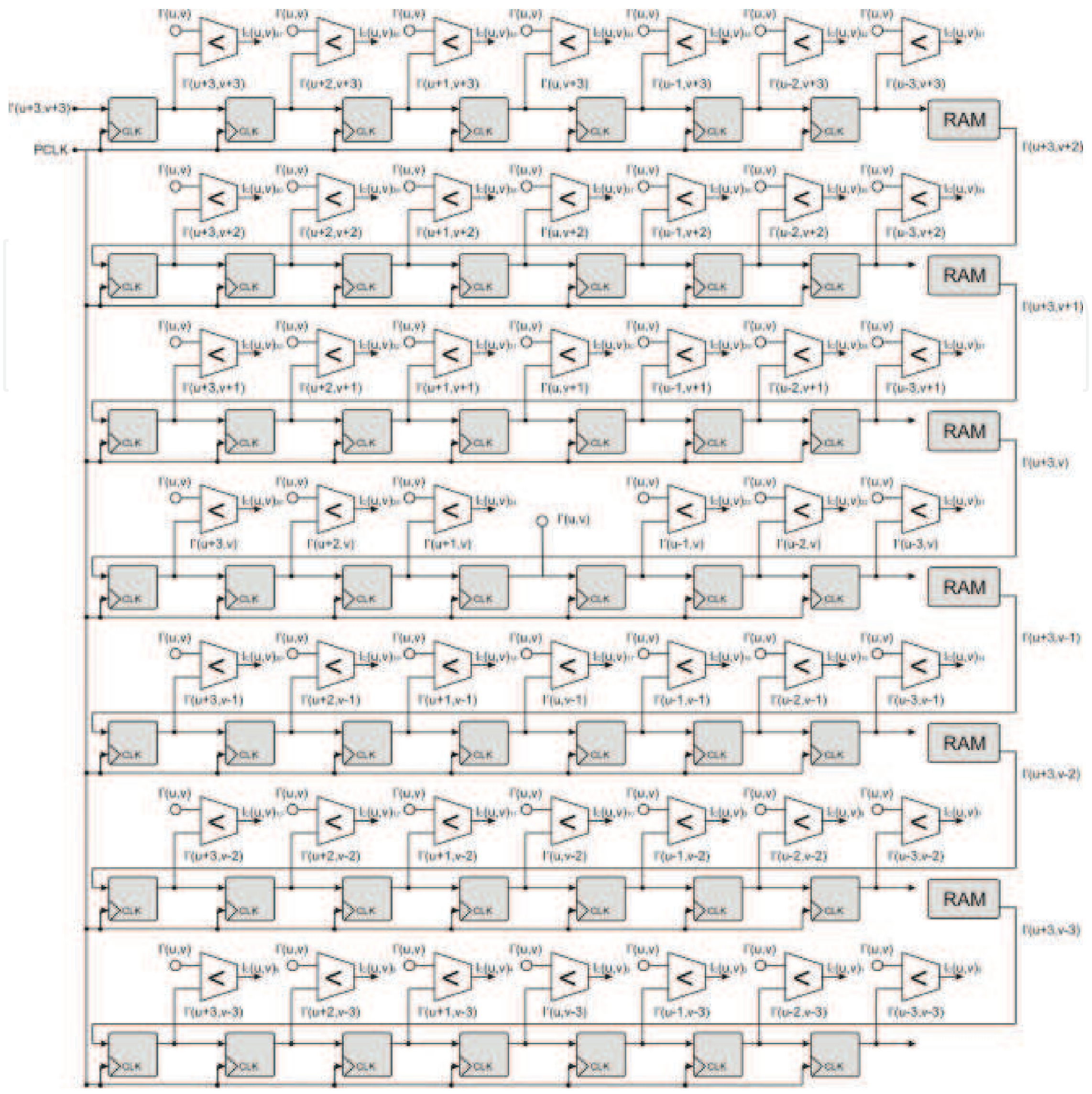


Fig. 3. Module architecture to calculate the Census Transform.

pixels window. This window needs 49 registers. On the other hand, 6 memory blocks are used in the processing module. The size of these memory blocks is obtained as follows: size of the image (usually 640) minus the size of the search window (7 pixels in our case) and the result is multiplied by 12. The constant 12 in the last multiplication is used because we look for the same size in the input of the Census Transform rather than in the output of the arithmetic mean module. Once we have selected the size of the search window, then we continue with the description of the Census Transform. The central pixel in the search window is compared with their 48 local neighbors. This last operation implies the connection of all the corresponding registers with parallel comparators as is shown in figure 3. The result is codified in 48 bits, where each bit corresponds to the comparator outputs. This stage has a delay equal to half of the search window by the length of the image.

3.3 The census correlation module

The correlation process consists of analyzing left and right images resulting from the Census Transform. Considering that both images contain a common object, the correlation process

has the aim of finding the displacement between two projected pixels belonging to that common object in the images. Hence, as images are acquired from two different points of view (associated with the left and right camera positions) there will exist a noticeable difference between point positions belonging to the same object, such difference is referred to as the disparity. Usually, the correlation intends to maximize the similarity between two images in order to find the disparity. We also use this common method which consists of two main stages: the calculation of the similarity measure and the search for the maximal value. Figure 4 shows the block diagram of the corresponding architecture. We are interested in reducing the delay in the correlation process, therefore it is more convenient to compare one point of the left Census Transform image with the maximal number of points in the right one. Furthermore, the correlation process is executed twice in order to minimize the error during the correlation computation. We will consider 64 pixels as the maximal disparity value for each left and right Census Transform image. The registers represented by the gray blocks on the left of figure 4 store those pixels. The registers as well as the pixels of the left Census Transform image enter the binary operators *XNOR* to deliver a 48-bit chain at the output. The *XNOR* function is used to find the maximal and minimal similarity associated with the disparity values at the input. All such pixel values enter by pairs into the *XNOR* gates. If all the compared pixels are equal, then the *XNOR* output will be '1', which means maximal similarity. Otherwise, if pixels are different, then 0 will be the output of the *XNOR*, which is associated with a minimal similarity value.
Once the similarity has been calculated, we continue with the search of the highest disparity values between the 64 pixels compared in both correlation results, from left to right and right to left correlations, but independently. This task requires several selector units, each one with 4 inputs distributed as follows: 2 for the similarity values that will be compared and 2 for the indicators. The indicators are associated with the displacement between pixels in the left and right Census Transform. That is, if one pixel has the same position in both right and left Census Transform, then the indicator will be zero. Otherwise, the indicator will represent the number of pixels between pixel position in the left Census Transform with respect to the pixel in the right one. The block diagram of the architecture shown in figure 4 describes graphically the implementation of the maximization process. By examining this figure, we can highlight that the selector unit receives two similarity measures and two indicators as inputs. The elements inside of these units are a multiplexer and a comparator. The multiplexer receives the pixel with the highest similarity value, while the comparator receives two similarity values that come from the first stage. Hence the output of that comparison will be considered as the selector inputs of the multiplexer. Thus, the multiplexer output will be the similarity measure and its refereed index pixel. However, in order to obtain the pixel with the maximal similarity measure, six levels of comparison units are needed. These levels are organized in a pyramidal fashion. The lowest level corresponds to the first layer that carries out the selector unit task described above 32 times. As we ascend the pyramid levels, each level reduces by half the number of operators used with the previous level. The last level delivers the highest similarity value between the corresponding pixels in the left and right Census images. Whereas right to left image correlation stores left Census Transform pixels, which are compared with one pixel of the right Census image, for left to right image correlation the comparison is relative to one pixel of the left Census image. For this reason, a similar architecture is used for developing both correlation processes. All this stage (including both right to left and left to right processes) has a delay that depends on the number of layers in the selector units, which at the same time depends on the maximal disparity value that we are using. In our case, we establish maximal disparity value to 64, thus the number of layers is equal to 6.

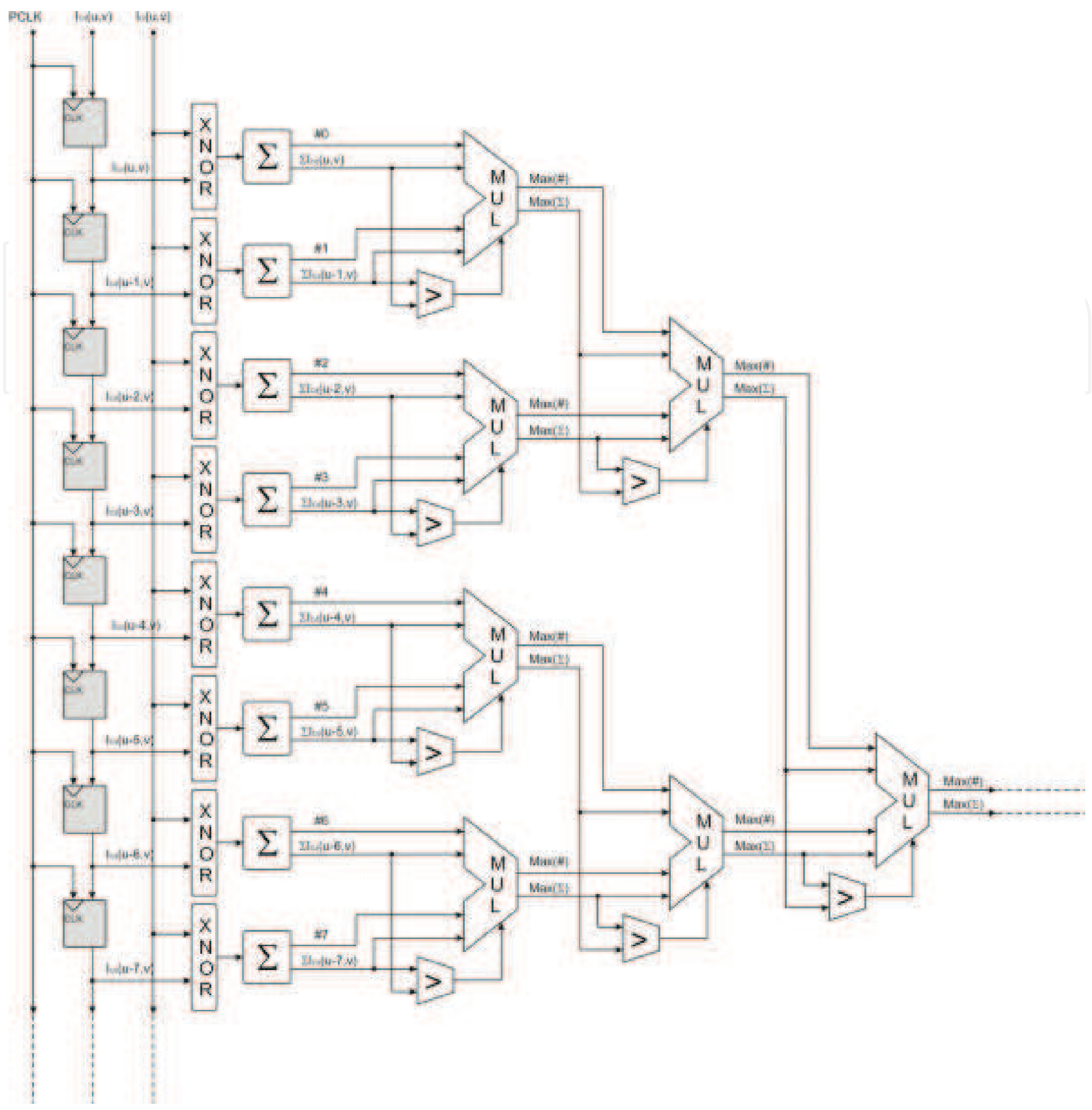


Fig. 4. Module architecture to calculate the Census correlation.

3.4 Disparity validation module

This module fuses the two disparity values obtained by the Census correlation processes (right to left and left to right). First, the difference between the two values is calculated, after that it is compared with a threshold δ. If that difference is lower than δ then the left to right disparity value is the output of the comparison, otherwise, the output will be labeled as undefined.

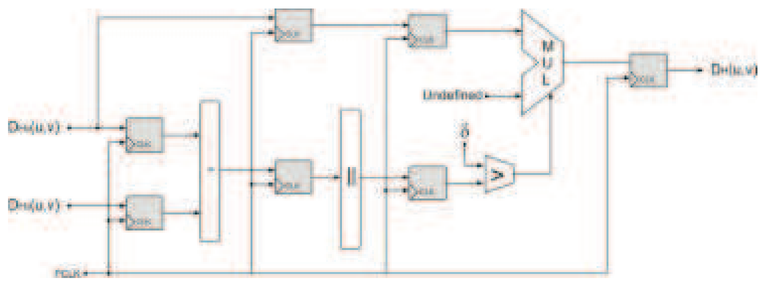


Fig. 5. Module architecture to validate the disparities.

The figure 5 shows the block diagram of the disparity validation architecture. The inputs of this module are the two disparity values obtained by the Census Correlation process. The absolute difference of values is used in the comparison with δ. The comparator delivers one bit that controls the multiplexer selector. If the result of the comparison is 1 (that is, δ is higher than the correlation differences), then the multiplexer will have the undefined label as result. This result is associated with the maximal disparity value plus one, which is referred to as the default value. If the comparison result is zero, then the output of the multiplexer will be the value of the left to right Census correlation.

3.5 The median filter module

Some errors have been detected during the disparity validation, due to the errors in the correlation process. Most of these errors appear because objects in the image have similar intensity values to their surrounding area. This situation produces very similar Census Transform values in pixels and consequently wrong disparity values in certain cases. We are interested in reducing these errors in the image by using a median filter. As we pointed out before, it is not recommended to use the same arithmetic mean filter as in the pre-processing stage because this filter will give us a new value (the average into the filtering window), which is not an element of the current disparity values. On the other hand the median filter works with the true values in the image, so the resulting value will be an element of the disparity window. The median filter uses a search window of size 3×3 pixels. This window is enough for notably reducing the error and improving the final disparity map as is shown in figure 6.

(a) Left stereo image (b) Disparity map (c) Filtered image of disparity map

Fig. 6. Median Filter. a) Left image, b) Resulting disparity map without filtering and c) with filtering.

Figure 7 shows the block diagram of the median filter architecture. This filter is based on the works of (Dhanasekaran & Bagan, 2009) and (Vega-Rodriguez et al., 2002). On the left side of the diagram is shown the nine registers and the two RAM block memories used to generate the sliding window that extracts the nine pixels of the processing window. This architecture works similar like to the processing window of the Census Transform. That is, the nine pixels in the window are processed by a pyramidal architecture but in this case with 9 levels. Each level contains several comparison units that find the higher value between two input values A and B. Each comparison unit contains a comparator and a multiplexer. If the input A in the comparator is higher than its input B, then the output will be 1, otherwise the output will be 0. The comparator output is used as a signal control of the multiplexer. When this signal is 1, then the multiplexer selects A as the higher value and B as the lower value, otherwise B is the higher value and A the lower one. Each comparator level in the median module orders the disparity values with respect to its neighbors in the processing window, completing in this way the descendent organization of all the values. However, here it is not necessary to order all the disparity values, because we are only looking for the middle value in the last level. Therefore, we only need the comparison unit at the last level, because previous levels give only the partial order of the elements. The connection structure between the comparison units at each level guaranty an optimal median value (Vega-Rodriguez et al., 2002).

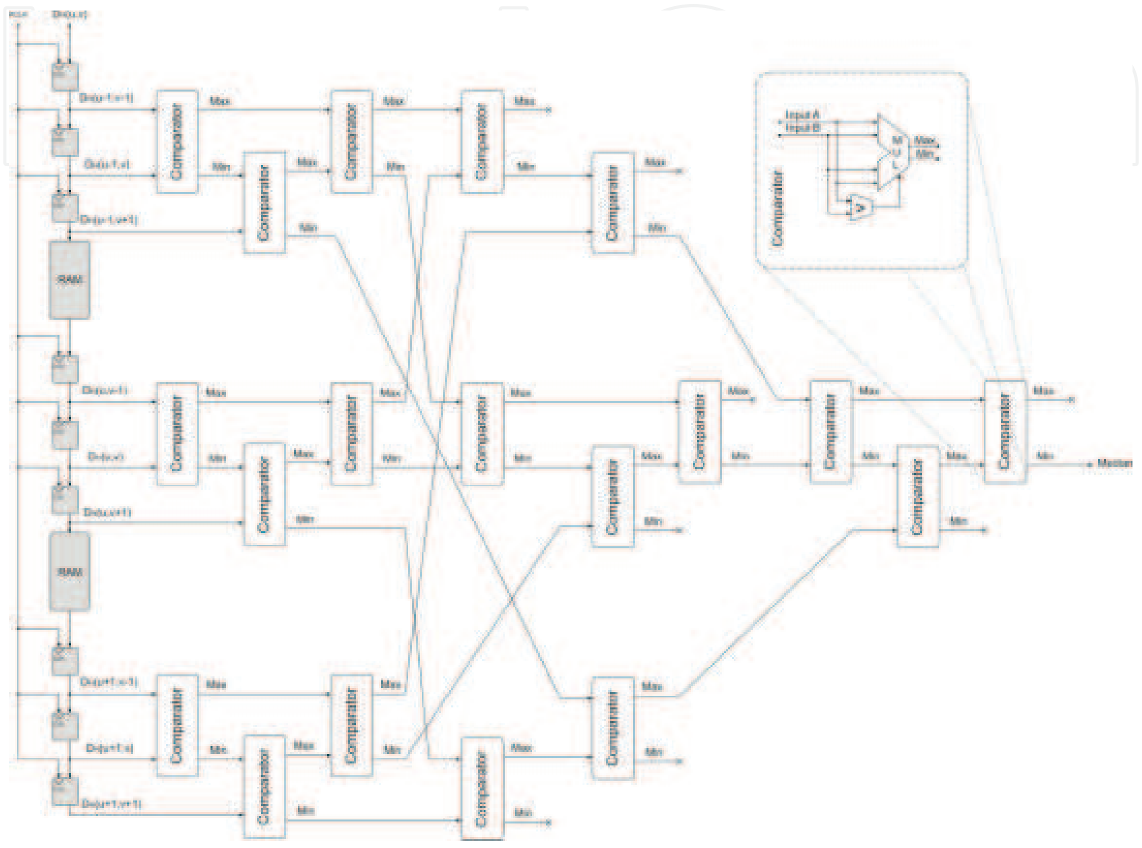


Fig. 7. Module architecture to calculate the median filter of disparities.

4. Resources and performance discussion

Our final architecture for executing the stereo vision algorithm based on the Census Transform was developed using the level design flow RTL (Ibarra-Manzano, 2011). The architecture was codified in VHDL language using Quartus II workspace and ModelSim. Finally, it was synthesized for an EP2C35F672C6 device contained in the Cyclone II family of Altera.
Some synthesis results associated with our architecture are: the implemented architecture implies 11, 683 combinatorial functions and 5, 162 dedicated logic registers, both represent 12, 183 logic elements in total. The required memory is 112, 025 bits. The quantity of logic elements represent only 37% of the total capacity in the device while the memory size represents 43%. The resources consumed by the architecture are directly associated with 5 essential parameters: the image size, window processing size used in both arithmetic mean and median filter, the window size in the search window of Census Transform and the maximal value in the disparity measure. In this architecture, we use an image size of 640×480 pixels, a window size of 3×3 pixels for both filters (arithmetic mean and median filters), a search window of 7×7 pixels for the Census Transform and a maximal disparity value of

64 pixels. With these parameters, the architecture is able to calculate 130 disparity images per second with a 50 Mhz signal clock until 325 disparity images per second with a 100 Mhz signal clock.

4.1 Architectural exploration through high-level synthesis

High level synthesis was used to implement the stereo vision architecture based on Census Transform. The algorithm was developed using GAUT (Coussy & Morawiec, 2008), which is a high level synthesis tool using C language. After that, the algorithm was synthesized using the (EP2C35F672C6) Cyclone II of Altera. Each state of the architecture (filtering, Census Transform and Correlation) was developed taking into account consumed resources and high performance (high speed of processing). The best trade off was found for implementing an optimal architecture system.
Tables 1 to 3 lay out three different architectures, labeled as Design 1, 2, and 3, with their most representative performance. In the following, we will describe how the different implementation details are related in our architecture. There exists a clear relation between performance, cadence and pipeline implementation. That is, if we reduce the performance, then the cadence increases, therefore the number of operations and stages in the pipeline is low. With the rest of feature design, it is more difficult to see how they are related. For example, the number of logic elements depends directly on the used combinational functions and the number of dedicated logic registers. The combinational functions are strongly associated with the quantity of operations and weakly with the state numbers in the state machine. As with any state machine, the cadence time controls the performance speed. Contrary to the combinational functions, the dedicated logic registers strongly depends on the number of states in the state machine and weakly on the number of operations. Finally, the delay is obtained based on the number of operations, the number of stages in the pipeline and specially in the cadence time established by the architecture design. The results shown in the tables 1 to 3 were carried out for an image size of 640 $\times$ 480 pixels with a processing window of 3 $\times$ 3 pixels for the arithmetic mean filter, a window size of 7 $\times$ 7 pixels for the Census Transform and a maximal disparity measure of 64 pixels, with a signal clock of 100 Mhz.

Characteristics	Design 1	Design 2	Design 3
Cadency (ns)	20	30	40
Performance (fps)	160	100	80
Logic elements	118	120	73
Comb, functions	86	72	73
Ded. log. registers	115	116	69
# Stages in pipeline	3	2	2
# Operators	2	2	1
Latency (μs)	25.69	38.52	51.35

Table 1. Comparative table for the arithmetic mean filter.

Taking into account the most common real time constraints, it is possible to choose the design 3 for the implementation of the arithmetic mean filter, because this represents the best compromise between performance and consumed resources. For the same reason, the design 2 could be chosen for developing the Census Transform and the design 3 for the Census correlation. The results of the hardware Synthesis in FPGA are summarized as follows: the global architecture needs 6,977 logic elements and 112,025 memory bits. The quantity of logic elements represents 21% of the total resources logic of the Cyclone II device, furthermore

Characteristics	Design 1	Design 2	Design 3
Cadency (ns)	40	80	200
Performance (fps)	80	40	15
Logic elements	2,623	1,532	1,540
Comb. functions	2,321	837	864
Ded. log. registers	2,343	1,279	1,380
# Stages in pipeline	48	24	10
# Operators	155	79	34
Latency (μs)	154.36	308.00	769.50

Table 2. Comparative table for the Census Transform.

Characteristics	Design 1	Design 2	Design 3
Cadency (ns)	20	40	80
Performance (fps)	160	80	40
Logic elements	1,693	2,079	2,644
Comb. functions	1,661	1,972	2,553
Ded. log. registers	1,369	1,451	1,866
# Stages in pipeline	27	12	8
# Operators	140	76	46
Latency (ns)	290	160	100

Table 3. Comparative table for the Census correlation.

the memory size represents 23%. This architecture calculates 40 dense disparity images per second with a clock of 100 Mhz. This performance is lower than the proposed architecture, although it proposes a well-optimized design, since it uses less resources than in the previous case. In spite of the low performance, this is high enough in the majority of real-time vision applications.

4.2 Comparative analysis of the architectures

First, we will analyze the system performance for four different solutions to the dense disparity image. Two of the above mentioned solutions are hardware implementations. The third one is a solution for a Digital Signal Processing (DSP) model ADSP-21161N, with a signal clock of 100 MHz from Analog Devices Company. The last one is a software solution for a PC DELL Optiplex 755 with a 2.00 Ghz Intel Core 2 Duo processor and 2 Gb in RAM. The performance comparison between these solutions is shown in table 4. The first column indicates the different image sizes used during the experimental test. The second column shows the sizes of the search window used in the Census Transform. The third column shows the processing time (performance).

In the FPGA implementation, the parallel processing allows short calculation time. The developed architecture uses the RTL level design which reaches the lower processing time, but it takes more time for the implementation. On the other hand, using high level synthesis for the architecture design allows the development of a less complex design, but it requires longer processing time. However, the advantage of high level synthesis is the short implementation time. Unlike FPGA implementations, the DSP solutions are easier and faster to implement, nevertheless the processing remains sequential, and so the computation time is considerably high. Finally, the PC solution, that affords the easiest implementation of all above discussed, requires very high processing times compared to the hardware solution, since it has an inappropriate architecture for real time applications.

Image size (pixels)	Census window size (pixels)	Time of processing FPGA	DSP	PC
192×144	3×3	$0.69ms$	$0.26s$	$33.29s$
192×144	5×5	$0.69ms$	$0.69s$	$34.87s$
192×144	7×7	$0.69ms$	$1.80s$	$36.31s$
384×288	3×3	$2.77ms$	$1.00s$	$145.91s$
384×288	5×5	$2.77ms$	$2.75s$	$151.39s$
384×288	7×7	$2.77ms$	$7.20s$	$158.20s$
640×480	3×3	$7.68ms$	$2.80s$	$403.47s$
640×480	5×5	$7.68ms$	$7.70s$	$423.63s$
640×480	7×7	$7.68ms$	$20.00s$	$439.06s$

Table 4. Performance comparison of different implementation.

We present a comparative analysis between our two architectures and four different FPGA implementations found in the literature. The first column of table 5 lays out the most common characteristics of the architectures. The second and third columns show the limitations, performance and consumed resources by our architectures using the RTL level design and the High level synthesis HLS (Ibarra-Manzano, Devy, Boizard, Lacroix & Fourniols, 2009), labeled as Design 1 and Design 2, respectively. The remaining columns show the corresponding values for the four architectures, labeled as Design 3 to 6. These architectures were designed by different authors. See their corresponding articles for more technical details (Naoulou et al., 2006), (Murphy et al., 2007), (Arias-Estrada & Xicotencatl, 2001) y (Miyajima & Maruyama, 2003) for Design 3 to 6. Besides all of these are FPGA implementations, they calculate dense disparity images from two stereo images. Our architecture could be directly compared with Design 3 and 4, since they use the Census transform algorithm for calculating the disparity map. We propose two essential improvements with respect to Design 3: the delay and the size of memory. These improvements directly affect the number of logic elements (area) that in our case increase. With respect to Design 2, we propose three important improvements: the delay, the area and the memory size. Again these improvements impact the performance, that is the processed image per second is lower. Although Design 4 has a good performance with respect to other designs, this is lower than our architecture performance. In addition, it uses a four-times-smaller image, it has a lower value of disparity measure and it consumes a bigger quantity of resources (area and memory). Our architecture cannot be directly compared with Designs 5 and 6, since they use the Sum Absolute of Differences (SAD) as a correlation measure. However, an interesting comparison point is the architecture performance required for calculating the disparity map, at the moment that an architecture uses only logic elements (Design 5) or when several accesses to external memories are used (Design 6). The big quantity of logic elements consumed by the architecture in Design 5 limits the size of the input images and the maximal disparity value. As a consequence, this architecture has a lower performance with respect to our architecture (Design 1). The Design 6 requires a large quantity of external memory that directly affects its performance with respect to our Design 1.

5. Implementation results

We are interested in obtaining the disparity maps relative to a image sequence acquired from a camera mounted in a moving vehicle or robot. It is important to point out the additional

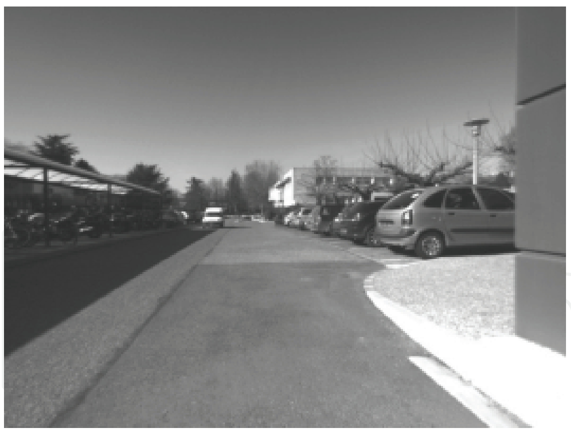
	Design 1	Design 2	Design 3	Design 4	Design 5	Design 6
Measure	Census	Census	Census	Census	SAD	SAD
Image size	640×480	640×480	640×480	320×240	320×240	640×480
Window size	7×7	7×7	7×7	13×13	7×7	7×7
Disparity max	64	64	64	20	16	80
Performance	325	40	130	40	71	18.9
Latency (μs)	115	206	274	–	–	–
Area	$12,188$	$6,977$	$11,100$	$26,265$	$4,210$	$7,096$
Memory size	114 Kb	109 Kb	174 Kb	375 Kb	–	–

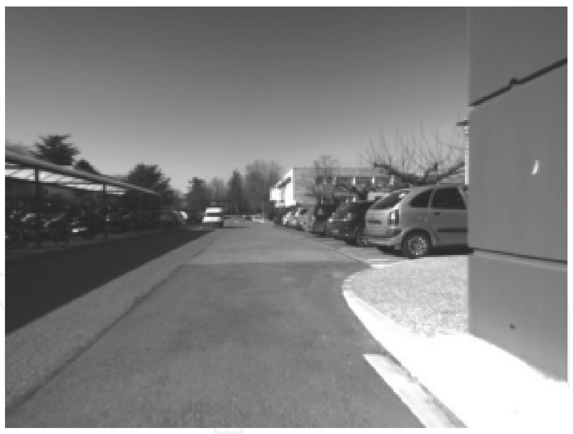
Table 5. Comparative table from different architectures.

constraint imposed by a vehicle in which the velocity is varying or very high. In this context, our architecture was tested for different navigational scenes using a stereo vision bank first mounted in a mobile robot and then in a vehicle. In this section, we present three operational environments. Figure 8 (a) and (b) respectively show the left and right images from the stereo vision bank. Dense disparity image depicts the disparity value in gray color levels in figure 8 (c). By examining this last image, we can determine that if the object is close to the stereo vision bank that means a big disparity value, so it corresponds to a light gray level. Otherwise, if the object is far from the stereo vision bank, the disparity value is low, which corresponds to a dark gray level. In this way, we observe that the gray color which represents the road in the resulting images gradually changes from light to dark gray level. We point out the right side of the image, where we can see the different tones of gray level corresponding to each vehicle in the parking. Since these vehicles are located at different depths from the stereo vision bank, the disparity map detects and assigns a corresponding gray color value.
The second test performed with the algorithm is shown in figure 9. In this case a white vehicle moves straightforward in our robot direction. This vehicle is detected in the disparity image and depicted with different gray color levels. Different depth points of the vehicle can be detected, since it is closer to our stereo vision bank than the vehicles parked at the right side of the scene. On the other hand, it is important to point out that sometimes the disparity validation fails because the similarity between left and right images is close. This problem is more significant when there are shadows close to the visual system (as in this experiment) producing several detection errors in the shadow zones.
In the last test (see figure 10), the stereo vision bank is mounted on a vehicle that is driven on a highway. This experimental test results in a difficult situation because the vehicle is driven at high-speed during the test. The left and right images (figure 10 (a) and (b) respectively) show a car that overtakes our vehicle. The figure 10 (c) shows the dense disparity map. We highlight all the different depths detected in the vehicle that overtakes our vehicle and how the gray color value in the highway becomes gradually darker until the black color which represents an infinity depth.

6. Back-projection for 3D environment reconstruction

Many applications use the obtained disparity image for the obstacles detection task because the association of disparity values with a certain depth in the real world is straightforward. However, maybe the most common application of the disparity image is the 3D reconstruction of the environment. The reconstruction method employs a technique called back-projection, which uses the intrinsic parameters of the camera and the disparity image for positioning one point in the real world.


(a) Left stereo image


(b) Right stereo image

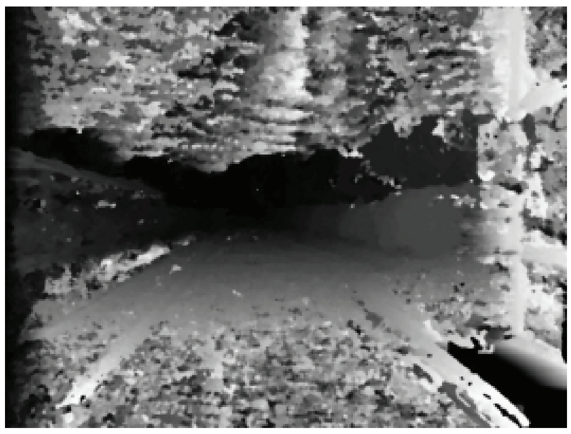
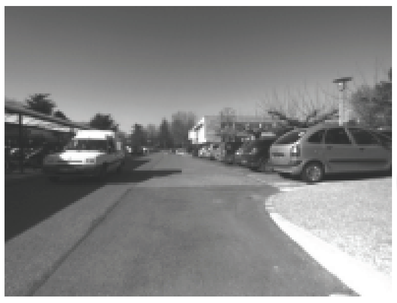
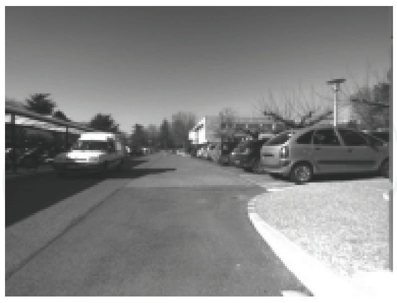

(c) Result from FPGA implementation

Fig. 8. Stereo images acquired from a mobil robot during outdoor navigation: a) left image b) right image and c) the disparity map.


(a) Left stereo image


(b) Right stereo image

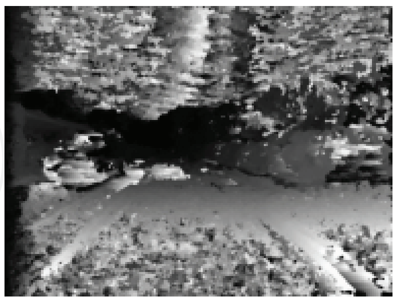
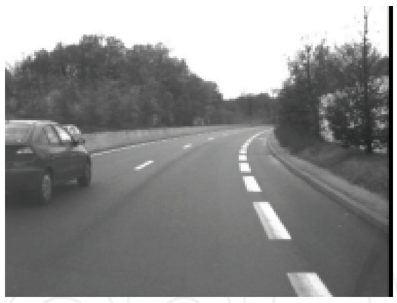

(c) Result from FPGA implementation

Fig. 9. Stereo images acquired from a mobile robot during outdoor navigation: a) left image b) right image and c) the disparity map.

Figures 11 (a) and (b) show the left image and the obtained disparity map respectively. Figure 11 (c) shows the reconstructed environment using the back-projection technique. Each point in the reconstructed scene was located with respect to a reference frame set in the stereo bank employing intrinsic/extrinsic parameters of the cameras and geometrical

(a) Left stereo image


(b) Right stereo image

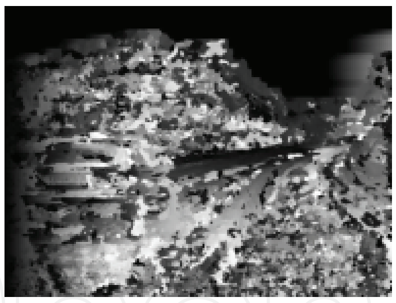
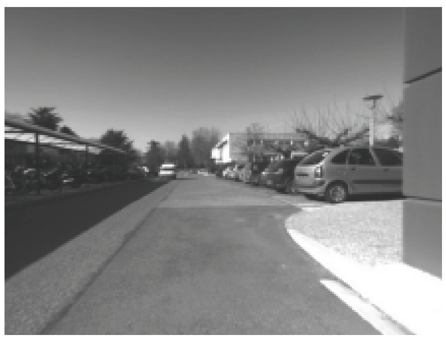
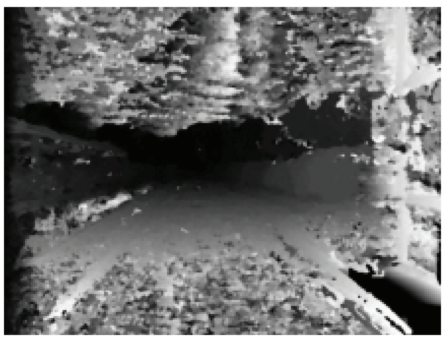
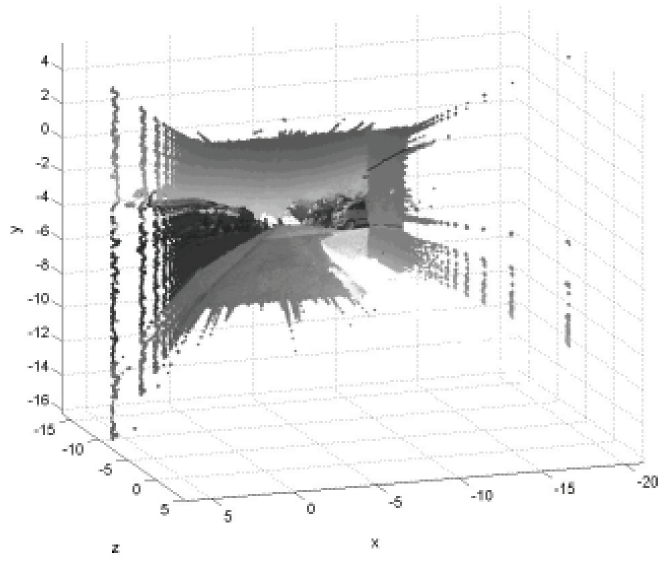

(c) Result from FPGA implementation

Fig. 10. Stereo images acquired from a vehicle in the highway: a) left image b) right image and c) the disparity map.


(a) Left stereo image


(b) Dense disparity image



(c) 3D reconstruction

Fig. 11. 3D reconstruction from outdoor environment using dense disparity map obtained by our architecture.

assumptions. By examining figure 11 (c), we can see that most of the undefined disparity points were removed, thus the reconstruction is based on the well-defined depth-points. Finally, it is important to point out that the reconstruction of this environment results in a difficult task, since the robot with the stereo vision bank moves with a considerable velocity of 6 meters per second and in an outdoor environment. Therefore, the ideal conditions of controlled illumination and controlled vibrations do not hold, and this will be reflected in some images, making it more difficult to obtain the disparity map and, consequently, the scene reconstruction.

7. Conclusions and perspectives

The logic-programmable technology is known as an intermediary solution between the programmable processors and the dedicated VLSI circuits due mainly to its price, performance and power consumption. Furthermore, it is difficult to establish frontiers for delimiting logic-programmable technology applications because of their continual evolution. All of this makes this technology an attractive solution.

Nowadays, the FPGAs allow the development of hardware systems that overcome most of the limitations of real-time applications. However, the price and design time of the FPGA solutions make this technology a complicated tool in comparison with the software solutions. The designers must repeat the flow diagram of the design several times in order to overcome the performance limitations of the application, under the constraints imposed by always reducing the resources consumed by the circuit. At each iteration, the flow design could take several hours and in some cases days before converging into the optimal solution.

The high level synthesis allows us to reduce the design time of the algorithm with confidence that the resulting code will be equally efficient. This fact requires that the quality in the design be independent of the designer abilities. With the aim of efficiency, the high level synthesis must use design methods that take into account the specialization domain of the application.

We take advantage of rapidly-processing natural stereo images to use our architecture in real time applications. Resulting disparity images demonstrate the correct detection of different depth planes in stereo image pairs. However, resulting images present several fail detections that make images corrupt and noisy. In order to improve disparity image quality, we could include an additional measure of correlation to our current Hamming distance (such as Tanimoto distance) or the latter as the only correlation measure used. Both (Hamming and Tanimoto) resulted in disparity measures that could be linked in a unique disparity measure that will be more discriminative than our current one.

The improvements in the stereo vision architecture include an algorithm for auto-calibration, in order to reduce the error during disparity calculation. Also, with the auto-calibration process we eliminate the previous calibration problem. In this way, the stereo vision system will be more suitable with respect to the current version. In the case of the modules, we are working on their parametrization. This consists of developing auto-configurable modules in which we could directly vary the window sizes (both processing and search window), and this allows us to develop a reconfigurable system useful for different purposes.

8. Acknowledgments

This work was partially funded by the CONACyT with the project entitled "Diseño y optimización de una arquitectura para la clasificación de objetos en tiempo real por color y textura basada en FPGA"

9. References

Arias-Estrada, M. & Xicotencatl, J. M. (2001). Multiple stereo matching using an extended architecture, *in* G. Brebner & R. Woods (eds), *FPL '01: Proceeding of the 11th International Conference on Field-Programmable Logic and Applications*, Springer-Verlag, London, UK, pp. 203–212.

Coussy, P. & Morawiec, A. (2008). *High-Level Synthesis: from Algorithm to Digital Circuit*, 1 edn, Springer.

Dhanasekaran, D. & Bagan, K. B. (2009). High speed pipelined architecture for adaptive median filter, *European Journal of Scientific Research* 29(4): 454–460.

Ibarra-Manzano, M. (2011). *Vision multi-caméra pour la détection d'obstacles sur un robot de service: des algoritmes à un système intégré*, PhD thesis, Institut National des Sciences Appliquées de Toulouse, Toulouse, France.

Ibarra-Manzano, M., Almanza-Ojeda, D.-L., Devy, M., Boizard, J.-L. & Fourniols, J.-Y. (2009). Stereo vision algorithm implementation in fpga using census transform for effective resource optimization, *Digital System Design, Architectures, Methods and Tools, 2009. 12th Euromicro Conference on*, pp. 799 –805.

Ibarra-Manzano, M., Devy, M., Boizard, J.-L., Lacroix, P. & Fourniols, J.-Y. (2009). An efficient reconfigurable architecture to implement dense stereo vision algorithm using high-level synthesis, *2009 International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, pp. 444–447.

Masrani, D. & MacLean, W. (2006). A Real-Time large disparity range Stereo-System using FPGAs, *Computer Vision Systems, 2006 ICVS '06. IEEE International Conference on*, p. 13.

Miyajima, Y. & Maruyama, T. (2003). A real-time stereo vision system with fpga, *in* G. Brebner & R. Woods (eds), *FPL '03: Proceeding of the 13th International Conference on Field-Programmable Logic and Applications*, Springer-Verlag, London, UK, pp. 448–457.

Murphy, C., Lindquist, D., Rynning, A. M., Cecil, T., Leavitt, S. & Chang, M. L. (2007). Low-cost stereo vision on an fpga, *FCCM '07: Proceeding of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE Computer Society, Washington, DC, USA, pp. 333–334.

Naoulou, A., Boizard, J.-L., Fourniols, J. Y. & Devy, M. (2006). A 3d real-time vision sytem based on passive stereovision algorithms: Application to laparoscopic surgical manipulations, *Proceedings of the 2nd Information and Communication Technologies, 2006 (ICTTA)*, Vol. 1, IEEE, pp. 1068–1073.

Schmit, H. H., Cadambi, S., Moe, M. & Goldstein, S. C. (2000). Pipeline reconfigurable fpgas, *Journal of VLSI Signal Processing Systems* 24(2-3): 129–146.

Vega-Rodriguez, M. A., Sanchez-Perez, J. M. & Gomez-Pulido, J. A. (2002). An fpga-baed implementation for median filter meeting the real-time requirements of automated visual inspection systems, *Proceedings of th 10th Mediterranean Conference on Control and Automation*, Lisbon, Portugal, pp. 1–7.

Woodfill, J., Gordon, G., Jurasek, D., Brown, T. & Buck, R. (2006). The tyzx DeepSea g2 vision system, ATaskable, embedded stereo camera, *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW '06. Conference on*, p. 126.

Zabih, R. & Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence, *ECCV '94: Proceedings of the Third European Conference on Computer Vision*, Vol. II, Springer-Verlag New York, Inc., Secaucus, NJ, USA, pp. 151–158.

ADVANCES IN
STEREO VISION
Edited by José R.A. Torreão
INTECH