Real-Time Stereo Image Matching on FPGA

Carlos Resende
DEEC, FEUP
ee04022@fe.up.pt

João C. Ferreira
INESC Porto, FEUP
jcf@fe.up.pt

Abstract

Real-time stereo image matching is an important computer vision task, with applications in robotics, driver assistance, surveillance and other domains. The paper describes the architecture and implementation of an FPGA-based stereo image processor that can produce 25 dense depth maps per second from pairs of 8-bit grayscale images. The system uses a modification of a previously-reported variable-window-size method to determine the best match for each image pixel. The adaptation is empirically shown to have negligible impact on the quality of the resulting depth map. The degree of parallelism of the implementation can be adapted to the available resources: increased parallelism enables the processing of larger images at the same frame rate (40 ms per image). The architecture exploits the memory resources available in modern platform FPGAs. Two prototype implementations have been produced and validated. The smaller one can handle pairs of images of size 208×480 *(on a Virtex-4 LX60 at 100 MHz); the larger one works for images of size* 640×480 *(on a Virtex-5 LX330 at 100 MHz). These results improve on previously-reported ASIC and FPGA-based designs.*

1. Introduction

Acquisition of three-dimensional information from images has important applications in computer vision [1] (including robotics [2], driver assistance [3] and surveillance [4]). This information can be obtained from stereo images in the form of dense disparity maps, which require the reliable establishment of correspondences between the images [5]. The computational effort of this task typically precludes achieving real-time performance with general-purpose processors. This has led to the development of various dedicated hardware systems [6, 7, 8, 9, 10].

A general approach to the calculation of the correspondences between the two images of a stereoscopic pair is based on a horizontal scan of the second image to find a matching position for each pixel of the first image. The matching pixel is the one whose neighborhood differs the least from the neighborhood of the pixel in the first image. Various metrics have been proposed [11], but the one based on the sum of absolute differences (SAD) of the neighborhood pixels is often chosen for hardware implementations due to its simplicity.

In the correspondence between stereo images using windows, the size of the neighborhood (size of the correspondence window) has a large influence on the quality of the matching. If the window is too small, the quantity of neighborhood information used is too small, producing errors of correspondence in large areas where pixel intensity is constant. On the other hand, if the window is too large, the quantity of neighborhood information used is too high, producing errors in the definition of object boundaries.

Since the quality of the matching depends so strongly on the correct size of the neighborhood, an adaptive window size should be used [12]. This approach has led to several implementations in dedicated hardware [13, 14, 8]. The more recent one [8] uses an Altera field-programmable gate array (FPGA) to process 64×64 pixel grayscale images well in excess of the target frame rate of 30 fps (frames per second).

We present a new FPGA-based implementation of the same general approach, that achieves a frame rate of 25 fps for grayscale images of 208×480 pixels (on a Virtex-4 FPGA) and 640×480 pixels (on a Virtex-5 FPGA). The algorithm used is a variant of the one employed in Ref. [8]. The 208×480 version has been integrated in a system that acquires images from two CMOS image sensors and displays the calculated disparity map on a VGA monitor in real-time.

The paper is organized as follows. Section 2 describes the correspondence algorithm used. Details of the hardware implementation are presented in Section 3. A second, expanded version of the hardware architecture with increased parallelism and capable of processing larger images was also developed, and is presented in Section 4. Section 5 analyzes the quality of the disparity maps obtained and the amount of resources used. Finally, Section 6 concludes the paper.

2. The Correspondence Algorithm

The system described here extracts three-dimensional information from images by calculating their disparity maps using a variant of the algorithm proposed by [8]. This modification reduces the quantity of neighborhood information used, and enables a simplified hardware architecture, with improved resource utilization and reduction of processing time.

The steps that constitute the modified algorithm are:

1. [Initialization] The algorithm starts with a window of size $w = 8$, as in the algorithm proposed by [8], where

it is stated that this value was empirically found to be the best starting window size. The algorithm divides the reference image in a grid and the candidate image in sections. The former constitute the various reference windows (RW in Figure 1), and the latter are the candidates considered during the search (CW in Figure 1).

Referring to Figure 1, the reference window represents the window (of the reference image) for which a correspondence is sought, the candidate windows are situated along a scan-line that covers the entire search area, and MW is the matching window, i.e., the candidate window with lowest SAD score.

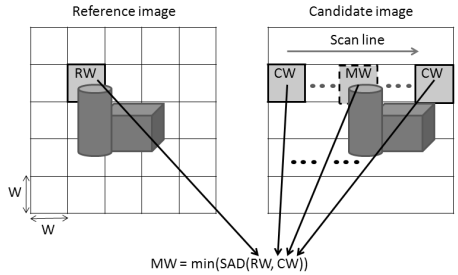


Figure 1. Set of reference and candidate windows (according to [8]). RW represents the reference window, CW the candidate windows, and MW the matching window with the lowest SAD (one of the CWs). The search for correspondence is made along the full scan-line.

2. [Select search area] Select the first section of eight lines.

   This step contains the largest difference between our algorithm and the version of [8]. While our algorithm applies the following steps to independent sections of eight lines, restricting the quantity of neighborhood information used to one section, the original one applies them to the entire image.

3. [Find best candidate] Calculate the matching between the reference window and all the candidates, by applying equation 1 to the various candidates windows and selecting the one that provides the lowest value (see also Figure 1).

$$\sum_{w}^{j=0}\sum_{w}^{i=0}|I_r(U_r+i,V_r+j)-I_c(U_c+i,V_c+j)| \quad (1)$$

   The functions $I_r(x,y)$ and $I_c(x,y)$ represent the intensity of pixels at position (x,y) in the reference and candidate images, respectively. Points (U_r,V_r) and (U_c,V_c) represent a reference pixel, which is used as the anchor point for the calculation of disparities between the reference and candidate images.

4. [Calculate disparity] Determine the disparity between the reference and the best candidate window from the previous step. Given the points corresponding to the best match (x_r,y_r) in the reference window and (x_m,y_m) in the candidate window, the disparity is given by $d=|x_r-x_m|$. (It is assumed throughout that $y_r=y_m$, i.e., the reference and candidate cameras are vertically aligned.)

   Disparity can be interpreted as the inverse of depth: pixels with larger disparities belong to objects that are nearer to the cameras.

5. [Shrink window] If $w \neq 1$, the window size is reduced by half horizontally and vertically.

   The situation of the reference and candidate windows is shown in Figure 2, where $w=8$ and the new windows RW (reference window), CW (candidate window) and MW (matching window) have an horizontal and vertical size of 4. The refined search for the matching window is restricted by considering the neighborhood information of the previous step.

   Figure 2 represents the following situation: the search of correspondence for windows with $w/2$ is restricted to the position where the two neighborhoods with $w=8$ have found the best correspondence (regions represented by shifts of $\pm d$ around the MW and CW windows, where $\pm d$ represents all the candidate windows of size 4 inside the matching window of the neighbors with size 8). In this way, the search is restricted to those 2 regions, because that is the maximum number of windows with size 8 for each window with $w=4$. This happens because the sections under consideration have a height of 8 lines, resulting in the existence of neighbors only on the left and right sides of each window. However, when $w=2$ and $w=1$, the number of neighbors of each window increases to 4 (neighbors on the left, right, above and below), and so does the amount of neighborhood information.

6. [Iterate] While $w \neq 1$, repeat from step 3.

7. [Proceed to next section] After calculating the correspondence for all pixels of a section, select the next one and repeat from step 3. If all sections have been processed, terminate.

3. System Architecture

The disparity processor implemented for this work is included in a system constituted by: a pair of CMOS cameras used to capture the images; a VGA monitor used to display the disparity maps and the reference and candidate images; and an evaluation board with the FPGA used to implement the processor and to establish the communication with the peripheral devices (CMOS camera and VGA monitor).

Image capture is done using two OV7620 CMOS cameras from OMNIVISION, which are controlled through an I2C interface. The evaluation board includes a Virtex-4 LX60 FPGA from Xilinx and all the peripherals used to communicate with the cameras and monitor. The interface

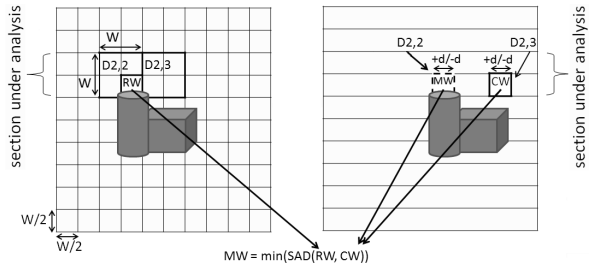


Figure 2. Set of reference and candidate windows of size $w/2$. The candidate windows analyzed at the lower window size must be inside the four regions that are within distance d from the position where the best correspondence for window size w was found.

with the VGA monitor, where the disparity maps are displayed, is made through an adapter card that takes care of all the synchronization necessary to correctly communicate with the VGA monitor.

3.1. Top-level Modules

The system implemented on FPGA is organized in the three top modules shown in Figure 3: a) data acquisition and control; b) SAD tree; c) calculation of correspondences. The last two modules together comprise the unit for the calculation of disparities. Depth map construction is done concurrently with image capture, and starts as soon as sufficient image data is available (one image section as described in Section 2).

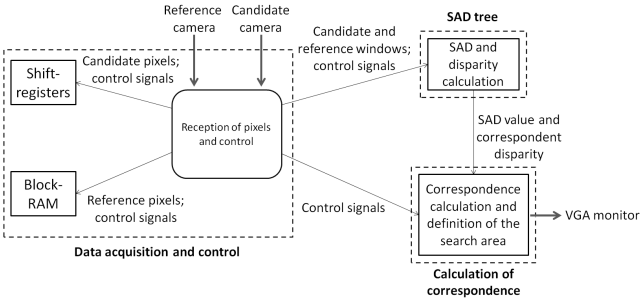


Figure 3. General view of the correspondence processor.

The module for data acquisition and control receives the pixels from the cameras and saves them in memory (shift-registers and Block-RAMs), controls of the size of the correspondence window and keeps track of its position in the image. This information allows the other modules to identify the current phase of the disparity calculation, and to update the control signals of their state machines accordingly.

The pixel intensity information and the control data concerning the windows being analyzed are sent to the module `SAD trees`, where the SAD metric (equation 1 of the correspondence algorithm) is applied to the multiple pairs of reference and candidate windows. Additionally, this module calculates the disparity associated with each one of these pairs.

The control data associated with the reference and candidate windows, and the associated disparity information are sent to the module for calculation of correspondences, which is responsible for defining the search area and for determining the best match amongst the candidate windows. The disparity for the matching window is stored in internal memory (in block RAM).

The disparity values calculated for the various windows are stored in block RAMs, whose depth and width depend on the associated window sizes. When the disparity values for all the pixels of a section have been calculated, they can be sent to the VGA monitor, while at the same time calculating and storing, in the same block RAMs, the intermediate disparities (disparities for windows of size 8, 4 and 2) of the next section. When the disparities for the windows of size one of the new section start to be calculated, the block RAMs used to store the disparities of windows of size one of the previous section are already free (because the disparities have already been sent to the VGA monitor), and can be used to store the new values.

Two different clock frequencies are used in the system: 12.5 MHz for the acquisition of pixel data from the CMOS cameras, and for sending the disparity information to the VGA interface; and 100 MHz for the core that processes the stereo images and determines the disparity information.

For the implementation of these modules various resources available on the FPGA are used: Block-RAM and shift-registers are employed for the memory structures used to save the pixels received from the cameras and the disparity information obtained for each window size; adders are used for the implementation of the SAD modules; and a DCM is used to generate the clock signals. A more extensive analysis of each of these units follows.

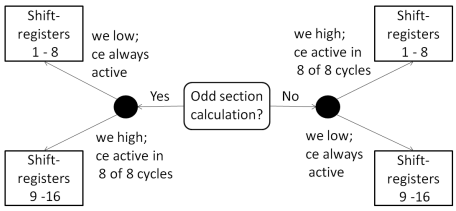
3.2. Management of Image Data

The image acquisition module uses two types of memory structures: shift registers for the pixels of the candidate image, and Block-RAM for the pixels of the reference image. This difference is justified by the different behavior of the two window types. Reference windows are shifted at least by eight positions and can, therefore, be efficiently implemented in Block-RAM. (The precise amount depends on the quantity of parallelism used: for the implementation being discussed they are shifted by 16 pixels, because the correspondence is made for two reference windows simultaneously). Candidate windows are shifted by one pixel, which is harder to implement in Block-RAM, but easily implemented by shift registers. This is another difference in comparison with the reference implementation, which uses shift-registers for both images, resulting in a significant increase in the number of logic blocks used.

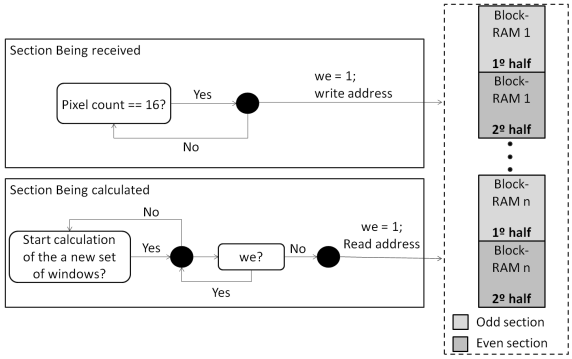
Although the pixel rate is 12.5 MHz for the two image sources, both memory structures operate at 100 MHz, since the memory units must provide image data at this rate to the disparity calculation modules.

In order to guarantee that the read and write accesses to

the memory modules are done without collisions, different approaches are used for the two types of memory structures, as shown in Figure 4. In this figure the CE and WE symbols represent the chip enable and write enable signals, respectively, and the word "section" always refers to the section of eight lines mentioned in the algorithm description (Section 2).



(a) Shift-registers access.



(b) Block-RAM access.

Figure 4. Coordinating access to image data.

The memory organization for candidate images uses two sets of shift registers: one set is used to store the section of eight image lines being analyzed at the moment, while another set is used to store the image lines being acquired at the same time. This is why the number of shift registers used in the implementation is 16. Figure 4(a) shows that the odd sections of eight lines are saved in the first eight shift-registers, and the even sections in the other eight. The depth of the shift registers is equal to the width of the images.

Figure 4(a) shows that, for write operations, each shift register is only active every eighth cycle of the 100 MHz clock. Read operations are done on every cycle, so that the pixels of the new candidate windows are sent to the SAD tree at the correct rate (100 MHz).

For the block-RAM-based reference window, the data for the section being currently processed and the section being acquired share the same physical memory, so access synchronization is more elaborate. As can be seen in Figure 4(b) each Block-RAM are divided in two halves: one half is used to save the pixels of the section being analyzed at each moment and the other half is used to save the pixels of the section being received. The first half is used to save the pixels of the odd sections of eight lines and the other half to save the pixels of the even sections.

Read access is only permitted when no write signal is active, as shown in Figure 4(b). This is done without delaying the calculation of disparities, since the write signal is only active once every 16 cycles of the 12.5 MHz clock. For each write operation, 16 pixels are committed to one memory position (Figure 4(b)). Thus, each memory position will contain all the pixels of a line of two reference windows.

The number of single-port block RAMs used for this approach (parameter n in Figure 4(b)) depends on the amount of parallelism used in the calculation of correspondences. In each cycle of the 100 MHz clock, a number of pixels equal to $8 \times 8 \times p$ (where p is the amount of parallelism supported) must be read from memory. Since the width of each block RAM is limited and it can only be accessed one position at a time (two, in the case of a dual port block RAM), it is necessary to use several block RAMs in parallel, so that a single read access provides the number of pixels required to exploit a processing core with parallelism of order p.

3.3. Calculation of Disparities

The unit responsible for the calculation of disparities is organized in two levels (see Figure 5). The first level contains the modules that calculate the SAD values (using a WPPP architecture with parallel processing of both reference and candidate windows [8]) and the corresponding disparity. The number n of SAD trees used determines the amount of parallelism used in the implementation. The second level determines the search area and calculates the correspondence for each reference window.

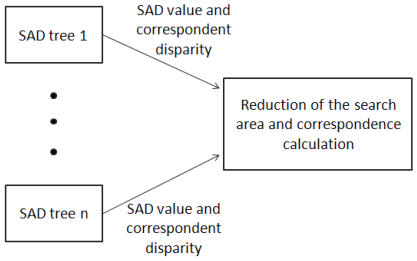
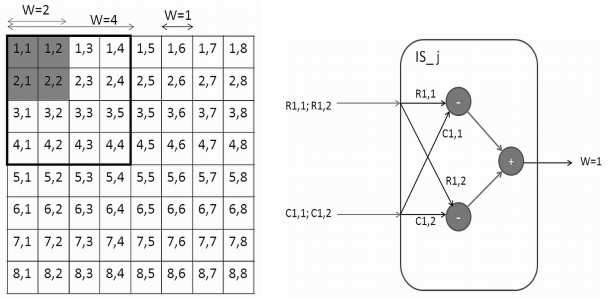


Figure 5. Calculation of disparities.

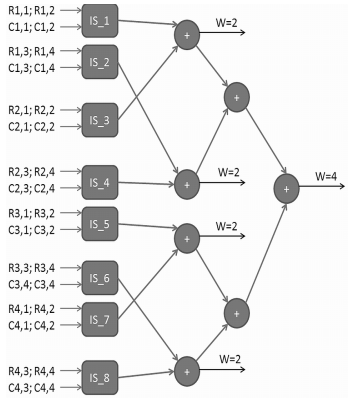
Figure 6(c) shows the constitution of each SAD tree: $R_{i,j}$ and $C_{i,j}$ represent the intensity of pixel (i, j) from the reference and candidate windows, respectively; the block IS_j is the element that calculates SADs for windows of size one (the absolute difference of two pixel values), as shown in more detail in Figure 6(b); and the rest of the SAD tree is composed of adders that combine the various absolute differences according to the window size. The example in the figure has an initial window size of four, but the analysis is valid for any size that is a power of two.

Figure 6(a)(a) represents a general correspondence window (candidate or reference). Establishing a correspondence between this window and the SAD tree of Figure 6(c) it can be seen that, for the different sizes of their sub-windows ($w = 2$ for sub-windows of size two, $w = 1$ for sub-windows of size one) the pixels considered in the calculations correspond to the pixels constituting each sub-window. This happens because the operations on SAD calculation are only additions and subtractions, allowing their

ordering on the SAD tree. With this, the pixels at each IS_j block are the same independently of the window size.



(a) Correspondence window. (b) Initial state.



(c) SAD tree.

Figure 6. General architecture used for the calculation of SAD.

This direct connections between memory modules and SAD units solve one of the major problems of this kind of implementations, that is, the need for multiplexers between memory modules and the processing units to allow a correct analysis of pixels when the window size is reduced.

Due to the block-RAM-based memory access scheme, disparity values are not obtained at a constant rate, since the speed of calculation depends on the number of cycles spent waiting until a read access is granted. In this case, the various registers keep their values until the pixels of the new window are read and the calculation is restarted. However, despite the variable data rate, a frame rate of 25 frames/second can be guaranteed.

The constant frame rate of 25 frames/second is guaranteed by the maximum processing time, which can obtained from the following expression:

$$T_{proc} = T_{store} + L_i + T_{calc} + L_f, \tag{2}$$

where:

1. T_{proc}

   is the processing time (in seconds).

2. $T_{store} = \frac{208 \times 8}{12.5 \times 10^6}$

   is the storage time for the pixels of a new section: 208×8 is the size of each section and 12.5×10^6 is the rate at which the pixels are received (in hertz).

3. $L_i = \frac{7+8+3}{10^8}$

   is the latency before the calculation. The first seven cycles of latency are due to the fact that pixels are received at a rate of 12.5 MHz, and the signal indicating that all data has been received is only updated seven cycles of the 100 MHz clock after the reception of the last pixel. The following eight cycles of latency represent the number of cycles required to store the first pixel of the new section. The final three cycles of latency are the time required to retrieve data from the block RAM and to update the synchronization signals for starting the the calculation of disparities.

4. T_{calc} is the time taken by the disparity calculation (in seconds). Due to the block RAM memory access scheme this value is variable:

   (a)

   $$\frac{208 \times \frac{208}{16} \times 4 + \frac{208}{16} \times 1 \times 4}{10^8}$$

   is the minimum time necessary for calculating disparities. $208 \times \frac{208}{16} \times 4$ is the number of shifts required to cover the section in question, for all window sizes; $\frac{208}{16} \times 1 \times 4$ is the minimum number of clock cycles (100 MHz clock) for the memory access that reads the pixels of a new window (for a complete section).

   (b)

   $$\frac{208 \times \frac{208}{16} \times 4 + \frac{208}{16} \times 8 \times 4}{10^8}$$

   is the maximum time required for calculating disparities. $208 \times \frac{208}{16} \times 4$ is the number of shifts required to cover the section in question, for all window sizes; $\frac{208}{16} \times 8 \times 4$ is the maximum number of clock cycles (100 MHz clock) that may be necessary to read the pixels of a new window due to contention during block RAM access.

5. $L_f = \frac{4+2}{10^8}$

   is the latency after the beginning of calculation. The first four cycles of latency come from the delay between the beginning of the displacement of candidate windows and the return of the first SAD. The following two cycles of latency represents the number of clock cycles between the return of the first SAD value and the return of their disparity information (100 MHz clock).

This results in a minimum processing time of 0.242 ms and a maximum of 0.246 ms. This processing time, together with the frame rate restriction of the external hardware (CMOS cameras and VGA monitor) allows, as already stated, an image processing rate of 25 frames per second.

4. Expansion of the Architecture

The size of the image processed is highly dependent on the amount of resources available to implement the architecture presented in section 3.

There are three units that may limit the size of the images processed, due to lack of FPGA resources. They are: the shift-registers used to store the pixels of the candidate images; the adders used to implement the SAD tree; and the logic path (number of slices) used to define the search area. The last two units are fundamental to implement the parallelism necessary to satisfy the real-time requirements of the task.

Thus, to increase the image dimensions it is necessary to have enough resources to: increase the depth of the shift-registers used to save the pixels from the candidate image; increase the quantity of parallelism used to calculate the disparities; and increase the number of block RAMs used to store the pixels from the reference image and the disparities calculated for the various window sizes. Although the block RAMs are a fundamental unit of the correspondence processor, they do not represent a limitation in the hardware platform used, since their occupation is below 50%, as can be seen on table 2.

An expansion of the architecture from the previous section was implemented in a Virtex-5 LX330. The new version is capable of handling images of size 640×480. The quantity of parallelism necessary to cope with the larger image size, while keeping the frame rate of 25 frames/second was obtained from the following expression:

$$T_A = \frac{640 \times \frac{640}{Q_{par}} \times 4 + \frac{640}{Q_{par}} \times 8 \times 4}{10^8} = \frac{640 \times 8}{12.5 \times 10^6}$$

where:

1. $T_A = \frac{640 \times 8}{12.5 \times 10^6}$
   is the storage time for the pixels of one section;
2. $\frac{640 \times \frac{640}{Q_{par}} \times 4 + \frac{640}{Q_{par}} \times 8 \times 4}{10^8}$
   is the maximum time for disparity calculation;
3. Q_{par} is the amount of parallelism, i.e., the width of pixels analyzed concurrently. For example, with a parallelism of 2 windows, the width is $8 \times 2 = 16$ pixels.

Therefore, the amount of parallelism that must be supported is $Q_{par} = 40.5$. This corresponds to $\frac{40.5}{8} = 5.06 \rightarrow 6$ windows analyzed in parallel.

Although a minimum of six windows is required, the expanded architecture parallel uses eight. The reason is that six is not a divisor of 640, so it would be necessary to introduce additional circuitry to control the displacement of the reference windows, making the implementation more complicated.

Although the expanded architecture has been validated for images with 640×480 pixels, it is able to process 1016-pixel wide images. Only the depth of the shift-registers and block RAMs needs to be increased appropriately (which is feasible for the Virtex-5 LX330).

5. Results

5.1. Disparity maps

Using a simplified version of the reference algorithm, as discussed in section 2, does not result in a serious impact on the disparity map obtained. The confirmation of this result was done by comparing each pixel of the disparity map obtained by the reference algorithm with the corresponding pixel of the disparity map obtained by the simplified algorithm. This evaluation was made using images from the database presented in [15].

For both processors, the one implemented in the Virtex-4 LX60 and the one implemented in Virtex-5 LX330, it was necessary to cut the images available to the size processed by each implementation. Since the images of the database are in color, we converted the original images from the PNG (Portable Network Graphics) format to the PGM (portable gray map) format, which is easier to use.

We compared the results of our implementation of the reference algorithm in Matlab with the outputs of the Verilog description of the matching processor as executed on the Modelsim simulator. The disparity maps obtained from the Verilog version were compared pixel by pixel with the disparity maps produced by the Matlab version.

Results are shown in Figure 7 and in Table 1. The comparison does not include those pixels of the right region of the reference image that are not presented in the candidate image, since they do not have "correct" disparity values in either case (since no corresponding object actually exists). Although the test was done for the Virtex-4 LX60 and Virtex-5 LX330 implementations, only the results for the Virtex-4 LX60 study are shown, since the others are similar.

Table 1. Mean of the absolute differences of disparities.

	Mean (pixel distance)
Test image 1	3.201
Test image 2	2.49
Test image 3	1.343
Test image 4	1.16

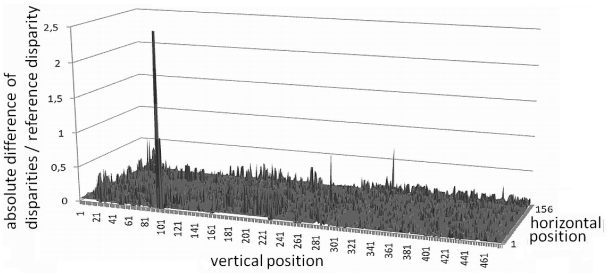


Figure 7. Relative difference of disparities for test image 4. The results for other images are similar.

Table 1 shows that for four different images the mean

absolute difference of disparities is very low, with a maximum value of three. "Pixel distance" is the difference of pixel position determined by the implemented algorithm and the one found by the original algorithm.

For most pixels ($\geq$90%) the absolute difference of disparities is less or equal to seven. Figure 7 represents the relative error (difference between hardware disparity and reference disparity as obtained by the original algorithm, divided by the reference disparity) for test image 4. The axes of Figure 7 labeled "vertical position" and "horizontal position" represent the vertical and horizontal position of each pixel. Similar results were obtained for all other tests. Figure 7 shows that the ratio is always near to zero, except for some sporadic peaks (the major one reaches a value of 2.3), which are due to occlusion. Because of that phenomenon, the disparity values for those regions are considerably different in both versions, since they react differently in this case.

The occlusion phenomenon mentioned in the previous paragraph occurs because of two reasons: some objects are not represented in the candidate image, since they are occluded by an object closer to the cameras; objects located at the right limit of the reference window are not represented in the candidate image.

5.2. Resource Utilization

Table 2 summarizes the utilization of FPGA resources for the two versions of the image matching processor. Comparison of the slice utilization in the two processors cannot be made directly, since slices in the Virtex-4 architecture (two flip-flops and two 4-input look-up tables) are different from slices in the Virtex-5 architecture (four flip-flops and four 6-input look-up tables). The high-level synthesis process was oriented towards maximizing clock frequency at the expense of FPGA occupation.

Table 2. Resource utilization for baseline and expanded processors.

Resources	Utilization (number)	Utilization (%)
Virtex-4 LX60 (208×480 pixels)		
Slices	20101	75
Block-RAM	64	40
Virtex-5 LX330 (640×480 pixels)		
Slices	50340	24
Block-RAM	168	58

Analyzing the occupation of the Virtex-4 LX60, the high utilization of slices is mainly due to the shift registers used for storage of two sections of 208×8 pixels of the candidate image, and to the quantity of parallelism used in the calculation of the disparities. The quantity of block RAM used is due to the storage of the reference image and the disparity values at each window size (for windows of size one, two, four and eight).

For the larger implementation on the Virtex-5 LX330, the number of slices used is determined by the shift registers, but also by the amount of parallelism used in this implementation, which is 4 times higher than the implementation on the Virtex-4 LX60. The number of block RAMs increases greatly for the same reason. This happens because the only way to increase the quantity of information that is available at each instant is to instantiate more block RAMs.

We can be concluded that the resource utilization depends on the size of the images analyzed and on the frame rate required. Images with larges dimensions require deeper shift-registers, in order to store the sections of eight lines of the candidate image. Relatively to the frame rate, higher frame rates require higher parallelism, which implies: the use of more block RAMs, needed to save all the reference windows analyzed at each instant; and the increase of slice utilization to implement the SAD and all the logic necessary to achieve the required parallelism.

The comparison between the implemented processor and the reference implementation, relatively to resources utilization, needs to be done carefully, since the types of FPGA used are different. However, the reference implementation [8] presents higher resource utilization, since it uses 42,508 logic elements of an APEX20KE from Altera (each consisting of a 4-input look-up table and one flip-flop), while the smaller of our implementations uses 19,978 slices of a Virtex-4 (two 4-input look-up tables and two flip-flops), for a total of 31,880 look-up tables and 16,951 flip-flops).

The lower resource usage of the proposed architecture is due mainly to the reduction of neighborhood information processed, and to the use of block RAM for storing the pixels of the reference image (instead of shift-registers).

5.3. Comparison with Other Approaches

This subsection presents a comparison between the implementation described in this paper with previously reported results, relatively to the dimensions of the images analyzed and the velocity of the processor. Table 3 summarizes the data. Column "time" represents the time spent to process one frame. For both implementations proposed in this paper, the one on Virtex-4 (Impl. 1 (V4)) and the one on Virtex-5 (Impl. 2 (V5)), the processing time is 40 ms, since both have a frame rate of 25 frames per second ($\frac{1}{25} = 40$ ms). Both are faster than the previously reported ASIC implementations [13, 14], but support a smaller maximum window size. Comparing with the FPGA implementation of Ref. [8], our implementations are able to process much larger images, while still satisfying real-time requirements.

6. Conclusion

This paper describes a hardware architecture for the calculation of dense depth maps from a pair of stereo images. The architecture is based on a modification of a previously reported variable-window-size method. Empirical tests indicate that the simplification introduced does not degrade

Table 3. Comparison between the proposed implementations and previous systems reported in the literature.

System	Image size	Max. window size	Freq. (MHz)	Time (ms)
Ref.[13]	512×512	25×25	200	60
Ref.[14]	320×240	15×15	125	100
Ref.[8]	64×64	8×8	86	0.19
Impl. 1 (V4)	208×480	8×8	100 MHz	40
Impl. 2 (V5)	640×480	8×8	100 MHz	40

The first two systems are implemented with CMOS ASICs: 0.5 µm and 0.18 µm technologies, respectively. Implementation [8] uses an FPGA from Altera (APEX20KE). The last two lines summarize the implementations described in this paper. All systems process 8-bit grayscale images.

the quality of the resulting depth maps. The proposed architecture admits implementations with a variable degree of parallelism, depending on the resources available. The architecture exploits the resources of modern platform FPGAs. In particular, the management of image data uses different memory resources for the reference and the candidate image, in order to take advantage of the different access patterns.

Two versions of the architecture with different resource requirements were implemented. Both produce dense depth maps in real-time (25 maps per second). The smaller implementations targets a Virtex-4 LX40 device and handles 208×480 images, while the larger one may use a Virtex-5 LV330 device (less than 60% of resource occupation) and handles 640×480 images. Additionally both are capable of finding a maximum disparity of 255.

Relatively to the velocity of the processor and the dimension of the images analyzed, the new implementations are faster than the previously reported ASIC implementations [13, 14], but support a smaller maximum window size. They are able to process much larger images than the FPGA implementation of Ref. [8], while still satisfying real-time requirements, as presented in Table 3.

References

[1] M. Z. Brown, D. Burschka and G. D. Hager, Advances in computational stereo, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.25, no.8, pp. 993-1008, Aug. 2003.

[2] D. Murray and J.J. Little, Using Real-Time Stereo Vision for Mobile Robot Navigation, *Autonomous Robots*, vol. 8, Abr. 2000, pp. 161-171.

[3] U. Franke and S. Heinrich, Fast obstacle detection for urban traffic situations, *IEEE Transactions on Intelligent Transportation Systems*, vol.3, no.3 (2002), pp. 173-181, 2002.

[4] J. M. Manendez. L. Salgado, E. Rendon and N. Garcia, Motorway surveillance through stereo computer vision, *IEEE 33rd Annual 1999 International Carnahan Conference on Security Technology*, pp.197-202, 1999.

[5] D. Scharstein and R. Szeliski, A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms, *International Journal of Computer Vision*, vol. 47, Abr. 2002, pp. 7-42.

[6] M. Kuhn, S. Moser, O. Isler, F. Gurkaynak, A. Burg, N. Felber, H. Kaeslin and W. Fichtner, Efficient ASIC implementation of a real-time depth mapping stereo vision system, *Proceedings IEEE International Symposium on Micro-NanoMechatronics and Human Science*,vol. 3, 2003, pp. 1478-1481.

[7] J. Woodfill, G. Gordon and R. Buck, Tyzx DeepSea High Speed Stereo Vision System, *Computer Vision and Pattern Recognition Workshop CVPRW '04*, 2004, p. 41.

[8] M. Hariyama and Y. Kobayashi and H. Sasaki and M. Kameyama , FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture, *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E88-A** (2005) 3516–3522.

[9] S. Lee, J. Yi and J. Kim, Real-Time Stereo Vision on a Reconfigurable System, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2005, pp. 299-307.

[10] L. Mingxiang and J. Yunde, Stereo Vision System on Programmable Chip (SVSoC) for Small Robot Navigation, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 1359-1365.

[11] R. Porter and N. Bergmann, A generic implementation framework for FPGA based stereo matching, *Proceedings of the IEEE Region 10 Annual Conference on Speech and Image Technologies for Computing and Telecommunications*, vol. 2, 1997, pp. 461-464.

[12] T. Kanade and M. Okutomi, A stereo matching algorithm with an adaptive window: theory and experiment, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, 1994, pp. 920-932.

[13] M. Hariyama, T. Takeuchi and M. Kameyama, VLSI processor for reliable stereo matching based on adaptive window-size selection, *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, 2001, pp. 1168-1173.

[14] M. Hariyama and M. Kameyama, VLSI processor for reliable stereo matching based on window-parallel logic-in-memory architecture, *Digest of Technical Papers Symposium on VLSI Circuits*, 2004, pp. 166-169.

[15] D. Scharstein and R. Szeliski, Middlebury Stereo Vision, June 2009, `http://vision.middlebury.edu/stereo/data/`