Using FPGAs for Real-Time Disparity Map Calculation

Carlos Resende
DEEC, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
Email: ee04022@fe.up.pt

João C. Ferreira
INESC Porto, Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
Email: jcf@fe.up.pt

***Abstract*—Real-time stereo image matching is an important computer vision task. This paper presents the architecture and implementation of an FPGA-based stereo image processor, that produces 25 dense depth maps per second from pairs of 8-bit-per-pixel gray-scale images. The system implements a modification of a previously-reported variable-window-size method to determine the best correspondence for each image pixel. 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). The proposed 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 , while the larger one works for images of size 640×480 (both operate at 100 MHz). These results improve on previously-reported ASIC and FPGA-based designs.**

I. Introduction

Acquisition of three-dimensional information from images has important applications in computer vision [1] (e.g., in robotics, driver assistance and surveillance). When two vertically-aligned cameras are available, their stereo images can be used produce dense disparity maps. Taking one of the images as reference, such a map gives, for each image, the horizontal distance to the corresponding pixel in the other image. This distance (the disparity) is inversely proportional to the distance between cameras and the object (the depth).

The calculation of disparity maps requires the reliable establishment of correspondences between the images [2]. The computational effort of this task typically precludes achieving real-time performance with general-purpose processors in embedded systems, and has led to the development of many dedicated hardware systems [3]–[7].

A general approach to the determination of the correspondences is to make 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 reference pixel (in the first image). Various metrics have been proposed to evaluate the similarity of the neighborhoods [8], but the one based on the sum of absolute differences (SAD) of all the neighborhood pixels is often chosen for hardware implementations due to its simplicity.

The size of the neighborhood (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 neighborhood may cover pixels from objects at different depths, producing errors in the definition of the disparity near object boundaries. For this reason, the use of an adaptive window has been proposed by [9], and has been implemented in several dedicated systems [5], [10], [11]. The most recent one uses an Altera FPGA to process 64×64 pixel gray-scale images well in excess of the target frame rate of 30 fps (frames per second) [5].

The present work describes a new FPGA-based implementation of the same general approach, achieving a frame rate of 25 fps for gray-scale images of 208×480 pixels (on a Virtex-4 FPGA) and 640×480 pixels (on a Virtex-5 FPGA). The 208×480 version has been used in a hardware prototype 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 II describes the correspondence algorithm. The main aspects of the implementation are presented in Section III. Section IV analyzes the implementation results and compares them to previous approaches. Section V concludes the paper.

II. The Image Matching Algorithm

The implementations presented in sec. III calculate dense disparity maps using a variation of the algorithm from [5]. The modification restricts the quantity of neighborhood information used, and enables a simplified hardware architecture, with improved resource utilization and reduction of processing time. Empirical tests show that the impact on map quality is small (cf. sec. IV-A). The steps of the algorithm are:

1. [Initialization] The algorithm starts with a window of size $w = 8$, as proposed by [5], who empirically determined this value to be the best starting window size. The algorithm divides the reference image in a grid and the candidate image in horizontal sections. The former are the various reference windows (RW in fig. 1), and the latter are the candidates considered during the search (CW in fig. 1).

As shown in fig. 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 of the candidate image, and MW is the matching window, i.e., the candidate window with lowest SAD score.

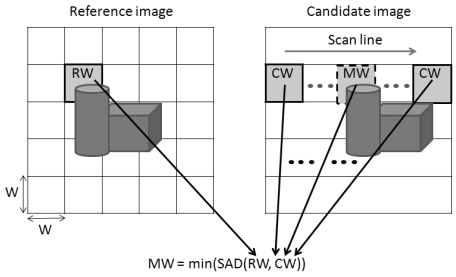


Fig. 1. Reference and candidate windows (following [5]). RW is the reference window, CW the candidate windows, and MW the matching window with the lowest SAD (one of the CWs). The search for correspondence is made along the full scan line to the right of the RW's position.

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

This step represents the most significant difference between our algorithm and the one in [5]. We apply the following steps to independent sections of eight lines, restricting the quantity of neighborhood information used to one section, while the original approach applies them to the entire image. It is assumed throughout that the cameras are vertically aligned.

3. [Find best candidate] Calculate the SAD between the reference window and each candidate. Select the candidate with the lowest SAD score (MW).

4. [Calculate disparity] Determine the disparity between RW and MW. The disparity is given by the difference of the horizontal coordinates of the two windows, RW and MW.

5. [Shrink window] If $w \neq 1$, the window size is reduced by half horizontally and vertically. If $w = 1$, the desired disparity has been calculated: go to step 7.

The new situation of the reference and candidate windows is shown in fig. 2, where $w = 8$ and the new windows RW, CW and MW have an horizontal and vertical size of 4. The next search for the matching window will be limited by using the neighborhood information of the present step.

For windows with $w/2 = 4$, the search will be restricted to two neighborhoods with $w = 8$, which are the regions represented by shifts of $\pm d$ around the MW and CW windows, so that all candidate windows of size 4 are inside the matching window of size 8. The search is restricted to two regions, because that is the maximum number of windows with $w = 8$ required to cover all windows with $w = 4$: sections have a height of 8 lines, and therefore only neighbors on the left and right sides of each window must be considered. For the cases with $w = 2$ and $w = 1$, the number of neighbors of each window increases to 4 (neighbors on the left, right, above and below).

6. [Repeat with smaller window sizes] If $w \neq 1$, repeat from step 3.

7. [Change reference window] Make $w = 8$, shift RW to the right and repeat from step 3, in order to find the disparity associated with the next pixel.

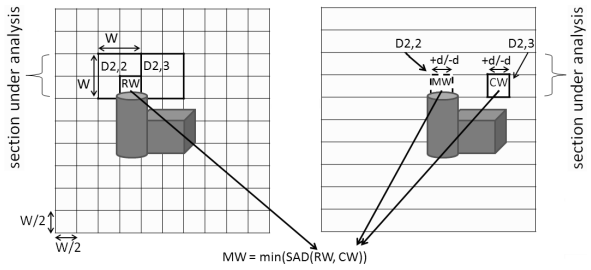


Fig. 2. Reference and candidate windows of size $w/2$. The candidate windows analyzed at the lower window size must be inside the four regions that are within distance d from the position where the best correspondence for window size w was found.

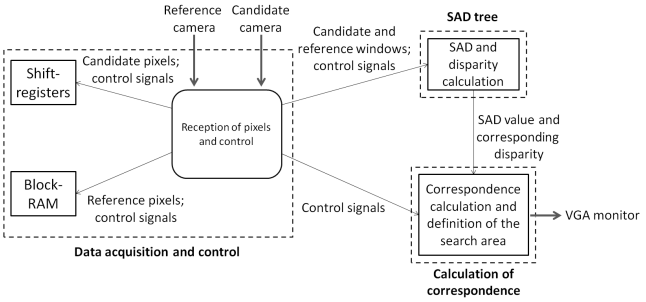


Fig. 3. Top-level view of the image correspondence processor.

8. [Proceed to next section] After calculating the correspondence for all pixels of a section, select the next one and repeat from step 2. If all sections have been processed, stop.

III. System Architecture and Implementation

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 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.

A. Top-level Organization

The system implemented on FPGA has three top modules, as shown in fig 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 one image section is available.

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 is applied to the multiple pairs of reference and candidate windows. The module also 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 (block RAM).

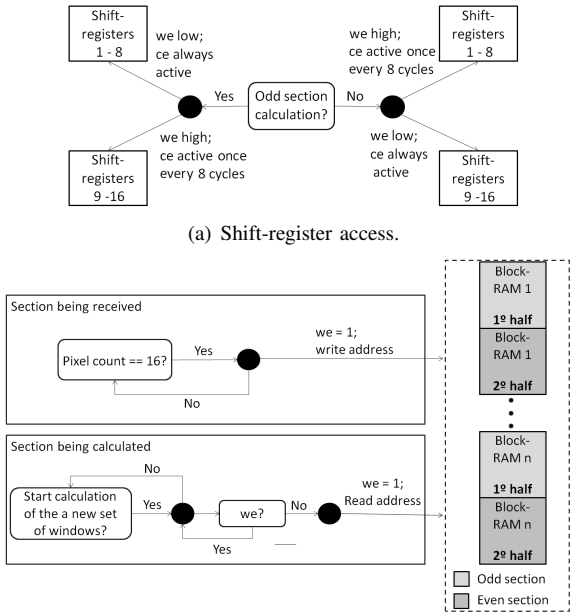
The disparity values calculated for the various windows are stored in block RAMs, whose depth and width depend on the associated window sizes. When the disparity values for all the pixels of a section have been calculated, they are sent to the output (a VGA monitor in our demonstration system), while 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 1 of the new section start to be calculated, the block RAMs used to store the disparities of windows of size 1 of the previous section are already free, and can be used to store the new values.

The implementation uses two different clock frequencies: 12.5 MHz for the acquisition of pixel data from the CMOS cameras, and for sending the disparity information to the VGA interface; 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-RAMs and shift-registers are employed for the memory structures used to save the images and the disparity information obtained for each window size; adders are used for the implementation of the SAD modules; and one digital clock manager is used to generate the clock signals. A more detailed analysis of each of these units follows.

B. 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 RAMs for the pixels of the reference image. This difference is due to the different behavior of the two window types. Reference windows are shifted by at least eight positions and can, therefore, be efficiently stored in block RAM. (The precise shift 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



(a) Shift-register access.

(b) Block-RAM access.

Fig. 4. Coordination of the access to internal image data.

shifted by one pixel, which is hard to implement with block RAM, but easy to do with shift registers. The implementation of [5] 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 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 fig. 4. In the figure, the CE and WE symbols represent the chip enable and write enable signals of the shift-registers and block RAMs, respectively. The word "section" always refers to the section of eight lines mentioned in the algorithm description.

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 8-bit-wide 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. 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 of 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 syn-

chronization is more elaborate. As shown in fig. 4(b) each block RAM is divided in two halves: one half is used to save the pixels of the section being analyzed, while the other half is used to save the pixels of the section being received. Read access is only permitted when no write signal is active. 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. 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 fig. 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, 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.

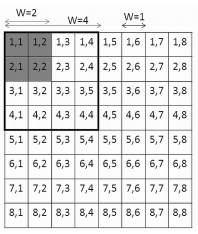
C. Calculation of Disparities

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

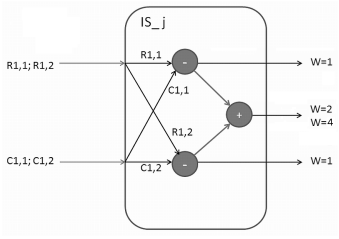
Figure 5(c) shows the structure of each SAD tree: $R_{i,j}$ and $C_{i,j}$ represent the intensity of pixel (i, j) from the reference and candidate windows, respectively; the block IS_j is the element that calculates SADs for windows of size one (the absolute difference of two pixel values), as shown in more detail in Figure 5(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 5(a) represents a general correspondence window (candidate or reference). Establishing a correspondence between this window and the SAD tree of fig. 5(c) shows that, for the different sizes of their sub-windows, the pixels considered in the calculations have always the same position in the SAD tree, which eliminates the need to use multiplexers between the memory modules and the calculation modules. This happens because the operations used for SAD calculation are only additions and subtractions, which can be re-ordered for each different window size in such a way that the pixels at the inputs of each IS_j block are always the same.

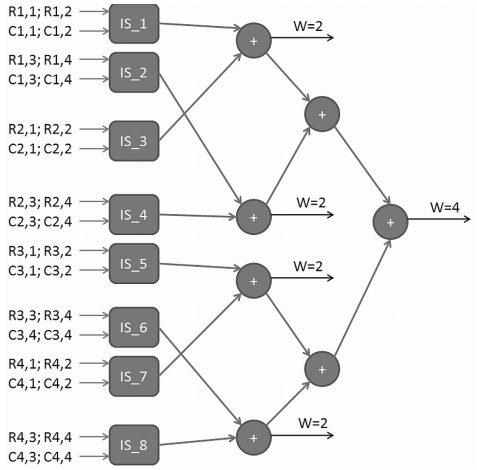
The direct connection between memory modules and SAD units solves one of the major problems of this type of implementation, i.e, the need for multiplexers between memory



(a) Window under analysis (candidate or reference).



(b) Basic block of the SAD tree. This block calculates the SAD for two windows o size one.



(c) SAD tree for windows of size 4. This block calculates the SAD for 1 window of size 4, 4 windows of size 2, or 16 windows of size 1.

Fig. 5. General architecture used for SAD calculation

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 still be guaranteed.

D. Determination of Processing Time

For the proposed implementation (100 MHz main clock and 12.5 MHz pixel clock), the processing time T_{proc} is given by:

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

where T_{store} is the storage time for the pixels of a new section, L_i is the latency before the calculation, T_{calc} is the time taken by the disparity calculation, and L_f is the latency after the beginning of calculation (before the first SAD value is produced). As discussed before, T_{calc} is variable due to the block RAM memory access scheme.

This results in a minimum processing time of 0.242 ms and a maximum of 0.246 ms for our Virtex-4 implementation. 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.

E. 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 III.

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 unit types 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: a) increase the depth of the shift-registers used to save the pixels from the candidate image; b) increase the quantity of parallelism used to calculate the disparities; and c) 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 this case, since their utilization is below 50% (cf. tab. I).

An expanded version of the proposed architecture was implemented in a Virtex-5 LX330. The new version is capable of handling images of size 640×480. The amount of parallelism necessary to cope with the larger image size can be obtained by equating the storage time for one section with the maximum time for disparity calculation, since one image section must be processed while the next one is being loaded:

$$\frac{640 \times 8}{12.5 \times 10^6} = \left(640 \times \frac{640}{p} \times 4 + \frac{640}{p} \times 8 \times 4\right) \times 10^{-8}$$

Here, p is the amount of parallelism, i.e., the number of pixels analyzed concurrently (8 per window). In our case, $p = 40.5$, so it is necessary to process 6 reference windows in parallel ($40.5/8 = 5.06 \rightarrow 6$).

Since 640 is not evenly divisible by 6, our expanded implementation processes eight candidate windows in parallel, which simplifies the circuitry to control the displacement of the reference windows. The expanded version has been validated for images with 640×480 pixels, but 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).

IV. Discussion

The stereo image processor presented in this paper was validated by software and hardware analysis. The software validation confirmed the results obtained by our processor by comparison with the results obtained by the reference one. While the hardware validation confirmed the functionality of the processor with real video images.

TABLE I
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

TABLE II
Comparison between the proposed implementations and previous systems reported in the literature

System	Image size	Max. window size	Freq. (MHz)	Time (ms)
Ref. [10]	512×512	25×25	200	60
Ref. [11]	320×240	15×15	125	100
Ref. [5]	64×64	8×8	86	0.19
Impl. 1 (V4)	208×480	8×8	100	40
Impl. 2 (V5)	640×480	8×8	100	40

The first two systems are implemented with CMOS ASICs: 0.5 µm and 0.18 µm technologies, respectively. Implementation [5] uses an FPGA from Altera (APEX20KE). The last two lines summarize the implementations described in this paper. All systems process 8-bit gray-scale images.

A. Disparity maps

The modified version of the of the correspondence algorithm does not result in a serious impact on the resulting disparity map. This was confirmed by comparing each pixel of the disparity map obtained by the original algorithm with the corresponding pixel of the disparity map obtained by the modified algorithm. This evaluation was made using images from the database presented in [12] (after conversion to gray-scale).

We compared the results of our implementation of the original algorithm in Matlab with the outputs of the Verilog description of the matching processor (as obtained from a functional simulation of the Verilog source code). The disparity maps produced by the Verilog version were compared pixel by pixel with the results produced by the Matlab version. For a set of four pairs of images, the mean absolute difference of disparities was very low, ranging between 1.3 and 3.2.

B. Resource Utilization

Table I 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 maximized clock frequency at the expense of FPGA occupation.

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 for each window size.

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 for the implementation on the Virtex-4 LX60. The number of block RAMs also increases greatly for the same reason, because the only way to increase the quantity of information available in 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 large dimensions require deeper shift-registers in order to store the sections of eight lines of the candidate image. Higher frame rates require more parallelism, which implies: a) the use of more block RAMs, needed to save all the reference windows analyzed at each instant; and b) the increase of slice utilization to implement the SAD and all the logic necessary to achieve the required parallelism.

C. Comparison with Other Approaches

The comparison of resource utilization between the our processor and the implementation of [5], needs to be done carefully, since the types of FPGA used are different. The implementation of [5] has 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).

Table II compares image size and processor speed for designs with different architectures. Column "time" shows the time required to process one frame. Note that a direct comparison is complicated by differences in technology. For both implementations proposed in this paper, on Virtex-4 (Impl. 1 (V4)) and on Virtex-5 (Impl. 2 (V5)), the processing time is 40 ms, since both target a frame rate of 25 frames per second. Both are faster than the previously-reported ASIC implementations ([10], [11]), but support a smaller maximum window size. However, as occurs with the comparison with the implementation of Ref. [5], these results are influenced not only by the algorithm and its implementation, but also by the technology used. FPGA technology and organization differ significantly from ASICs, a factor that influences the comparison results. Comparing with the FPGA implementation of Ref. [5], our implementations are able to process much larger images, while still satisfying real-time requirements.

V. Conclusion

This paper described and evaluated 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 [5]. Empirical tests indicate that the simplification introduced does not degrade the quality of the resulting depth maps. The proposed architecture exploits the resources of modern platform FPGAs, and allows the creation of implementations with a variable degree of parallelism, depending on the available resources. The management of image data uses different memory resources for reference and candidate images 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 implementation targets a Virtex-4 LX40 device and handles 208×480 images, while the larger one uses a Virtex-5 LV330 device (less than 60% of resource occupation) and handles 640×480 images. Both are capable of finding a maximum disparity of 255. They are faster than the previously reported ASIC implementations [10], [11], but support a smaller maximum window size. They are able to process in real-time much larger images than the FPGA implementation of Ref. [5].

Acknowledgment

The present work was partially supported by research contract PTDC/EEA-ELC/69394/2006 from the Foundation for Science and Technology (FCT), Portugal.

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. 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.

[3] 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, *Proc. IEEE Intl. Symp. Micro-NanoMechatronics and Human Science*,vol. 3, 2003, pp. 1478-1481.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] R. Porter and N. Bergmann, A generic implementation framework for FPGA based stereo matching, *Proc. IEEE Region 10 Ann. Conf. Speech and Image Tech. for Comp. and Telecom.*, vol. 2, 1997, pp. 461-464.

[9] 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.

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

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

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