J Real-Time Image Proc (2016) 11:473–485
DOI 10.1007/s11554-013-0338-1

ORIGINAL RESEARCH PAPER

Scalable hardware architecture for disparity map computation and object location in real-time

Pedro Miguel Santos · João Canas Ferreira · José Silva Matos

Received: 31 July 2012 / Accepted: 12 March 2013 / Published online: 1 June 2013

Abstract We present the disparity map computation core of a hardware system for isolating foreground objects in stereoscopic video streams. The operation is based on the computation of dense disparity maps using block-matching algorithms and two well-known metrics: sum of absolute differences and Census transform. Two sets of disparity maps are computed by taking each of the images as reference so that a consistency check can be performed to identify occluded pixels and eliminate spurious foreground pixels. Taking advantage of parallelism, the proposed architecture is highly scalable and provides numerous degrees of adjustment to different application needs, performance levels and resource usage. A version of the system for 640 × 480 images and a maximum disparity of 135 pixels was implemented in a system based on a Xilinx Virtex II-Pro FPGA and two cameras with a frame rate of 25 fps (less than the maximum supported frame rate of 40 fps on this platform). Implementation of the same system on a Virtex-5 FPGA is estimated to achieve 80 fps, while a version with increased parallelism is estimated to run at 140 fps (which corresponds to the calculation of more than 5.9×10^9 disparity-pixels per second).

P. M. Santos (✉)
Faculdade de Engenharia, Universidade do Porto, Porto, Portugal
e-mail: pedro.miguel.santos@fe.up.pt

J. C. Ferreira · J. S. Matos
INESC TEC and Faculdade de Engenharia,
Universidade do Porto, R. Dr. Roberto Frias,
4200-465 Porto, Portugal
e-mail: jcf@fe.up.pt

J. S. Matos
e-mail: jsm@fe.up.pt

Keywords Dense disparity map · Reconfigurable embedded system · Real-time image processing

1 Introduction

Many applications require an efficient way to compute the distance of objects in a scene to a camera or image sensor, as is the case of automobile crash-avoidance systems [18], or human-computer interface systems such as the one described in [9, 11], which inspired the present work. In that system, the user's hand position is detected and its coordinates sent to a computer so that the hand itself serves as pointing device, much like a computer mouse.

A typical stereoscopic setup uses two side-by-side cameras for capturing images of a scene from two slightly different viewpoints. The position of a given object will exhibit a relative displacement in the two images, which is inversely proportional to the object's closeness to the cameras. This displacement is called disparity: objects with a large disparity are close to the cameras, while those with small disparity are farther away. Image processing techniques, such as block-matching algorithms, can be applied to stereo image pairs to compute the distance of all points in a scene to the cameras, thus generating dense disparity maps from which depth maps can be calculated. Disparity maps can be used as a means to obtain a segmentation of the scene into objects, by aggregating pixels that have similar disparities (or in other words, points at similar distance from the cameras). Identification of foreground objects is a direct application of this technique.

This work presents the implementation of a system for isolating foreground objects, which is capable of processing pairs of 640 × 480 images (8-bit pixels) at a frame rate of 40 frames per seconds (fps), detecting a

maximum disparity of 135 pixels. It was deployed in a hand-tracking application operating at a frame rate of 25 fps, the maximum rate allowed by the used cameras. The hardware platform used in this application uses a Xilinx Virtex-II Pro xc2vp30 FPGA. The same architecture, with increased parallelism, achieves 140 fps on a Virtex-5, as estimated by post-layout timing analysis. The proposed architecture is scalable and adjustable with respect to several system parameters, such as maximum attainable disparity, window and image size. The use of an FPGA as the hardware platform allows system reconfiguration for different application needs and different trade-offs of resolution, speed and power consumption for the same application.

The overall data flow of our system is as follows: stereoscopic image pairs arrive from two CMOS cameras set side-by-side. A disparity for each pixel of an image is computed by finding the displacement, in pixels, that corresponds to the same point of the scene in the other image, using a block-match algorithm with two different similarity metrics: sum of absolute differences (SAD) and Census. Pixels from the right image are searched in the left one and vice-versa, thus resulting in two independent disparity maps. These are then thresholded to isolate the foreground pixels, and a consistency check takes place over the two resulting bitmaps to eliminate spurious assignments to the foreground due to occlusion. Finally, the coordinates of the center of gravity of the foreground pixels are computed.

The major innovative characteristics of the proposed architecture are

- expandable architecture that can be used to generate implementations for various image sizes and frame rate requirements (by increasing the number of similarity calculation modules);
- parallel computation of the disparity of several neighboring pixels (minimizing data transfers between memory and similarity calculation modules);
- highest maximum disparity among reported implementations;
- highest frame rate for implementations that support two similarity metrics.

The remainder of the article is organized as follows: Section 2 provides background information of disparity map determination from stereoscopic images and reviews previous work. Section 3 then provides an overall description of the system's architecture. The hardware modules and their interaction are described in detail in Sect. 4. Section 5 analyses how system parameters can be adjusted to meet different goals and quantifies system performance. Resource usage and comparison with other implementations are discussed in Sect. 6. Finally, Sect. 7 presents the conclusions.

2 Background and related work

2.1 Depth-extraction algorithms using block matching

Computing the disparity of a point in one image requires finding its corresponding point in a second image and computing the displacement in number of pixels. Repeating this operation for all pixels of an image yields a *dense disparity map*.

An extensive overview of disparity-computation algorithms is given by [25]. Due to their good compromise between efficiency and complexity, block matching algorithms are commonly used for computing this displacement in hardware implementations. For each pixel in an image of the stereoscopic pair, a block matching algorithm picks a block of pixels around it (with at least 3×3 pixels) and compares it with blocks of pixels extracted from the other image, along an adequate search area. The horizontal difference in number of pixels between the location of the initial block and the block that was found to be most similar in the other image, is the disparity. For convenience, the block being searched for will be named as reference block, and blocks drawn from the search area on the other image will be named candidate blocks; Fig. 1 shows these concepts and how the process takes place. Note that it is sufficient to perform the search in the right

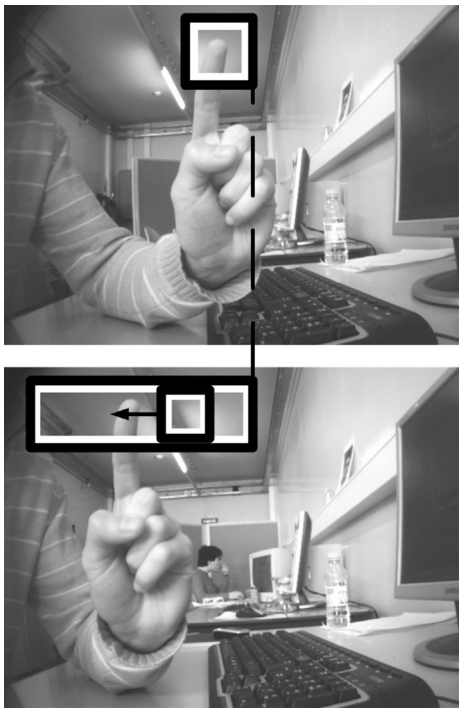

Fig. 1 The *top image* refers to the *left image* of the stereoscopic pair and a reference block for which a match is to be found in the right image. The *bottom image* shows the area to be swept in search of a similar block and, within it, a candidate block

image starting from the X-coordinate of the reference block.

For a complete disparity map, the search process must be repeated for all pixels in the image, which means that a large number of similar operations with small mathematical complexity will be required, thus rendering the system susceptible to take advantage of a parallel implementation.

Various metrics have been used to assess the similarity of two blocks, as discussed by [3]. One well-known metric, used in a wide variety of contexts, is the Sum of Absolute Differences (SAD). For blocks of size $w \times w$ (odd w) the SAD value can be calculated, for a given displacement d, by

$$\sum_{\Delta u=-w'\Delta v=-w'}^{\Delta u=w'\Delta v=w'} |I_1(u+\Delta u, v+\Delta v) - I_2(u+\Delta u+d, v+\Delta v)| \tag{1}$$

where $w' = \lfloor w/2 \rfloor$ is the half-width of the block, and I_1 and I_2 specify the pixel intensity for the reference and candidate blocks, respectively. The reference block is centered on (u, v); the candidate block is horizontally displaced by the disparity d.

The essence of SAD is to subtract the intensity value of a pixel in the reference block from the intensity of the corresponding pixel in the candidate block. Summing the absolute values for all pixels of a block yields a measure of the intensity difference of the two blocks: the smaller the value, the more similar the two blocks are considered to be.

Similarity metrics are not restricted to direct mathematical operations, but may involve transformations of the pixel data (see [3, 4]). The metric based on the Census transform, described by [27], belongs to this category. For $w \times w$ blocks, the dissimilarity between a block centered on (u, v) and another displaced horizontally by d pixels is given by

$$D_H(\text{Census}(I_1, u, v), \text{Census}(I_2, u+d, v)), \tag{2}$$

where $D_H(b_1, b_2)$ gives the Hamming distance between bit vectors b_1 and b_2, and

$$\text{Census}(I, u, v) = \bigotimes_{\Delta u=-w'\Delta v=-w'}^{\Delta u=w'\Delta v=w'} \{I(u+\Delta u, v+\Delta v) \geq I(u, v)\} \tag{3}$$

Here, $\bigotimes^{2}$ denotes the concatenation of the bits encoding the result of the test included in brackets.

The idea underlying the Census metric is to take the central pixel of a block and then to classify all other pixels according to their relative brightness. A value of one or zero corresponding to the result of the comparison is assigned to each pixel, producing a code word for each block (the Census transform of the block). The Hamming distance between two code words (i.e., the number of equal bits in corresponding positions of the two words) provides a measure of the similarity between the blocks. This metric is sensitive to the position and relative brightness of the block pixels, but not to their absolute brightness. Therefore, it can be used to complement the information provided by the SAD metric.

Color information may be used to extend the scope of these metrics. The SAD operation can be extended to incorporate the chrominance or the RGB color components of the pixel, as done in [7, 19]. Alternatively, color segmentation can complement the operation of the Census transform, which tends to ignore textureless, usually homogeneously colored areas; this is the approach of [12]. Some of the issues that arise in exploiting color information are discussed by [24] in the context of image enhancement.

Using disparity information, it is possible to segment the image into objects or groups of objects that occupy the same focal plane by clustering pixels with similar disparities. Assigning a focal plane to a given range of disparity values cannot be done straightforwardly without having some kind of knowledge about the scene in advance. It is usually an empirical association based on the scene's characteristics, and there will always be a degree of uncertainty associated. Although the system provides full support to this segmentation, for the current work the goal is only to identify those objects that are closer to the cameras. Hence, there is only the need to distinguish between *foreground* and *background*, meaning that a single threshold may be used.

A problem that all disparity computation methods face is the handling of occluded areas. In fact, the operation of the block matching algorithm is such that a given background pixel in one image may not be visible in the other image because it is occluded by a foreground object. In this case, the block matching algorithm returns an incorrect disparity value based on a spurious match, as exemplified in Fig. 2.

For the specific case of disparity maps used for two-level foreground/background segmentation (as for the current work), a simple consistency check for detection of occluded pixels is possible if the search is repeated with the roles of both images switched. By combining the information of both searches, pixels that appear in only one of the images and for which the disparity information is invalid can be identified (see [6]). Although conceptually simple, it requires twice the computational resources necessary for computing a single disparity map; thus, many implementations avoid this feature.

The reasoning behind the consistency check is to verify, for each pixel assigned to the foreground in one disparity map, whether the other disparity map also assigned it to the foreground. If the condition is true, the pixel has been correctly classified as belonging to the foreground; otherwise, it is considered as belonging to an occluded area and

Fig. 2 The background area confined by the block in the left image (*left figure*) is not found in the right image (*center figure*) due to occlusion by the hand (*dashed arrow*), causing the system to find a match farther along the left-wise search (*solid line*), and thus erroneously assigning that block a high disparity. The resulting thresholded disparity map (*right figure*), where white pixels represent foreground pixels, exhibits this wrong classification (*circled area*)

Fig. 3 The consistency check crosses the information of the two thresholded disparity maps to identify errors due to occlusion. For an area that was erroneously considered as foreground in the left bitmap (*circled area* in *left figure*), the other bitmap is verified at the location indicated by the area's assigned disparity (*circled area* in *center figure*). If a foreground area is not found as well, the initial classification is discarded, and thus a corrected bitmap is produced (*right figure*)

therefore to the background. Figure 3 provides more insight on this.

2.2 Dedicated hardware for disparity map computation

Many hardware architectures for disparity computation are reported in the literature (see Table 2, Sect. 6). This summary focuses on FPGA-based implementations that target real-time performance using block matching with SAD or Census metrics. The work in the field is very heterogeneous, as many aspects of the systems (e.g., hosting platforms, algorithms, internal architecture and additional features like rectification) vary in such a manner that a meaningful comparison is hard to achieve. Nevertheless, some parameters are often used to provide a framework for comparisons, such as frame rate, maximum disparity range, operating frequency, image and block size.

A seminal work in the area is the PARTS system by [26]. It is composed of 16 Xilinx 4025 FPGAs and is capable of computing stereo disparities of up to 24 pixels on 320 × 240 images at 42 fps using the Census metric. The same metric is used by the implementation described in [5], which employs two boards with 6 FPGAs each (Xilinx 4K family) to process 256 × 256 images at 29 fps.

A single-FPGA implementation, described in [2], employs a Virtex XCV800 device to implement a SAD-based disparity computation system. Operating at 66 MHz, this implementation processes 320 × 240 images at 71 fps for a maximum detectable disparity of $d_{\max} = 16$. The block size used is 7 × 7.

The characteristics of more recent, noteworthy systems are summarized in Table 2 (Sect. 6), together with the implementation proposed in this work. Comparing these hardware implementations needs to consider multiple objectives which depend on the intended application area. Nevertheless, the table includes a column with the GDPS score (trillion (Giga) Disparity-Pixel results produced per seconds) as a rough indicator of computational performance. The score has been estimated for all implementations as

$$\text{GDPS} = \text{width} \times \text{height} \times d_{\max} \times \text{frame rate} \times 10^{-9} \quad (4)$$

The implementation discussed in this paper is closest to the one described in [9] since it also uses two metrics (SAD and Census) and performs consistency checks, handling a total of four intermediate disparity maps. An ASIC implementation is reported, which handles 320 × 480 images with $d_{\max} = 47$ and an image data rate of 5 MPixels/s. An ASIC (application-specific integrated circuit) implementation of a similar approach (using the sum of squared differences instead of SAD) is discussed in [16]. It handles 256 × 192 images at more than 50 fps with $d_{\max} = 25$ and 10 × 3 blocks. More recently, the ASIC implementation of [10] employs SAD with left/right

consistency checking to process 320×240 images (11×11 block size and $d_{\max} = 64$). Our FPGA-based implementation processes 640×480 images at 80 fps, implements both SAD and Census metrics and performs consistency checks while achieving $d_{\max} = 135$.

3 Architecture overview

This section gives a general description of our system for location of foreground objects in real-time that includes the expandable architecture for disparity map computation and object location proposed in this paper. The key innovations are to be found in the disparity computation stage, which will be described in more detail in the following section.

The inputs are two video streams and the system produces a VGA output with the visual representation of the disparity map (useful only for debugging and system tuning) and the coordinates of the foreground object. The overall data flow and main hardware modules, based on [21], are presented in Fig. 4, and the four main computation stages are described next.

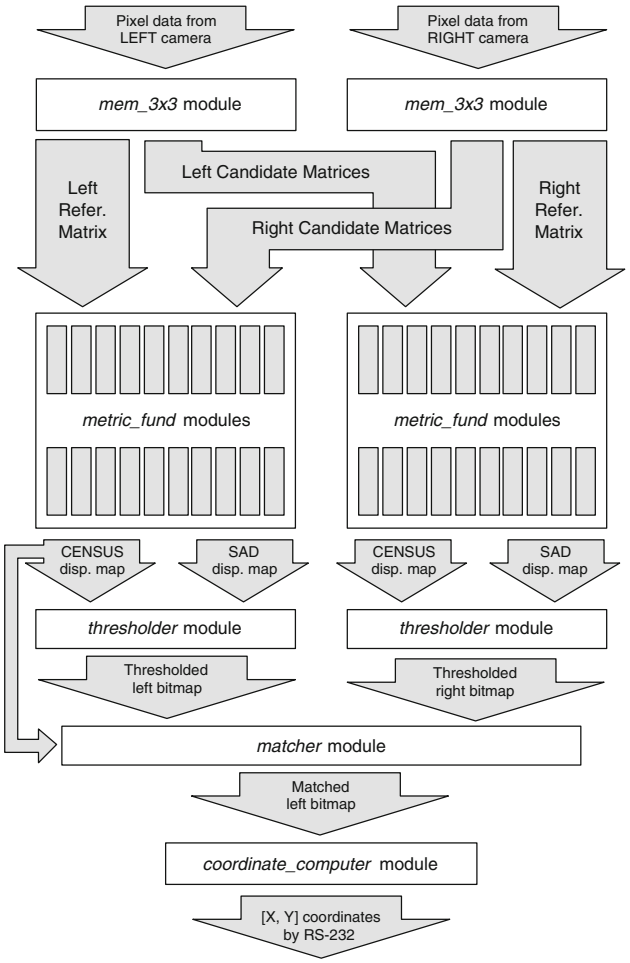

Fig. 4 Diagram depicting the overall dataflow and the interaction between the main hardware modules

Disparity computation: This stage uses the block-match algorithm to compute disparity maps. Both the right image and the left image are used as source for reference blocks, using two different metrics (SAD and Census), thus producing four distinct disparity maps. Two mem 3x3 modules receive data from the cameras and feed them to an array of `metric_fund` modules, which perform the actual disparity calculation.

Thresholding In this stage, the scene is segmented into two focal planes by performing a binarization of the disparity values in each of the four maps , according to a user-defined threshold, at the `thresholder` modules. Moreover, the bitmaps obtained from the SAD and Census metrics are combined by a logical-OR operation; thus only two bitmaps are output: one corresponding to the disparity map of which the left image provided the reference blocks and another in which the reference blocks came from the right image.

Consistency check Spurious areas, that were erroneously considered as foreground due to occlusion, are removed in this stage. The `matcher` module combines the two bitmaps from the previous stage and performs the consistency check discussed in Sect. 2. This stage produces a single bitmap.

Coordinate computation The center-of-gravity of the pixels in the bitmap is computed by module `coordinate_computer` and its coordinates output.

A goal for the current architecture is to be efficiently adaptable to the amount of logic resources available in the target platform and to exploit parallelism as much as possible. In this context, an important architectural decision was to have multiple instances of a fundamental hardware module (`metric_fund`) for calculating block similarity metrics: each module receives a reference block and compares it with a stream of incoming candidate blocks. At the end of a computational cycle, each module outputs the maximum disparity found for a given reference block. Processing the disparity of several pixels simultaneously employs a potentially large number of these modules (which may vary for different designs according to the available resources).

Another relevant architectural option is to use two sets of these modules so that the consistency check discussed earlier can be performed. This allows the two types of disparity maps to be computed simultaneously: one having the right image providing the reference blocks that are searched for in the left image and vice-versa.

A third key decision is to support two different metrics to increase the system's robustness. SAD and Census are fundamentally different in nature, and thus their results are somehow complementary and highlight different visual

aspects, as discussed in Sect. 2.1. In addition, both metrics present a good quality vs. resource consumption trade-off for hardware implementation. In simulation, both metrics showed good results, while only demanding fairly basic arithmetic and boolean operations such as additions, comparisons and XORs. Moreover, the use of color information to improve the metrics' performance was found unnecessary. If applied to the SAD metric computing hardware, it would consume the triple of the resources due to the color components. Alternatively, Census could be complemented by a color analysis to identify homogeneously textured areas, but SAD achieves the same goal with a more straightforward implementation.

For other applications beyond the proof-of-concept implementation described here, the last stage may be replaced by implementations of more sophisticated techniques such as, for instance, connectivity analysis or color-based segmentation.

4 Architecture and operation

This section addresses the four computation stages while focusing on the internal operation and the key innovating features of the disparity computation modules. For the sake of concreteness, the following discussion examines disparity calculations for 8-bit grayscale images of size 640 × 480 at 25 fps, as implemented in our prototype. A general discussion of implementation characteristics for other image sizes and frame rates is presented in Sect. 5.

Regarding the system inputs, the pixel data are provided directly by the cameras as a row-ordered stream of bytes. Camera synchronization is ensured by the interface electronics. Dedicated signals from the cameras determine the start and end of each pixel row.

4.1 Disparity computation

To compute a disparity map for an 640 × 480 pixel image, each one of the 640 reference blocks in a line has to be compared with up to $d_{\max}$ candidate blocks. Our strategy is to handle each reference block independently by assigning it to a dedicated `metric_fund` module, which receives up to $d_{\max}$ candidate blocks and outputs the disparity of the reference block at the end. Candidate blocks are drawn sequentially from the respective search area, shifted by one pixel, one per clock cycle.

Complete parallelization would require a dedicated module for each reference block. In our case, that would be 640 hardware modules, which is a number unfit for FPGA implementation. Instead, the proposed architecture performs the computation in rounds. Only 20 `metric_fund` modules are used in the current implementation: during the first round, 20 reference blocks are processed; in the second round, reference blocks 21 to 40 are handled and so on until all 640 blocks are processed.

Within each round, the proposed architecture is able to share resources between the processing of multiple reference blocks. Note that the spatial relation between two consecutive reference blocks is a mere shift of one pixel. This observation also applies to the respective search areas in the other image. Furthermore, reference blocks and search areas, although extracted from different images, start from the same X-position. Our approach is then to provide one single stream of candidate blocks to all `metric_fund` modules and have the modules start their operation at different times.

An exemplification is now presented, with the help of Fig. 5. First of all, note that the `mem3x3` module (which holds the data received from the cameras) outputs a column of 3 pixels every clock cycle and that three consecutive columns form a block. If we designate a block by the columns it is composed of (1, 2, 3 for the first block) and label the block with R and C for reference and candidate blocks respectively, the search area of $R_{1,\,2,\,3}$ will be $C_{1,\,2,\,3}$, $C_{2,\,3,\,4}$, $C_{3,\,4,\,5}$ and so forth up to $C_{d_{\max}-2,d_{\max}-1,d_{\max}}$. For $R_{2,\,3,\,4}$, the search area is the same, simply shifted by a pixel: $C_{2,\,3,\,4}$, $C_{3,\,4,\,5}$, $C_{4,\,5,\,6}$, up to $C_{d_{\max}-1,d_{\max},d_{\max}+1}$.

With this column-by-column mechanism, a steady flow of candidate blocks is guaranteed. The spatial shift between search areas of consecutive reference blocks is respected by activating each `metric_fund` module precisely during the time interval that the relevant search area is being provided. This is shown in Fig. 5, where grey areas represent the activation intervals of each module. In this small-scale example, $d_{\max}$ is 4.

From the perspective of the `metric_fund` modules, this operation has two stages. In the first one, which takes place on the first cycle of the round, the modules receive a reference block from one of the `mem3x3` modules and store it a dedicated register. In the second stage, starting on the following cycle, a stream of candidate blocks starts arriving from the other `mem3x3` module and the circuitry dedicated to compute the metrics SAD and Census starts operating. For each metric, the stored reference block is compared with the incoming candidate blocks, and a similarity value is output.

This similarity value is then compared with the value stored in a register named best_match, which stores the similarity value of the most similar candidate block found so far. If the similarity value of the current candidate block turns out to be better than the existing value, the best_match register gets updated as well as a second register called disparity. This one stores the distance of the

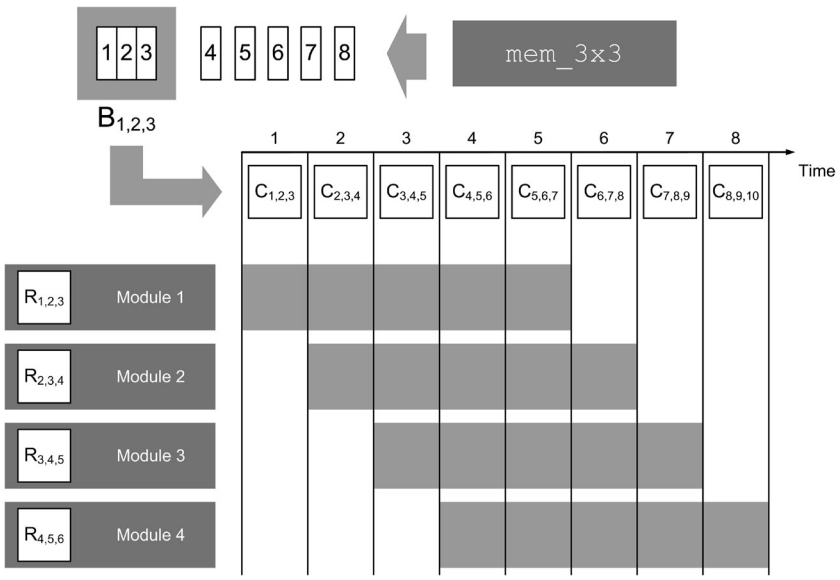

Fig. 5 The column-by-column and sequential activation mechanism. *Blocks* labeled with *C* and *R* are candidate and reference blocks, respectively. *Gray areas* in the diagram correspond to the operation intervals of each module. As consecutive reference blocks, as well as their search areas, are shifted by only one pixel, a common search area is fed to all modules, and each module is only activated when the relevant search area is made available

candidate block to the reference block. In the end, when the stream of candidate blocks is over, the module outputs the content of the disparity registers, one per metric.

The described hardware organization involves some redundancy in the calculations, because the same pixel differences are calculated in three different `metric_fund` modules in consecutive computation cycles. This redundancy might be avoided by providing direct connections between adjacent `metric_fund` modules and implementing special case processing for the first and last two final modules. This approach would incur in additional routing and control complexity, so it is not clear whether final performance would benefit from it. The implemented architecture has less internal dependencies and more straightforward control.

As for the `mem3x3` module, it receives the images from one of the cameras and delivers pixel blocks to the array of `metric_fund` modules upon request. Two major requirements constrain the design of this module. The first is that area-matching algorithms require at least 3×3 pixels blocks to operate. In fact, both width and height of the block are design parameters (w and h), and other values may be defined for different speed, power and resource usage trade-offs. The second is that, for the system to be real-time, the processing of one line has to be done in the same time it takes to receive one.

The first requirement translates to the need of the `mem3x3` module possessing h internal memories, one for each line. The second requirement results in the incoming line having to be stored while the other h are being processed. Hence, an additional memory is used, along with a round-robin mechanism: while h memories provide data for the processing stages, the $(h + 1)$th is being filled with the pixel data that comes from the cameras.

Finally, the flow of data and all operations are managed by the `global_ctrl` module, which is an elaborate finite state-machine in charge of managing data flows and modules operation. It activates and deactivates the `metric_fund` modules at the correct time, through dedicated enable signals, so that each one starts operating as soon as the adequate block is output by the `mem3x3` modules.

The `global_ctrl` module also provides the read and store pointers for the `mem3x3` modules and, as soon as a line has been fully received from the cameras, it enforces the round-robin mechanism, indicating another memory to store the pixels arriving from the cameras and resetting the store pointers. As for the read pointers, there are in fact two sets of those, since two disparity maps are being computed independently.

4.2 Segmentation, consistency check and coordinate computation

The disparity maps of the previous stage encode the disparity obtained for each pixel of the input images, in 8-bit values. There are four disparity maps: for each kind of disparity map (right image as reference, and vice-versa), one is based on the SAD metric, and another on the Census metric.

At this point, the module `thresholder` performs a simple binarization for all four disparity maps: pixels are transformed into a 1 or a 0, according to whether they are above or below a user-defined threshold. Also, the SAD and Census result of each kind of disparity map are merged by an OR operation. If a pixel is a 1 in the SAD or Census bitmap, the pixel in the resulting bitmap is also a 1.

The motivation for using this particular logical operation is twofold. On one side, as discussed in Sect. 2.1, the results of both metrics are somehow complementary, meaning that if an intersection was to be performed instead, relevant information could be lost. Given that the subsequent stage will discard all results that were not coherently considered foregroung pixels in both kinds of disparity maps, it is important for this later stage to receive as much relevant information as possible. Otherwise, the final coordinate computation stage might not receive enough data for computing a meaningful result.

In the end, the thresholding stage outputs two bitmaps: one based on reference blocks extracted from the right image (for convenience, it will be named the "right" bitmap), and other in which reference blocks were extracted from the left image (the "left" bitmap). The `matcher` module takes on these two bitmaps and verifies that the foreground pixels in the left bitmap were correctly identified as such. It uses the left and right bitmaps produced by the threshold stage, together with the disparity value for each pixel of the left bitmap computed ealier. Either SAD or Census values can be used; our experience agrees with [11] in that using the value from the Census disparity map leads to better performance, and, therefore, our prototype also uses it.

The bits from the right bitmap are stored in a shift register, in which all positions are accessible for reading. As each pixel from the left bitmap (and its associated disparity) arrives to the `matcher` module, one per clock cycle, this memory is accessed by a multiplexer that fetches the right bitmap's bit indexed by the disparity value of the left bitmap's bit. The two bits are then ANDed together: if both are 1, the pixel of the left bitmap is considered foreground. The reasoning behind this logical operation is explained in Sect. 2.1 (or, more succintly, in Fig. 3). Figure 6 presents the module's internal architecture.

As the disparity value of each bit can only address $d_{\max}$ positions (in our proof of implementation, $d_{\max} = 135$ pixels), the shift-register only needs to store this number of bits from the right bitmap. For example, if the left bitmap's pixel under analysis is the 430th pixel from the left, the bits from the right bitmap available in the memory are those from position $430 - 135 = 295$ up to position 430.

The need for storing a full line of bits and fetching from it as needed is avoided by another architectural design option. The right bitmap is produced from left to right: reference blocks, as well as candidate blocks, are drawn right-wise and one would expect the production of the left bitmap to mirror this direction. However, as in [11], the production of the left bitmap is slightly modified to accompany the production of the right bitmap, thus obtaining the temporal and spatial closeness that eliminates the need to store a full line of results.

Finally, the coordinates of the center of gravity of the foreground objects are calculated by the module `coordinate_computer`. For that purpose, the number of foreground pixels of each row and column is computed. These values are multiplied by the index of the corresponding row or column. The weighted sums of all rows and columns are stored in two registers. Dividing the content of these two registers by the total number of foreground pixels in the bitmap yields, respectively, the X and the Y coordinates of the center of gravity.

The major architecture restriction on this stage is the fact that results arrive to this module in the same manner the cameras deliver the images: from left to right, from top to bottom. This means that, while the total number of foreground pixels in a line can be computed at once, temporary totals for all rows must be kept, requiring an extra memory.

5 Design parameters and performance

This section analyses the various trade-offs that can be obtained by adjusting the system parameters and discusses the dependencies between design parameters and performance of the system.

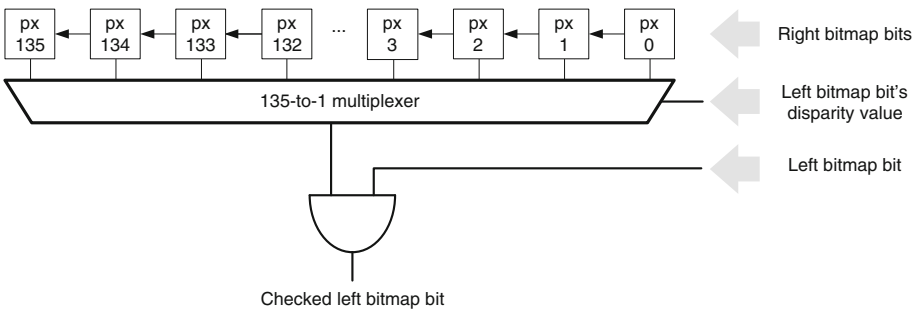

Fig. 6 Hardware structure of the `matcher` module. A linear memory as long as the maximum disparity range (135 bits in this case) stores a window of bits from the right bitmap. The memory is then accessed according to the value given by the disparity value for the left bitmap's pixel under analysis. The bit fetched from the memory and the left bitmap's bit are then used to confirm the classification of the later

5.1 Application requirements and design parameters

The main requirement posed by the application is the minimum distance R between the cameras and the objects that must be detected. To meet such application requirement, the designer must determine a single architectural parameter: N, the number of `metric_fund` modules.

The system can be configured by a set of eight interrelated parameters, which can be classified into two types:

- camera parameters:
  - distance between cameras d_C;
  - focal distance of the cameras f;
  - frame width in pixels W;
  - frame width in meters W_e ("effective width").
- system parameters:
  - maximum disparity $d_{\max}$;
  - operating frequency f_{op};
  - data rate of incoming pixels f_{in};
  - width of the block w;
  - number of `metric_fund` modules N.

The maximum disparity parameter $d_{\max}$ plays a key role because it establishes a relationship between system parameters (which determine the system parameter N), and camera parameters (that must be compatible with the desired R). Ignoring any internal latency, system parameter $d_{\max}$ is given by

$$d_{\max} = \frac{f_{\text{op}}}{f_{\text{in}}} \times N \tag{5}$$

To achieve a target $d_{\max}$, only the relative values of parameters f_{in} and f_{op} matter. If $f_{\text{op}} = f_{\text{in}}$, then the number of `metric_fund` modules needed will be the same as the desired maximum disparity $d_{\max}$, because each `metric_fund` module handles only one reference block at a time.

Our implementation has $f_{\text{in}} = 12.5$ MHz and $f_{\text{op}} = 100$ MHz. Their ratio is $100/12.5 = 8$, so there are eight clock cycles available for processing each block. Since the implementation has $N = 20$, it can handle 20 reference blocks simultaneously and thus achieve $d_{\max} = 8 \times 20 = 160$ in the absence of additional internal delays.

The actual value of $d_{\max}$ is smaller due to two sources of latency. First, N clock cycles are necessary to sequentially load the `metric_fund` modules with the reference blocks at the beginning of each round. The second source is memory latency: the $w \times w$ blocks are output by the `mem3x3` modules in columns, one per cycle, and hence it takes $w + 1$ clock cycles for each module to get all the columns of its reference block and of the first candidate block. Furthermore, an additional cycle must be subtracted to account for the comparison of the candidate block that has a relative displacement of 0 with respect to the reference block. The resulting expression for $d_{\max}$ is

$$\begin{aligned} d_{\max} &= N \times \frac{f_{\text{op}}}{f_{\text{in}}} - N - (w+1) - 1 \\ &= N \times \left(\frac{f_{\text{op}}}{f_{\text{in}}} - 1 \right) - w - 2 \end{aligned} \tag{6}$$

Hence, for $N = 20$ and $w = 3$, we have $d_{\max} = 160 - 20 - 5 = 135$. Increasing the block width w results only in a small latency penalty equal to the number of additional columns of the block. Thus, for a 5-pixel wide block, the value would be $d_{\max} = 160 - 20 - 7 = 133$.

Using the camera parameters, $d_{\max}$ can be linked to the main application requirement R (minimum distance between cameras and objects in the image foreground). Following the discussion by [22], the disparity d of an object at a given distance R to the cameras is estimated by

$$d = \frac{d_C \times f}{R} \times \frac{W}{W_e} \tag{7}$$

where the proportionality factor W/W_e is used to obtain the value in pixels.

For our prototype frame width is $W = 640$, inter-camera distance $d_C = 7.5$ cm, focal distance $f = 6$ mm, and the effective width of the image is $W_e = 4.86$ mm, as found in the cameras datasheet [21]. Taking $d = d_{\max} = 135$ results in $R = 43.9$ cm, which is quite adequate for our target application.

Combining Eqs. 5 and 7 gives a single relationship for recomputing the necessary number of `metric_fund` modules for a given R:

$$N \geq \left(\frac{d_C \times f}{R} \times \frac{W}{W_e} + w + 2 \right) \times \frac{f_{\text{in}}}{f_{\text{op}} - f_{\text{in}}} \tag{8}$$

Only one recommendation applies to the value obtained from this formula: it is advisable to have N be a divisor of the frame width W in order to ensure optimal usage of the hardware resources.

Therefore, the work of exploring the existing large solution space, in which FPGA resource usage can be traded off with camera parameters and vice-versa, becomes easier, and a designer can tailor the architecture for specific applications while achieving different speed and power trade-offs.

5.2 Calculation effort

This section estimates the amount of computational effort of the proposed architecture using as indicator the number of arithmetic operations (addition and subtractions)

performed by the disparity calculation modules. Take H to be the number of pixels in a image row and $w' = \lfloor w/2 \rfloor$ to be the distance between a block's central point and the block's border. Then, the number of pixels for which a disparity value is calculated is defined by $H' \times W'$, where

$$H' = H - w' \tag{9}$$

$$W' = W - w' \tag{10}$$

Consider the processing of one image line. For $W' - d_{\max} + 1$ pixels, a full search of $d_{\max}$ positions in the other image is made (for each position, a block comparison is made). For the remaining pixels, the distance to the image borders is less than $d_{\max}$, so fewer block comparisons will be done. The number b of clock comparisons per line is

$$b = (W' - d_{\max} + 1) \times d_{\max} + (d_{\max} - 1) + \cdots + 1 \tag{11}$$

$$= (W' - d_{\max} + 1) \times d_{\max} + \frac{1}{2} \times d_{\max} \times (d_{\max} - 1) \tag{12}$$

Since the process is applied twice, as both images serve as source for reference blocks, the total number of block comparisons per image pair is

$$B = H' \times 2 \times b \tag{13}$$

For the proof-of-concept implementation, we have $w' = 1$, $H' = 638$, $V' = 478$ and $d_{\max} = 135$, resulting in $B = 73{,}976{,}760 \approx 7.40 \times 10^7$ block comparisons per frame. Since the supported frame rate is $f_{\text{fr}} = 25$, the system executes 1.85×10^9 block comparisons per second.

The `metric_fund` modules implement the block comparisons. The calculation of the SAD metric involves $w \times w$ subtractions and the additions of their results, which is performed by a tree of a two-operand adders. So, the calculation of the SAD metric requires the following number of arithmetic operations:

$$n_{\text{sad}} = w \times w + a \tag{14}$$

For $w = 3$, the tree has $a = 8$ adders, resulting in $n_{\text{sad}} = 17$ operations.

The calculation of the Census transform involves $w^2 - 1$ subtractions. Ignoring for simplicity the logic operations required for Hamming distance calculation, a lower bound for the total number of arithmetic operations per block comparison n_b is given by

$$n_b = n_{\text{sad}} + w^2 - 1 = 2w^2 + a - 1 \tag{15}$$

For the proof-of-concept implementation, this results in $n_b = 25$. Overall, the implementation executes 4.62×10^{10} arithmetic operations per second for disparity map computation.

6 Implementation and evaluation

In our proof-of-concept implementation, the disparity computation is embedded in a hand position detection system. The inputs come from two [21] digital cameras placed side-by-side 7.5 cm apart. Each image has 640×480 8-bit pixels delivered at 12.5 MHz. The system produces a VGA output with the visual representation of the disparity map (which is useful for debugging and system tuning), and sends the coordinates of the foreground object over a serial port to a desktop computer. The processing board is a Digilent XUP board with a Virtex-II Pro FPGA (xc2vp30 device).

The complete system is described in Verilog HDL. It has been synthesized for the evaluation platform just described as well as for other FPGA families using the ISE tools (version 10.1 for Virtex-II Pro devices, version 13.2 for the other devices). Table 1 summarizes the complete resource usage (including the modules required for interfacing with cameras and VGA monitor) and the main characteristics of the each implementation. The Virtex-II Pro version was successfully deployed on the target platform.

Versions with $N = 40$ `metric_fund` modules were synthesized for all device families. Only results for the Virtex-4 device implementations are included in Table 1, since the versions for both values of N target the same device model. The other device families exhibit similar patterns of resource usage, which required devices with a larger number of slices to be targeted. Based on post-place-and-route timing information, the implementation with $N = 40$ for the Virtex-5 device is estimated to run at 175 MHz, for a frame rate of 140 fps.

All the target FPGA families have efficient block RAMs (BRAM), which are used in this implementation to store each image line. As described before, four lines of each image must be stored at all times. Since each line/memory may have two simultaneous accesses, the mapping tools replicate those memories, thus yielding a total of 16 memories. Another BRAM is used in the coordinate computation stage to store the temporary totals per row.

The number of flip-flops used can be partially explained by the 2×20 `metric_fund` modules. Counting only the registers needed to hold the reference window and the nine absolute differences of the SAD computation, the number of flip-flops is $40 \times 9 \times 8 \times 2 = 5{,}760$ (since each register is 8 bit wide, and disparity maps are computed in two directions), which accounts for approximately half the total of used flip-flops.

Total power consumption (Table 1) depends strongly on the technology of each device family. As an example, the Virtex-5 implementation for $N = 20$ has a lower power consumption than the corresponding Virtex-4

Table 1 Resource usage and characteristics of system implementations on various FPGA families

	Virtex-II Pro	Virtex-4		Virtex-5
	$N = 20$	$N = 20$	$N = 40$	$N = 20$
Resource usage				
Slices	9,565 (69%)	12,881 (48%)	22,514 (84%)	5,082 (70%)
LUTs	16,847 (39%)	18,461 (34%)	34,090 (64%)	12,144 (42%)
Flip-flops	10,936 (61%)	11,682 (21%)	24,546 (46%)	12,144 (42%)
BRAM blocks (18 Kb)	18 (13%)	18 (11%)	19 (11%)	18 (38%)
Characteristics				
Frequency (MHz)	100	133	133	200
Power consumption (W)	0.92	1.25	2.21	1.04
Frame rate (fps)	40	53	106	80
Maximum disparity (pixels)	135	135	135	135
GDPS	1.66	2.20	4.40	3.32

The target devices are xc2vp30-7 (Virtex-II Pro), xc4vlx60-12 (Virtex-4), xc5vlx50-3 (Virtex-5). Frequency and total (static + dynamic) power consumption estimates obtained from post place-and-route data by Xilinx tools. Slices have two 4-input LUTS and two flip-flops (Virtex-II Pro, Virtex-4) or four 6-input LUTs and four flip-flops (Virtex-5)

Table 2 Overview of FPGA-based systems for dense disparity map computation (present work characteristics are for implementations on Virtex-5 devices)

Reference	W × H	$d_{\max}$	FPS	Metric	B'lock	Check	GDPS
[14]	320 × 240	64	120	SAD	7 × 7	N	0.59
[23]	1024 × 1024	32	47	SAD	16 × 16	N	1.58
[17]	640 × 480	64	30	SAD	32 × 32	N	0.59
[9]	640 × 480	47	32	SAD/Census	3 × 5	Y	0.46
[20]	320 × 240	20	150	Census	7 × 7	N	0.23
[1]	450 × 375	100	599	SAD	9 × 9	N	7.4
[8]	640 × 480	80	275	SAD	7 × 7	N	6.76
[13]	640 × 480	64	130	Census	7 × 7	N	2.56
[15]	640 × 480	64	230	Census	11 × 11	Y	4.52
Present work (Virtex-5)							
$N = 20$	640 × 480	135	80	SAD/Census	3 × 3	Y	3.32
$N = 40$	640 × 480	135	140	SAD/Census	3 × 3	Y	5.91

implementation despite the estimated operating frequency being significantly higher. For Virtex-4 FPGAs, doubling the number of `metric_fund` modules almost doubles the power consumption.

Table 2 summarizes the characteristics of other recent FPGA-based implementations of disparity calculation. The "check" column indicates whether the system performs a left/right consistency check involving the calculation of two disparity maps. Note that the GDPS score does not reflect the block size, the effort due to the calculation of intermediate disparity maps (for consistency checks) or the quality of the disparity map.

The implementation discussed here has a maximum disparity $d_{\max} = 135$, using 3×3 blocks, and processes 640×480 images, at a maximum frame rate of 80 fps when running with a 200 MHz clock ($N = 20$), or 140 fps when running with a 175 MHz ($N = 40$).

Compared with the other systems of Table 2, the proposed system features a fairly large disparity and a relatively small maximum window size. The frame rate is adequate for most image processing tasks.

Only two systems of Table 2 have a higher GDPS than the implementation evaluated in this work (with 40 `metric_fund` modules). Both [1] and [8] achieve very high frame rates, but do not perform consistency checks. The work of [15] achieves a lower GDPS score due to the limited disparity range, but features a higher frame rate and performs consistency check (using larger blocks). All of these systems support only one metric with large blocks, while our implementation supports two metrics with consistency check and smaller blocks, together with the highest value of $d_{\max}$, a key feature for our intended application. The only other system that supports two metrics and performs consistency check is FingerMouse

[9, 11], but it achieves a lower frame rate and a smaller maximum disparity than the present architecture.

In terms of architecture, there is one major alternative to our system: whereas we compute the disparity of several pixels simultaneously, other systems compute a single pixel's disparity at a time. For instance, in the case of [11], each pixel is assigned 16 clock cycles for computation; during each cycle, the reference block is compared with three candidate blocks in parallel. The maximum disparity is thus $3 \times 16 - 1 = 47$ pixels. This strategy requires a large portion of the search area to be available in a very short time; in our system, however, the memories need not provide more than one column of the search area per clock cycle. By parallelizing the computation of the disparity of several pixels over an array of hardware modules, our architecture can take advantage of the spatial closeness of the pixels to feed the same data stream to all the modules, thus spreading the processing over a larger period of time and hence attenuating the requirements for data availability, with just a small cost in the maximum disparity range.

7 Conclusion

We have presented the implementation of a system that receives a pair of stereoscopic images and computes disparity maps, from which it can isolate the closest objects to the cameras in real-time. The goal is to isolate the user's hand from the background and send its coordinates to a computer, so it can support new forms of human-computer interface. The system is completely described in structural Verilog HDL. An implementation sized to handle 640×480 images and achieve a maximum disparity of 135 using 3×3 windows was evaluated for different Xilinx FPGA families. A hardware prototype was built using a Virtex-II Pro FPGA, achieving 40 fps (with a 100 MHz main clock). Synthesis results show that a Virtex-5 implementation of the same architecture achieves a maximum frame rate of 140 fps with a 175 MHz main clock.

The heart of the system is an expandable hardware architecture for the computation of disparity maps. It uses an array of modules that work in parallel, each comparing a block from one of the images with a stream of blocks from the other image. Our architecture implements a simplified dataflow in which the stream of candidate blocks offered to each of the modules is the same. The presented methodology exhibits significant trade-off and scalability capabilities and, in particular, it allows a high disparity range to be detected, an important characteristic for the application addressed in the prototype implementation.

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

References

1. Ambrosch, K., Humenberger, M., Kubinger, W., Steininger, A.: SAD-based stereo matching using FPGAs. In: Kisacanin, B., Nhattacharya, A., Chai, S. (eds) Embedded Computer Vision, chapter 6, pp. 121–138, Springer, Verlag (2008)
2. Arias-Estrada, M., Xicotencatl, J.M.: Multiple stereo matching using an extended architecture. In: Brebner, G., Woods, R. (eds) Field-Programmable Logic and Applications, Springer, Berlin, vol. 2147, pp. 203–212 (2001)
3. Banks, J., Bennamoun, M., Corke, P.: Non-parametric techniques for fast and robust stereo matching. In: Proceedings of IEEE Region 10th Annual Conference Speech Image Technologies for Computing and Telecommunications, vol. 1, pp. 365–368 (1997)
4. Bergmann, N.W., Porter, R.B.: A generic implementation framework for FPGA based stereo matching. In: Proceedings of IEEE Region 10th Annual Conference Speech Image Technologies for Computing and Telecommunications, vol. 2, pp. 461–464 (1997)
5. Corke, P.I., Dunn, P.A.: Frame-rate stereopsis using non-parametric transforms and programmable logic. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 3, pp. 1928–1933 (1999)
6. Fua, P.: Combining stereo and monocular information to compute dense depth maps that preserve depth discontinuities. In: Proceedings of 12th International Joint Conference on Artificial Intelligence, pp. 1292–1298 (1991)
7. Georgoulas, C., Andreadis, I.: A real-time fuzzy hardware structure for disparity map computation. J Real-Time Image Process **6**(4), 257–273 (2011). doi:10.1007/s11554-010-0157-6
8. Georgoulas, C., Kotoulas, L., Sirakoulis, G.C., Andreadis, I., Gasteratos, A.: Real-time disparity map computation module. Microprocess Microsyst **32**(3), 159–170 (2008)
9. de la Hamette, P., Tröster, G.: Architecture and applications of the FingerMouse: a smart stereo camera for wearable computing HCI. Pers Ubiquit Comput **12**(2), 97–110 (2008)
10. Han, S., Woo, S., Jeong, M., You, B.: Improved-quality real-time stereo vision processor. In: Proceedings of 22nd International Conference on VLSI Design, IEEE, pp. 287–292 (2009)
11. Hebb, J., Koch, T., Kuonen, S.: VLSI Implementation of the FingerMouse algorithm. Master's thesis, Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology, Zurich (2005)
12. Humenberger, M., Engelke, T., Kubinger, W.: A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 77–84, (2010). doi:10.1109/CVPRW.2010.5543769
13. Ibarra-Manzano, M.A., Almanza-Ojeda, D.L., Devy, M., Boizard, J.L., Fourniols, J.Y.: Stereo vision algorithm implementation in FPGA using Census transform for effective resource optimization. In: 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, IEEE, pp. 799–805 (2009)
14. Jia, Y., Zhang, X., Li, M., An, L.: A miniature stereo vision machine (MSVM-III) for dense disparity mapping. Proceedings of 17th International Conference Pattern Recognition (2004)
15. Jin, S., Cho, J., Pham, X.D., Lee, K.M., Park, S., Kim, M., Jeon, J.W.: FPGA design and implementation of a real-time stereo vision system. IEEE Trans Circuits Syst Video Technol 20(1), 15–26 (2010)

16. Kuhn, M., Moser, S., Isler, O., Gurkaynak, F., Burg, A., Felber, N., Kaeslin, H., Fichtner, W.: Efficient ASIC implementation of a real-time depth mapping stereo vision system. In: 2003 IEEE International Symposium on Micro-NanoMechatronics and Human Science, vol. 3, pp.1478–1481 (2003)
17. Lee, S., Yi, J., Kim, J.: Real-time stereo vision on a reconfigurable system. In: Hämäläinen, T., Pimentel, A., Takala, J., Vassiliadis, S. (eds) Embedded computer systems: architectures, modeling, and simulation, Lecture Notes in Computer Science, vol. 3553, pp. 225–236, Springer, Berlin (2005)
18. Marsi, S., Saponara, S.: Integrated video motion estimator with retinex-like pre-processing for robust motion analysis in automotive scenarios: algorithmic and real-time architecture design. J Real-Time Image Process **5**(4), 275–289 (2010). doi:10.1007/s11554-009-0148-7
19. Muhlmann, K., Maier, D., Hesser, R., Manner, R.: Calculating dense disparity maps from color stereo images, an efficient implementation. In: Proceedings of IEEE Workshop Stereo and Multi-Baseline Vision (SMBV 2001), pp. 30–36 (2001). doi:10.1109/SMBV.2001.988760
20. Murphy, C., Lindquist, D., Rynning, A., Cecil, T., Leavitt, S., Chang, M.: Low-cost stereo vision on an FPGA. Proceedings of 15th Annual IEEE Symposium Field-Programmable Custom Computing Machines, pp. 333–334 (2007)
21. OmniVision OV7620 Single-chip CMOS VGA Color Digital Camera / OV7120 Single-chip CMOS VGA B&W Digital Camera Datasheet (2000)
22. Overington, I.: Computer vision: a unified, biologically-inspired approach. Elsevier, Amsterdam (1992)
23. Roh, C., Ha, T., Kim, S., Kim, J.: Symmetrical dense disparity estimation: algorithms and FPGAs implementation. In: Proceedings of 2004 IEEE International Symposium Consumer Electronics, pp. 452–456, (2004). doi:10.1109/ISCE.2004.1375987
24. Saponara, S., Fanucci, L., Marsi, S., Ramponi, G.: Algorithmic and architectural design for real-time and power-efficient retinex image/video processing. J Real-Time Image Process **1**, 267–283 (2007). doi:10.1007/s11554-007-0027-z
25. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Intl J Comput Vis **47**(1), 7–42 (2001)
26. Woodfill, J., Von Herzen, B.: Real-time stereo vision on the PARTS reconfigurable computer. In: Proceedings of 5th Annual IEEE Symposium FPGAs Custom Computing Machines, pp. 201–210 (1997)
27. Zabih, R., Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In: Proceedings of Third European Conference on Computer Vision, Springer Verlag, vol. 2, pp. 151–158 (1994)

Author Biographies

Pedro Miguel Santos got his Bachelor's and Master's Degree in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto, in 2007 and 2009, respectively. Immediately after, he enrolled the PhD Program in the same field and institution and was awarded a Doctoral Scholarship by the Portuguese Foundation for Science and Technology. He is currently working at Instituto de Telecomunicações, and his research interests are computer architecture and wireless sensor networks.

João Canas Ferreira received the PhD degree in electrical and computer engineering from the University of Porto (Portugal) in 2001. He is currently an assistant professor with the Faculty of Engineering, University of Porto. His current research interests include dynamically reconfigurable systems, application-specific digital system architectures, and ECAD tools and algorithms.

José Silva Matos received his Licenciatura in Electrical Engineering, in 1973, from FEUP (the Faculty of Engineering of the University of Porto, Portugal). He later obtained degrees of Master of Science (1980), and Philosophy Doctor (1983) in Electrical Engineering, from Syracuse University, Syracuse, NY, USA. His professional experience includes positions as an Engineer with the Electronics Laboratory of General Electrical Company, in Syracuse, NY (1982–1983), as a Visiting Professor and Research Associate with the CASE Center of Syracuse University (1986/87), and as head of the CAD and Microelectronics Group at INESC Porto, Portugal. Currently he is a Full Professor at the University of Porto, where he chaired the Department of Electrical and Computer Engineering of FEUP between 2001 and 2010. His teaching and research interests are centered on Synthesis for Reconfigurable Systems and on Microelectronics Design and Test. He serves in the Technical and Program Committees of international conferences like EUROMICRO Digital System Design (DSD), Design of Circuits and Integrated Systems (DCIS), and International Symposium on Quality Electronic Design (ISQED). Prof. José Matos has represented the Portuguese government in several Committees. Currently, he is a member of the Management Committee of the Information and Communications Technologies Program (ICT), of the 7th Framework Program on Research and Development of the European Union.