Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 386059, 12 pages
doi:10.1155/2008/386059

*Research Article*

# Flexible Hardware-Based Stereo Matching

**Karina Ambrosch,[1] Wilfried Kubinger,[1] Martin Humenberger,[1] and Andreas Steininger[2]**

[1] *Austrian Research Centers GmbH-ARC, 1220 Vienna, Austria*
[2] *Institute of Computer Engineering, Vienna University of Technology, 1040 Vienna, Austria*

Correspondence should be addressed to Karina Ambrosch, karina.ambrosch@psi.ch

Received 28 February 2008; Revised 5 June 2008; Accepted 20 November 2008

Recommended by Dragomir Milojevic

To enable adaptive stereo vision for hardware-based embedded stereo vision systems, we propose a novel technique for implementing a flexible block size, disparity range, and frame rate. By reusing existing resources of a static architecture, rather than dynamic reconfiguration, our technique is compatible with application specific integrated circuit (ASIC) as well as field programmable gate array (FPGA) implementations. We present the corresponding block diagrams and their implementation in our hardware-based stereo matching architecture. Furthermore, we show the impact of flexible stereo matching on the generated disparity maps for the sum of absolute differences (SADs), rank, and census transform algorithms. Finally, we discuss the resource usage and achievable performance when synthesized for an Altera Stratix II FPGA.

## 1. INTRODUCTION

The deployment of autonomous systems mainly depends on their capability to navigate in unfamiliar environments as well as their object detection and classification abilities. The main requirement for the implementation of these two features is the availability of reliable three-dimensional information.

A technique able to cope with these requirements is stereo vision. Stereo vision uses two cameras side-by-side and extracts the displacement of the objects caused by the cameras' different viewpoints. The displacement, called disparity, is directly correlated to the distance of the objects, which can be calculated easily using triangulation. This way a three-dimensional depth map can be computed. Unlike other techniques, the depth map computed by stereo vision is typically centered on one of the cameras. Thus, object classification tasks can enhance their recognition performance by directly matching the depth map against the intensity image produced by the camera.

Another feature of stereo vision is that in contrast to other techniques, such as laser range finders, supersonic, or radar sensors, stereo vision comes along without active systems or movable mechanical parts.

Applications of stereo vision are not limited to common robot navigation [1] or distant measures in laproscopic surgeries [2]. They also include the use on autonomous vehicles like at the DARPA grand challenge [3] or the deployment in the NASA/JPL Mars Exploration Rover mission [4].

The core element of a stereo vision system, the stereo matching algorithm, has a high computational complexity. Fortunately, area-based algorithms, and box-filtering algorithms in particular, proved to be very suitable for solutions using hardware-based parallel processing [5], enabling the implementation of real-time embedded stereo vision systems with high frame rates.

Recent works reveal great advances in this field of research. Woodfill et al. proposed the DeepSea application specific integrated circuit (ASIC) [3], enabling the processing of $512 \times 480$ images at a disparity range of 52 pixels, a block size of $7 \times 7$, and a high frame rate of 200 fps. On the other hand, Lee et al. [6] presented a system processing $640 \times 480$ images with a disparity range of 64 pixels at 30 fps using field programmable gate arrays (FPGAs), having a considerably large block size of $32 \times 32$ pixels. Then again, the FPGA-based stereo vision system proposed by Perri et al. [7] operates on $512 \times 512$ images, having a block size of $3 \times 3$, a frame rate of 25.6 fps, and a large disparity range of 255 pixels.

Given the aforementioned applications, embedded stereo vision systems are likely to operate in an environment that is subject to repeated changes. For instance, in the application of robot navigation for domestic robots, the environment of each room can be very different, demanding different features for the algorithms. A tidy kitchen with unique colored furniture usually has very sparse texture, and requires a highly increased block size for the algorithm. Otherwise, a typical teenager's room can be pretty untidy, including a high number of small objects spread over the room, which could be unrecognizably blurred when using the same block size as for the kitchen. Thus, as soon as the robot enters this room, a smaller block size would be advisable. As the robot has to interact with the furniture or handle single objects, it also has to recognise them at close distances. In these situations the stereo system should be able to recognise close objects rather than operate at a high frame rate. On the other hand, while the robot drives along the hallway, the application of the stereo system could often be reduced to collision avoidance. Here, neither the detection of close objects nor the large block size would be required, rather a high frame rate for covering distances in a speedy way. In any case, even if each of the aforementioned systems implements an algorithm with outstanding features, none of them is able to adapt its features to the situation. For instance, if the DeepSea's high frame of 200 fps is not required for the moment, there is no possibility for taking advantage of a reduced frame rate, like an increase in disparity range or block size.

This is caused by the fact that in contrast to software-based implementations, hardware-based solutions cannot easily change their behavior without deactivating parts of their resources. Jacobi et al. [8] proposed a method to dynamically reconfigure an FPGA for different block sizes using a sum of absolute differences (SADs) algorithm. Dynamically, reconfiguring an FPGA for different behavior is surely a very elegant method, but it drastically increases the complexity of the FPGA design and can only be performed on specific FPGAs. The time needed for the reconfiguration is related to the size of the reconfigured chip area. This time span must not be disregarded for the design of a real-time system that has to meet a hard deadline.

Therefore, we propose a novel technique to adapt block size, disparity range, and frame rate for hardware implementations of area-based stereo matching algorithms. This technique is suitable for FPGAs as well as ASICs, that is, it comes along without dynamic reconfiguration, enabling its implementation on real-time systems with short deadlines.

In Section 2 we give an overview of stereo matching algorithms. Section 3 presents our novel flexible hardware-based stereo matching technique and describes the implementation for adapting the block size as well as the disparity range. Then, we describe how a flexible frame rate follows. The experimental evaluation of our technique is given in Section 4. Here, we show a detailed presentation of our hardware-based stereo matching architecture implementing the proposed technique. Furthermore, we present the experimental results along with a discussion on them. Our final conclusions are revealed in Section 5.

## 2. STEREO MATCHING ALGORITHMS

### 2.1. Overview

Stereo matching algorithms are used to solve the correspondence problem of a pair of camera images. They extract the displacement of all objects and generate a disparity map. Therefore, stereo matching algorithms search for the correspondences in the stereo images using feature-, phase-, or area-based matching to calculate the disparities in the images. Feature-based matching searches for characteristics in the images, like edges or curves [9], and calculates the best matches according to their similarities. Phase-based algorithms band pass filter the images and extract their phase [10]. Area-based algorithms take blocks of pixels from both images and calculate their matching costs. This can be done in parallel for all analyzed pixels. When using a constant block size over the whole image, called box filtering [11], these algorithms are especially amenable to parallel and hardware-based solutions.

Color information can be used to improve the matching performance significantly [12]. However, the required hardware resources for processing color images with area-based algorithms on embedded real-time systems are still very high. To keep the focus on algorithms that are suitable for state-of-the-art hardware, we use gray-scale images only.

Based on the stereo taxonomy by Scharstein and Szeliski [11], most area-based algorithms start the matching procedure by applying a neighborhood transformation function $T$ on the primary $I_1$ and secondary $I_2$ stereo image, with a defined block size $s_t$, to achieve better results. When using $x$ and $y$ for the neighborhood region's center pixel, the transformed images' pixel values $t_{1/2_{x,y}}$ are defined as

$$t_{1/2_{x,y}} = T(I_{1/2}, x, y, s_t). \tag{1}$$

Afterwards, the matching costs $c_{x,y,d}$ of the transformed images are calculated for each pixel $x, y$ using the matching costs function $C$. This is performed for all disparity levels $d$ that are within the disparity range

$$c_{x,y,d} = C(t_{1_{x,y}}, t_{2_{x+d,y}}). \tag{2}$$

Then, the matching costs are aggregated over a defined block size $s_a$ for each disparity level. The aggregated matching costs $a_{x,y,d}$ are defined as

$$a_{x,y,d} = \sum_n \sum_m c_{x+m,y+n,d}, \tag{3}$$

where

$$n, m \in \left[ -\frac{s_a - 1}{2}, \frac{s_a - 1}{2} \right]. \tag{4}$$

The matching procedure's total block size $s_b$ is the sum of the transformation function and the aggregation's block sizes $s_t$ and $s_a$. Finally, the most accurate matching costs have to be searched for. Their position defines the resulting disparity map's pixel value $d_{\text{map}_{x,y}}$. In our work, this search is always defined as the search for the absolute minimum matching costs value, also called the winner takes all (WTA) algorithm, which has a low complexity when implemented as a very large scale integration (VLSI) circuit [13]. Here, $d_{\max}$ is the maximum value of the disparity range and $d_{\min}$ is the start value for the search, being 0 if the image background is searched for as well

$$d_{\text{map}_{x,y}} = n \mid a_{x,y,n} = \min(a_{x,y,d_{\min}}, a_{x,y,d_{\min}+1}, \dots, a_{x,y,d_{\max}}). \tag{5}$$

For easing our descriptions we will assume $d_{\min} = 0$ in the following, without loss of generality.

When using area-based matching in poorly textured environments, the quality of the disparity map depends mainly on the total block size $s_b$. Otherwise, a large block size leads to a deformation of the image object edges [14] and high computational costs. Thus, the quality of the disparity maps benefits from algorithms that are able to adapt the block size to the texture coarseness in the images, for example, by increasing the transformation block size $s_t$ or the aggregation's $s_a$.

For the description and the analysis of our flexible stereo matching technique, we focus on three stereo matching algorithms, that are suitable for hardware-based implementations. These are the SAD, the rank transform, and the census transform. For these algorithms we examined the achieved disparity map quality as well as the resource usage, when implementing the proposed technique using our hardware-based stereo matching architecture. Other common algorithms, like the sum of squared differences (SSDs) or the normalized cross correlation (NCC) are not analyzed, since they are well known for having too high resource usage when implemented on hardware [5].

### 2.2. *Sum of absolute differences*

SAD [15] is a simple and popular algorithm for FPGA- or ASIC-based stereo matching. One of the early implementations is [16], while there exist various recent implementations as well [8, 17–20].

SAD is an area-based algorithm that gets along without a transformation function, which reduces its resource usage when implemented in hardware. Its cost function $C_{\text{ad}}$ is the absolute difference of the pixel values

$$C_{\text{ad}}(t_{1_{x,y}}, t_{2_{x+d,y}}) = |t_{1_{x,y}} - t_{2_{x+d,y}}|. \tag{6}$$

### 2.3. *Rank transform*

The rank transform, originally proposed by Zabih and Woodfill [21], and more recently used by Banks and Bennamoun [22], is an algorithm that is very robust to local brightness variations [23]. It is very suitable for hardware implementations as shown in [24], where an FPGA implementation for the application of target tracking is used, which is quite similar to finding correspondences in stereo vision. However, it is rarely used for stereo matching. One of its implementations for stereo matching is shown in [5].

The transformation function of the rank transform $T_{\text{rank}}$ is the number of pixels in a local neighborhood region $N$, having the dimensions of the block size $s_t$, with a smaller intensity value than the center pixel's value $I_{1/2}(x, y)$

$$T_{\text{rank}}(I_{1/2}, x, y, s_t) = \| \{ n, m \in N \mid I_{1/2}(n, m) < I_{1/2}(x, y) \} \|. \tag{7}$$

The cost function is the absolute difference (6) as used in the SAD algorithm.

### 2.4. *Census transform*

The census transform [21], an algorithm with even higher robustness than the rank transform [25], is often used in hardware implementations because it offers a good tradeoff between resource usage and quality. Some of the earlier implementations on FPGAs are [26, 27]. More recently, the census transform has been used for ASIC implementations in [3, 28].

The transformation consists of a comparison function $\xi$, which is used to compare the center pixel's value $i_1$ with the pixel intensity values $i_2$ in the neighborhood region $N$,

$$\xi(i_1, i_2) = \begin{cases} 1 \mid i_1 > i_2, \\ 0 \mid i_1 \leq i_2. \end{cases} \tag{8}$$

Its result, 1 if the center pixel is larger, and otherwise 0, is then concatenated ($\otimes$) to a bit vector. Thus, the transformation function $T_{\text{census}}$ is defined as

$$T_{\text{census}}(I_{1/2}, x, y, s_t) = \bigotimes_{[n,m] \in N} \xi[I_{1/2}(x, y), I_{1/2}(n, m)]. \tag{9}$$

The cost function $C_{\text{census}}$ is defined as the hamming distance over the bit vectors

$$C_{\text{census}}(t_{1_{x,y}}, t_{2_{x+d,y}}) = h\,\text{dist}(t_{1_{x,y}}, t_{2_{x+d,y}}). \tag{10}$$

## 3. FLEXIBLE HARDWARE-BASED STEREO MATCHING

### 3.1. *Flexible disparity range*

To enable the disparity range's flexible calculation we split it into $r$ partitions with $d_r$ pixels and compute every single partition separately as depicted in Figure 1. That is, for each image line, $r$ matching rounds are performed. The position of the partition's best match and its aggregated matching costs are stored in memory. Once all partitions are computed, the stored results are searched for the partition with the smallest matching value and its position value is selected as the resulting disparity. Thus, the disparity search is redefined as

$$\begin{aligned} d_{\text{map}_{x,y}} &= n \mid a_{x,y,n} \\ &= \min[\min(a_{x,y,0}, \dots, a_{x,y,d_r-1}), \dots, \\ &\qquad \min(a_{x,y,(r-1)\cdot d_r}, \dots, a_{x,y,r\cdot d_r-1})]. \end{aligned} \tag{11}$$

Using (11) for the disparity search, the maximum disparity $d_{\max}$ is defined by

$$d_{\max} = r \cdot d_r - 1. \tag{12}$$

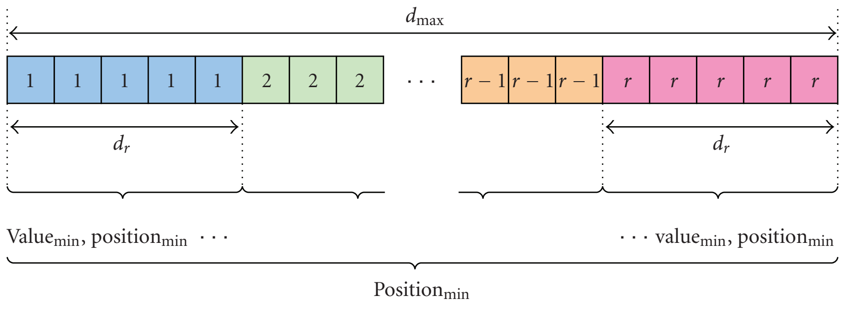
$d_{\max}$
1
2
$r-1$
$r$
$d_r$
$d_r$
Value$_{\min}$, position$_{\min}$ · · ·
· · · value$_{\min}$, position$_{\min}$
Position$_{\min}$

FIGURE 1: Disparity range split into $r$ partitions.

Now the flexible disparity range is achieved by varying the number of calculation rounds $r$. The highest possible disparity range is only limited by the memory used to store the calculation rounds' interim values, that is, the position and costs of the best match for each pixel in the current round.

The key to a well-performing implementation is to pipeline the whole calculation. Pipelines enable a highly parallel execution but also require initialization times at the start of computation. The reason for computing one line per round, instead of one pixel per round, is to keep these pipeline initializations to a minimum. The price for this performance increase is a greater demand for memory, because the interim values, that is, the partitions' minimum values and positions, of the whole line need to be stored before the global minimum can be searched for.

Figure 2 presents the block diagram for the two-staged implementation of our round-based calculation technique.

The calculation of the partitions' interim values forms the first stage of the circuit. Here, the data of the stereo images are provided in the chip's internal memory. The stereo matching circuit reads the image data from the memory and calculates the matching costs for the disparity range partition of the current round only. Then the matching costs' minimum value is searched for. The selected minimum value and its position are stored in the internal memory. For better support of the memory architecture within FPGAs, which memory blocks have a fixed size and position on the chip surface, we use two separate blocks of memory for the matching costs' value and position. This way, merging single memory blocks with different positions on the chip can be avoided, or at least kept at a minimum. For each calculation round, the interim values are stored in a different pair of memory blocks. The current memory block is selected using a demultiplexer that has the calculation round counter as the input.

After all $r$ calculation rounds have been performed, the final disparity value can be searched for by the extraction stage. Here, the pairs of block memory holding the interim values are accessed using a multiplexer that has the extraction round counter as the input. Once the last partition value has been analyzed, the smallest value's position is read and selected as the final disparity value.

For better resource utilization, we suggest duplicating the interim values' memory and assigning one set of memory to each stage, separating the two computations. By switching the assignment of the memory sets after each computed line, the disparity can be extracted for the previous line while the next interim values are calculated for the current line. This way, the two computations can be performed in parallel, resulting in a two-staged macropipeline. Figure 3 shows the pipelined circuit, where the assignment of the memory sets is performed by connecting it to the calculation stage using a de-multiplexer and a multiplexer for the extraction stage. The assignment of the sets is performed using the line counters' least significant bit (LSB) and negating it for the extraction stage.

### 3.2. Flexible block sizes

For the resource-efficient implementation of a flexible block size it is necessary to reuse the computation blocks, changing mostly the interconnects only, rather than switching between different computation blocks for each block size or deactivating large parts of them. To reuse a computation block for a larger block size, it is necessary that the computation can be split into two or more smaller functions similar to the predefined, using different input parameters. The results of these functions have to be merged afterwards. For a reuse of resources it must be guaranteed that the further computation is performed in the same way as for the smaller block size.

The transformation function (1) has the block size as a parameter and is therefore suitable for a deeper analysis. It seems possible to use two transformation functions and merge the results for the neighboring blocks, but the transformation function is not defined in general. Examining our three presented algorithms, for the rank transform an aggregation would apply as a merging function, while for the census transform the bitstream had to be merged. The SAD algorithm does not even have a transformation function. This leads to the conclusion that it is not possible to define a generic function for merging the transformation function's results and a technique that reuses the transformation function blocks would be restricted to specific algorithms.

The cost function (2) does not have the block size as a parameter and therefore does not directly correspond to it.

The aggregation function (3) is similar for all area-based algorithms [11]. Using a larger block size for the aggregation function leads directly to a higher number of matching cost summations. When the larger block size's number of summations is a multiple of a smaller one, the aggregation

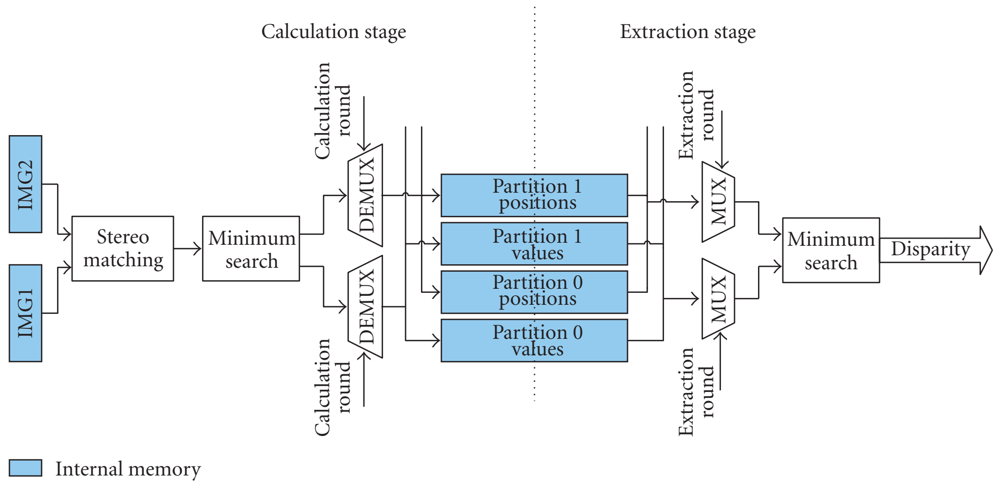

Figure 2: Round-based disparity calculation.

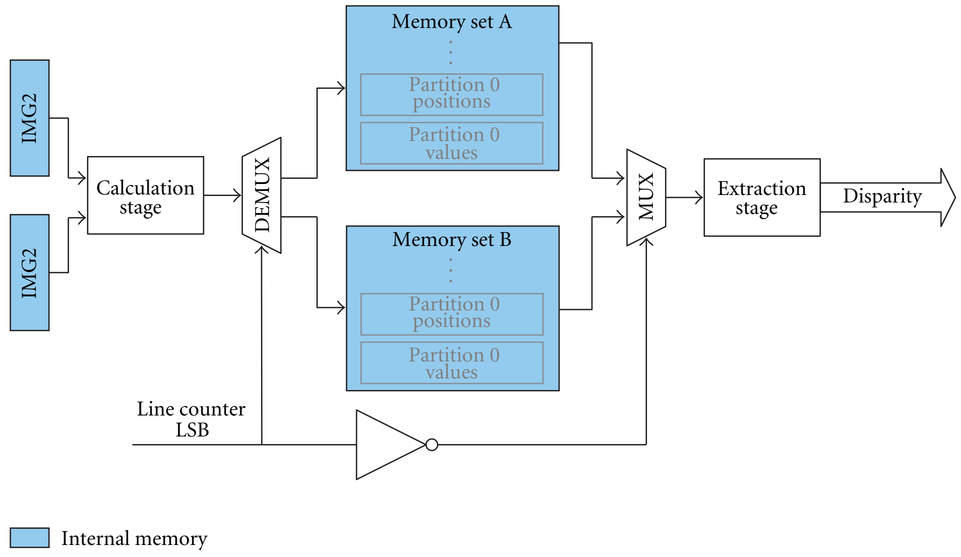

Figure 3: Pipelined disparity calculation.

blocks can be reused, having different input parameters. Equation (13) shows the new aggregation function for a horizontal enlargement of the aggregation block size from $s_{a_{\text{small}}}$ to $s_{a_{\text{large}}}$:

$$a_{\text{large}_{x,y,d}} = \sum_{n}\sum_{m_1} c_{x+m_1,y+n,d} + \cdots + \sum_{n}\sum_{m_k} c_{x+m_k,y+n,d}, \quad (13)$$

where

$$n \in \left[-\frac{s_a - 1}{2}, \frac{s_a - 1}{2}\right],$$

$$m_l \in \left[-\frac{s_{a_{\text{large}}} - 1}{2} + (l-1)s_{a_{\text{small}}}, -\frac{s_{a_{\text{large}}} - 1}{2} + l \cdot s_{a_{\text{small}}} - 1\right], \quad (14)$$

$k$ is the block size multiplication factor, and $a_{\text{large}_{x,y,d}}$ are the aggregated matching costs when using the enlarged block size.

As can be seen, the new aggregation function's sub-aggregations are similar to the original one in (3), with the exception that the boundaries are calculated differently. This results in the requirement for different inputs for the aggregation blocks.

The input parameters of the aggregation function are delivered by the cost function. Thus, switching the input parameters has to be performed before the cost function.

The results of the small aggregation units can be merged by $k - 1$ summations, when using the larger block size. Additionally, the input and output values have to be switched, except for the input of the primary image for the first cost function of each larger block, which always stays the same. This leads to $3 \cdot k - 1$ required additional $k$-input multiplexers.

Figure 4 shows the block diagram for the implementation, using $k = 2$. Here, the data of the transformed stereo images are supplied in the chip's registers. These registers have to be shifted when switching to the next pixel's computation. For refilling the resulting emptied registers, the next pixels' transformation has to be performed preliminarily. The inputs of the cost functions and the outputs of the aggregation blocks are switched using multiplexers, which have the selected block size as the input.

When the smaller block size is selected, that is, the multiplexer's input 0 is selected, the even cost functions and aggregation blocks are connected, calculating the matching costs for the even disparity levels. Likewise, the odd cost functions and their aggregation blocks are connected for calculating the costs for the odd disparity levels.

As soon as the larger block size is selected, that is, the multiplexers' input 1 is selected, the even and the odd aggregation blocks, as well as the corresponding cost functions, are combined for calculating the same disparity level together. Now, the even part is used to calculate the left part of the larger block and the odd part calculates the right part. Both aggregation blocks' results are merged by a single summation and output as the even aggregation block's result. The odd aggregation block does not output a valid result in this situation. Therefore, its output is set to a higher value than the maximum achievable matching costs, for example, the maximum value plus one. This way, the WTA algorithm is forced to ignore these results, since the even aggregation block's valid results are by definition smaller. In any case, the output position of the WTA algorithm is now scaled by two and has to be adjusted by cutting off its LSB, dividing it by two.

The enhancement of the block size goes along with a reduction of the calculated disparity levels per calculation round. That is, the partition size $d_r$ is divided by $k$. To ensure a constant disparity range, the number of calculation rounds has to be increased by the same factor. This leads to a lower frame rate of the stereo matching algorithm, when using the larger block size.

### 3.3. Flexible frame rate

The flexibility of the frame rate results from using a flexible disparity range and block size. When the demand for a higher frame rate exists (e.g., the robot operates at higher speed) the disparity range and the block size of the stereo matching algorithm can be reduced to ensure a shorter processing time.

Thus, the present frame rate of the system in fps is defined as

$$\text{fps}\cdot\frac{d_{\max}+1}{d_r}\sim\text{const}, \qquad (15)$$

where

$$d_r = d_{r_{\max}}\cdot\frac{1}{k_{\text{akt}}}, \qquad (16)$$

$k_{\text{akt}}$ is the currently selected block size multiplication factor, and $d_{r_{\max}}$ is the achievable disparity per round when $k_{\text{akt}} = 1$. The constant factor in (15) depends on the processing speed of the implemented design.

## 4. EXPERIMENTAL EVALUATION

### 4.1. Evaluation architecture

To evaluate the performance of our flexible stereo matching technique, we implemented a generic hardware-based stereo matching architecture that is suitable for different area-based matching algorithms.

Our architecture consists of three major pipeline stages as depicted in Figure 5, which are capable of working in parallel after a primary initialization and synchronization phase. These three pipeline stages are the input, the calculation, and the extraction stage. The interfaces between those stages are formed by memory blocks, each holding a full image line.

The application of our hardware-based stereo matching architecture includes robot navigation. This task requires the constant collection of data about the surrounding background to enable the localization of the robot's position. Thus, for the implementation of our architecture we assume that the disparity range has a fixed minimum value of $d_{\min} = 0$ and therefore the background of the images is always searched for as well.

Furthermore, for the last $d_{\max}$ pixels, there would be insufficient image data available in the secondary image to perform the matching for the whole disparity range. Since this would lead to border effects in the disparity map, we do not waste resources on the matching of this image region and simply stop the calculation at this point. Thus, the output disparity map's image width is reduced by $d_{\max}$ pixels. Also, the border regions of the image that would have border effects caused by the total block size $s_b$ are not processed. These are the first and the last $(s_b - 1)/2$ image lines as well as the lines' first and last $(s_b - 1)/2$ pixels.

For a less complex implementation, we assume that the images used for the matching procedure are rectified [29] by an additional preprocessing stage. Thus, the search for correspondences is limited to the image lines.

The first stage of our architecture is the input stage. It reads the incoming image data from the input port and stores it in the internal memory. Here, we use one memory block for each single image line. The number of stored lines is defined by the total block size $s_b$ plus one additional line to enable the implementation of a cyclic memory. This way, the input stage can access one memory block in write mode while the calculation stage reads the others, ensuring maximum independence of each stage.

The calculation stage performs the round-based matching as soon as the input stage has stored the first $s_b$ image lines. It reads the image data necessary to perform the next pixel's matching from the memory. If the algorithm used consists of a transformation, the pixels are transformed instantly and stored at the end in the image buffer. This image buffer holds the transformed values for the following cost function. It has a height of $s_a$ pixels, which is the aggregation block size, and a width of $d_r + s_a$ pixels. This image buffer is shifted by one pixel column after each pixel's computation, that is, at each clock cycle. Then our flexible block matching technique is applied as described in Section 3.2. The resulting aggregated matching costs of the current disparity partition

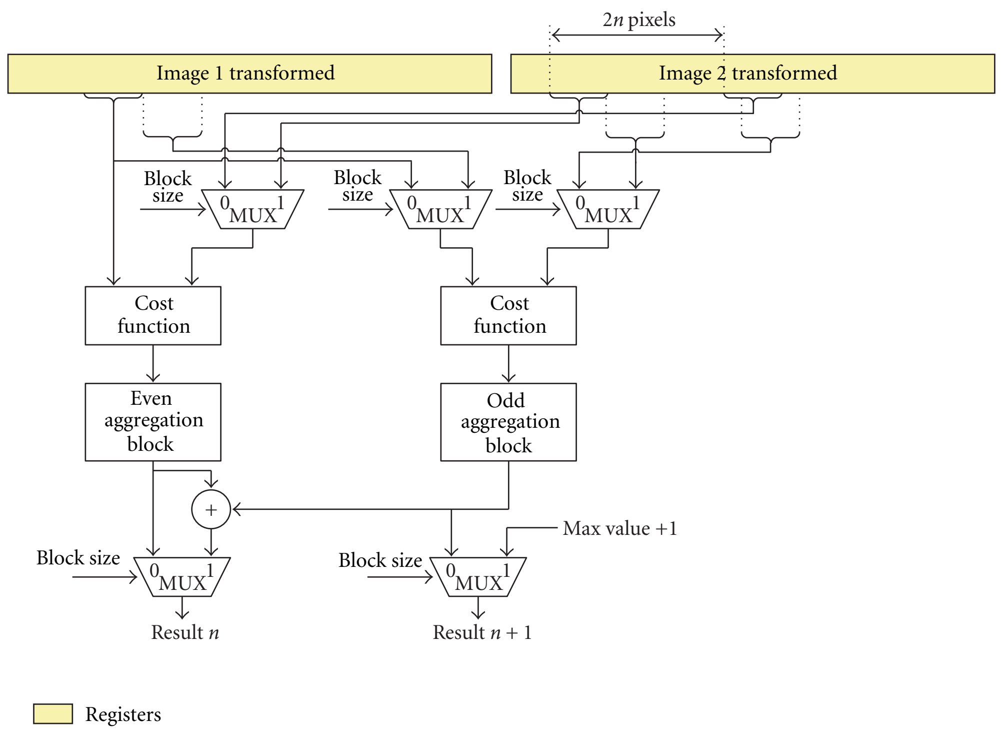

FIGURE 4: Flexible block size using $k = 2$.

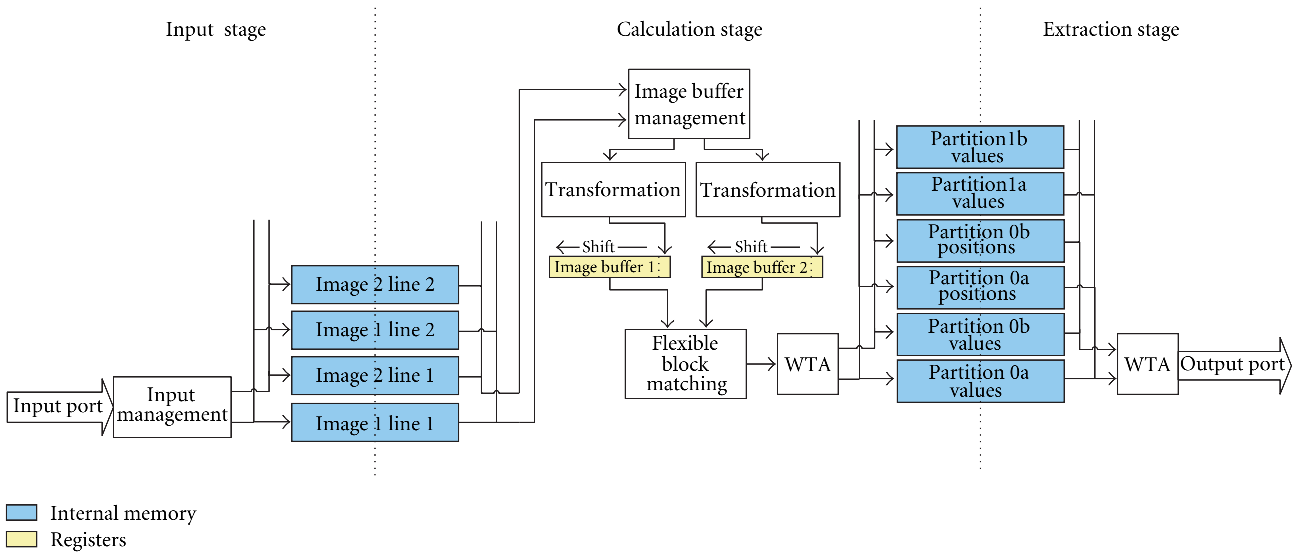

FIGURE 5: Evaluation architecture.

are the input for the following WTA algorithm. This WTA algorithm is implemented as a binary search tree as shown in Figure 6 and therefore fully pipelined. It outputs the minimum value and its position. Both are stored in the interface memory.

The pipelined implementation of the calculation stage results in a processing time of one clock cycle per pixel as soon as the pipeline is filled. Depending on the algorithm implemented and the partition size, this initialization time $t_{\text{init-calc}}$ varies between 50 and 70 clock cycles for each calculation round. Thus, the computation time of one image line is $r \cdot (\text{image\_width} - d_{\max} - s_b + 1 + t_{\text{init-calc}})$ clock cycles.

As described in Section 3.1, the interface between the calculation and the extraction stage is formed by two sets of memory blocks, which are switched between the stages. Hence, there exist two pairs of memory blocks for each round's interim values, namely $a$ and $b$ in Figure 5.

The extraction stage reads the results of all matching rounds and again selects the best match by using the WTA algorithm. Here, the WTA is implemented in series. Thus,

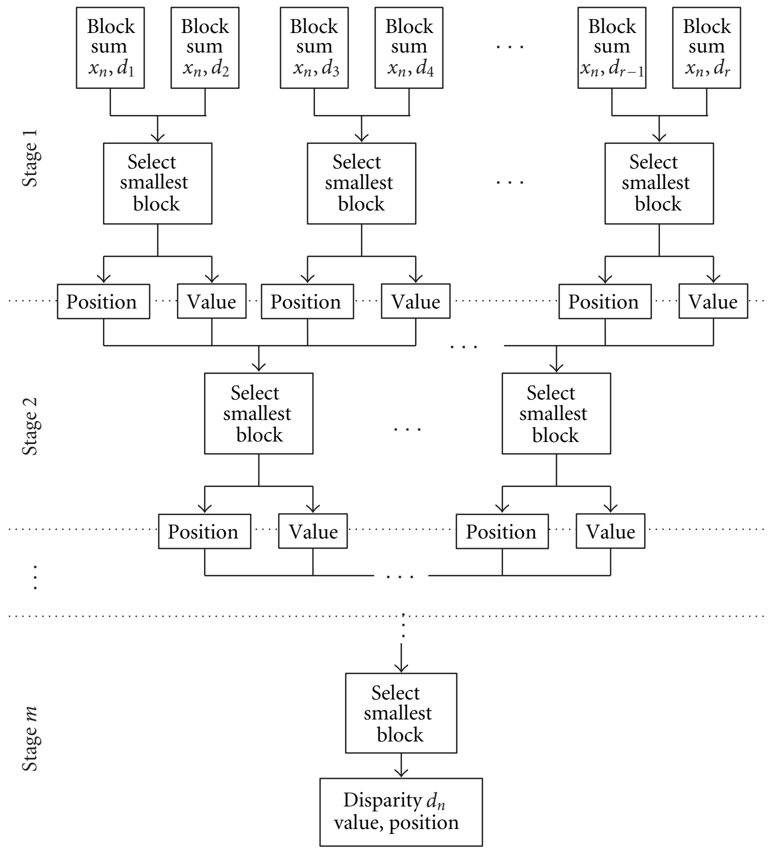

Figure 6: Tree-based WTA.

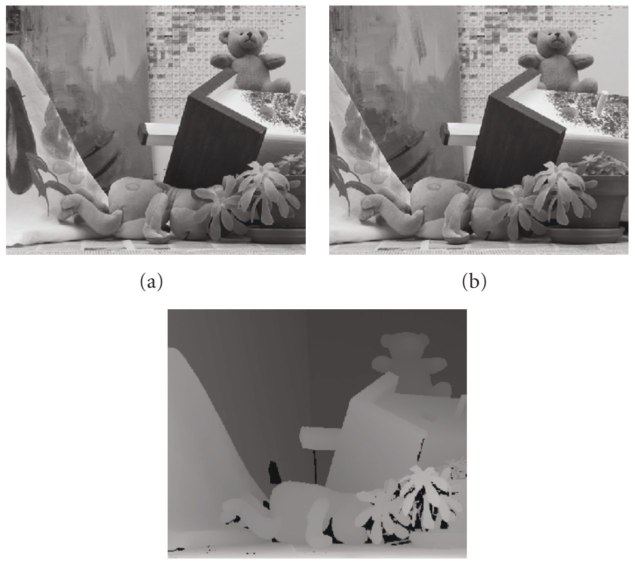

Figure 7: Teddy images from the Middlebury dataset. (a), (b): camera images; (c): ground truth image.

the value and position of just one round is evaluated per clock cycle and the minimum value stored in the registers. When evaluating the pixel's last round, either the stored position or the current round's position is set on the output port as the resulting disparity. The serial implementation of the WTA algorithm is more resource aware than the parallel one of the calculation stage. The extraction stage needs to be initialized only once per line, in contrast to the calculation stage, which has to be initialized $r$ times per line. Furthermore, the pipeline of the extraction stage is much shorter than that of the calculation stage, having an initialization time $t_{\text{init-extr}}$ of two to five clock cycles, depending on the chip's memory access delay. Thus, the extraction stage takes $t_{\text{init-extr}} + r \cdot (\text{image\_width} - d_{\max} - s_b + 1)$ clock cycles per image line, which is less than the calculation stage's computation time.

The total processing time of the architecture is given by the time to read the first $s_b$ image lines, the processing time of the calculation stage for the following image_height $- s_b + 1$ image lines, and the extraction stage for the very last image line.

### 4.2. Test configuration

We evaluated our algorithms using the teddy images, illustrated in Figure 7, from the Middlebury stereo dataset [30], since this image set offers high- as well as low-textured surfaces. This image set has a resolution of $450 \times 375$, a maximum disparity of 60 and is converted to 8-bit gray scale. The ground truth of these images, which is defined as

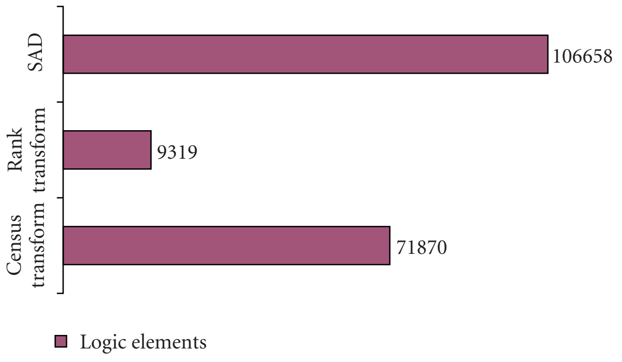

FIGURE 8: Required logic elements of the algorithms.

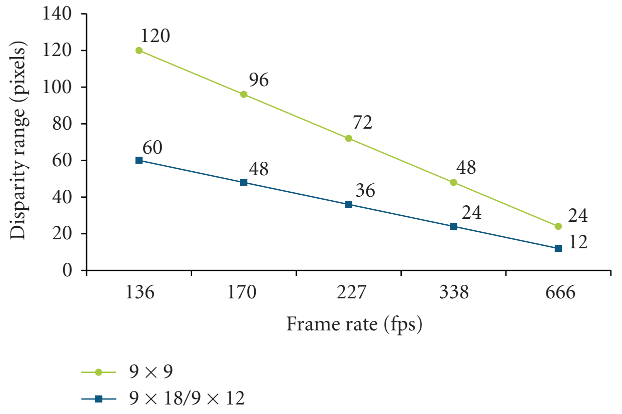

FIGURE 9: Frame rate vesus disparity range for block sizes $s_b = 9 \times 9$, $9 \times 18$ for SAD and $9 \times 12$ for rank and census transform, respectively.

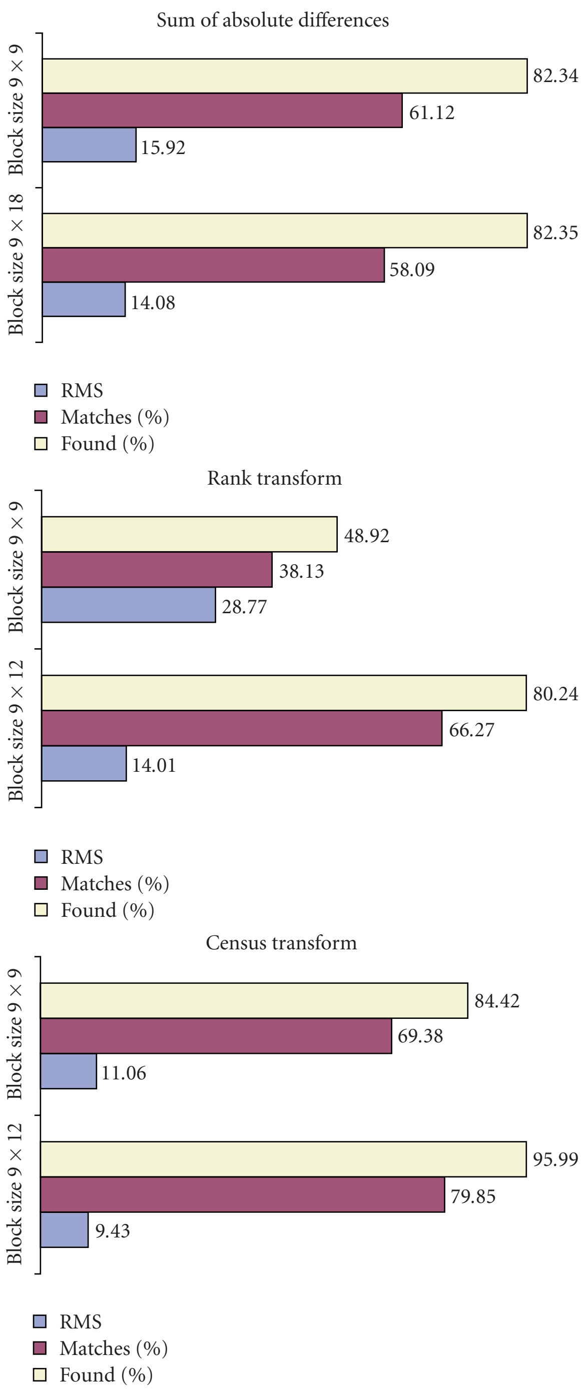

FIGURE 10: Evaluated algorithm quality for block sizes $s_b = 9 \times 9$, $9 \times 18$ for SAD, and $9 \times 12$ for rank and census transform, respectively.

the measured reference disparity map, is scaled by factor of 4. Thus, we also scaled our disparity maps this way to ensure a comparable result.

For the block size multiplication factor we used $k = 2$, since we expect that doubling the horizontal block size will be the most typical application. The disparity range was 120 to avoid distortions caused by a too small disparity range for both block sizes. We used a round size of 5, resulting in a partition size of 24 disparity levels per calculation round.

To demonstrate the need for a flexible disparity range, we also evaluated the image sets using a disparity range of 60. With the larger block size, this resulted in an effective disparity range of 30, which is half of the maximum disparity in the image sets.

The SAD algorithm was configured to $s_b = 9 \times 9$ for the smaller and $s_b = 9 \times 18$ for the larger block size. The rank and census transform also used $s_b = 9 \times 9$, distributed to $s_t = 7 \times 7$ for the transformation and $s_a = 3 \times 3$ for the aggregation, leading to $s_a = 3 \times 6$ and $s_b = 9 \times 12$ for the larger block size.

To evaluate the algorithms' quality, we performed a left/right consistency check and calculated the root mean square (RMS) over the deviations to the ground truth image as well as the found pixels and correct matches, which are within a maximum deviation of one pixel.

### 4.3. *Experimental results*

We synthesized the implemented algorithms for an Altera EP2S130 with Altera Quartus II in order to analyze the algorithm's resource usage. All of them had a total memory consumption of 425 984 bits. Here, the image memory requires 81 920 bits for storing 10 lines for both images. Furthermore, storing the interim position requires 114 688

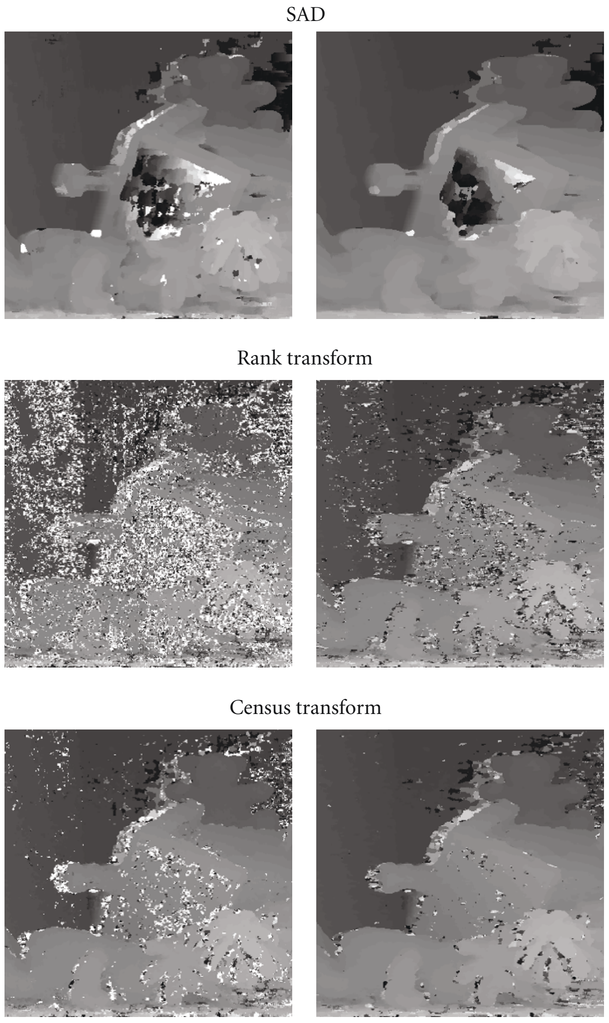

Figure 11: Disparity maps generated from the Middlebury dataset's teddy images. Left: block size $s_b = 9 \times 9$; right: block size $9 \times 18$ for SAD, and $9 \times 12$ for rank and census transform, respectively.

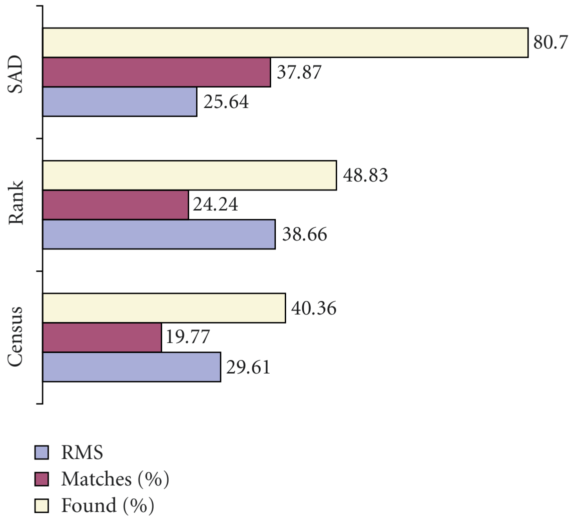

Figure 12: Evaluated algorithm quality with $d = 30$ and block size $9 \times 18$ for SAD and $9 \times 12$ for rank and census transform, respectively.

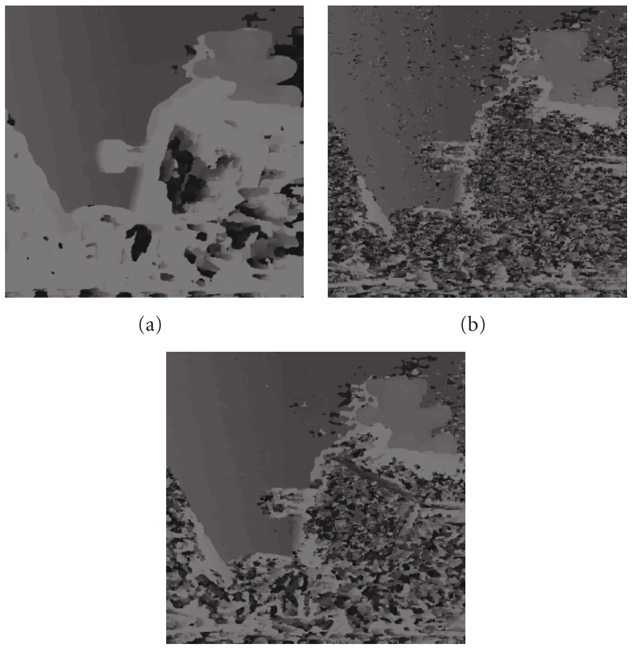

Figure 13: Disparity maps with $d = 30$ and block size $9 \times 18$ for SAD and $9 \times 12$ for rank and census transform, respectively. (a): SAD; (b): rank transform; (c): census transform.

memory bits and storing the 16-bit matching cost values requires 229 376 memory bits.

The algorithms' logic consumption in terms of logic elements is presented in Figure 8. The results show the highest values for the SAD algorithm due to its high number of aggregations, while the census transform requires slightly less logic. The rank transform has by far the smallest logic requirements, being 3.7 times smaller than that of the census transform. This is the reason, why the rank transform is of special interest for low-cost embedded stereo vision systems, even if it is well known that the census transform is the more robust algorithm [25].

Figure 9 reveals the disparity range versus the achieved frame rate for both block sizes using our flexible technique. As can be seen, the system is able to select the algorithm's features as required out of a disparity range between 12 and 120 pixels, a frame rate ranging from 136 fps to 666 fps, and a total block size of either $9 \times 9$ or $9 \times 18$ for SAD and $9 \times 12$ for the rank and census transform. As defined by (15), the frame rate times the number of calculation rounds ($(d_{\max} + 1)/d_r$) is nearly a constant. This shows that the static initialization times of our architecture, like filling the image memory or the final examination of the last image line, are negligible compared to the time required for the computation itself. Finally, it is illustrated that the disparity range can be easily doubled by selecting the smaller block size of $s_b = 9 \times 9$.

Our architecture produces $330 \times 375$ disparity maps with 120 disparity levels at 136 fps, which makes a total amount of about 2 billion disparity calculations per second. The DeepSea ASIC [31], producing $512 \times 480$ disparity maps with 52 disparity levels at 200 fps, has a total of 2.55 billion disparity calculations per second. Thus, our FPGA-based stereo architecture is just 22% slower than this ASIC-based solution, while having a 65% times larger total block size of $9 \times 9$ rather than $7 \times 7$.

Kuon and Rose [32] evaluated the achievable performance gain when moving from an FPGA to an ASIC. For FPGA designs making use of the internal block memory, they revealed a performance gain factor between 2.8 and 4.3, being 3.5 on average, when implementing the same design as an ASIC. This further outlines the potential of our flexible stereo matching technique, since it would outperform the DeepSea ASIC implementation by a minimum factor of 2.2 when implemented on the same platform, while offering the advantages of an adjustable stereo algorithm.

The time span to reconfigure the algorithm to a new feature set is one clock cycle. This reconfiguration is incorporated into the architecture's state change cycle that takes place before each frame is processed. Thus, no measurable reconfiguration time is required for adapting the algorithm's features.

The achieved RMS, found pixels, and correct matches of each algorithm on the teddy images are presented in Figure 10. The disparity maps are shown in Figure 11. Image areas, in which our architecture did not calculate the disparity in order to avoid border effects, are black and disregarded for the evaluation.

Both, the measurements and the depicted disparity maps show a great advantage in quality for the disparity maps from the rank and census transform when using the larger block size. This shows, that our technique also improves the quality of the output disparity maps, even if the block size of the transforms is constant and only the aggregation is enlarged. The SAD algorithm has an improved RMS at the larger block size, but the number of correct matches is slightly decreased. The disparity maps show that in this specific configuration, the house's roof in the teddy image contains noticeably more information for the larger block size, while the deformations caused by the block size in the rest of the image outweigh this advantage.

The measured quality for the insufficient disparity range is presented in Figure 12 and the corresponding disparity maps in Figure 13. It is noticeable that the reduced disparity range highly reduces the number of found and correct matches and increases the RMS. This further emphasizes the need for a correct adjustment of the disparity range.

## 5. CONCLUSIONS

The proposed flexible hardware-based stereo matching technique gives the ability to adapt the algorithm to varying application driven demands and therefore to ensure a highly accurate disparity map in real-time embedded stereo vision systems even under changing conditions. The flexible disparity range gives the opportunity to dynamically balance the detection of close objects against the frame rate. The opportunity to balance the disparity map's noise against the frame rate is assured by a flexible block size.

Our experimental evaluation of the presented technique shows a good response of the increased block size on low-textured surfaces for all algorithms, even if the transformation block size remains constant. In particular, the rank transform denotes good performance when a resource-aware implementation is demanded. Using our hardware-based stereo matching architecture, the census transform has the best overall quality. We pointed out that a correct disparity range is essential for a high-quality disparity map, which is easily achievable using our novel flexible technique. Due to the basic principle of our technique, only small amounts of additional memory, multiplexers, and summations are required, while remaining consistent with ASIC implementations by avoiding dynamic reconfiguration.

We compared our flexible stereo matching architecture with a well-known ASIC implementation, outlining the high performance of our architecture. Finally, we revealed the even higher potential of our flexible stereo matching technique when implemented on ASICs.

## ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Sixth Framework Programme (FP6/2003-2006) under Grant agreement no. FP6-2006-IST-6-045350 (robots@home).

## REFERENCES

[1] L. Mingxiang and J. Yunde, "Stereo vision system on programmable chip (SVSoC) for small robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pp. 1359–1365, Beijing, China, October 2006.

[2] A. Naoulou, J.-L. Boizard, J. Y. Fourniols, and M. Devy, "A 3D real-time vision system based on passive stereo vision algorithms: application to laparoscopic surgical manipulations," in *Proceedings of the 2nd International Conference on Information and Communication Technologies (ICTTA '06)*, vol. 1, pp. 1068–1073, Damascus, Syria, April 2006.

[3] J. I. Woodfill, G. Gordon, and R. Buck, "The Tyzx DeepSea high speed stereo vision system," in *Proceedings of the Conference on Computer Vision and Pattern Recoginition Workshop (CVPR '04)*, vol. 3, p. 41, Washington, DC, USA, June-July 2004.

[4] L. Matthies, M. Maimone, A. Johnson, et al., "Computer vision on Mars," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 67–92, 2007.

[5] R. B. Porter and N. W. Bergmann, "A generic implementation framework for FPGA based stereo matching," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 2, pp. 461–464, Brisbane, Australia, December 1997.

[6] S. H. Lee, J. Yi, and J. S. Kim, "Real-time stereo vision on a reconfigurable system," in *Proceedings of the 5th International*

*Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '05)*, vol. 3553 of *Lecture Notes in Computer Science*, pp. 299–307, Samos, Greece, July 2005.

[7] S. Perri, D. Colonna, P. Zicari, and P. Corsonello, "SAD-based stereo matching circuit for FPGAs," in *Proceedings of the 13th IEEE International Conference on Electronics, Circuits, and Systems (ICECS '06)*, pp. 846–849, Nice, France, December 2006.

[8] R. P. Jacobi, R. B. Cardoso, and G. A. Borges, "VoC: a reconfigurable matrix for stereo vision processing," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS '06)*, p. 6, Rhodes Island, Greece, April 2006.

[9] 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, 2003.

[10] D. J. Fleet, "Disparity from local weighted phase-correlation," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (ICSMC '94)*, vol. 1, pp. 48–54, San Antonio, Tex, USA, October 1994.

[11] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002.

[12] A. Koschan, V. Rodehorst, and K. Spiller, "Color stereo vision using hierarchical block matching and active color illumination," in *Proceedings of the 13th International Conference on Pattern Recognition (ICPR '96)*, vol. 1, pp. 835–839, Vienna, Austria, August 1996.

[13] W. Maass, "On the computational power of winner-take-all," *Neural Computation*, vol. 12, no. 11, pp. 2519–2535, 2000.

[14] 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, no. 9, pp. 920–932, 1994.

[15] J. Banks, M. Bennamoun, and P. Corke, "Non-parametric techniques for fast and robust stereo matching," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 1, pp. 365–368, Brisbane, Australia, December 1997.

[16] A. E. Kayaalp and J. L. Eckman, "Near real-time stereo range detection using a pipeline architecture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 6, pp. 1461–1469, 1990.

[17] C. Cuadrado, A. Zuloaga, J. L. Martín, J. Lázaro, and J. Jiménez, "Real-time stereo vision processing system in a FPGA," in *Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics (IECON '06)*, pp. 3455–3460, Paris, France, November 2006.

[18] Y. Jia, M. Li, L. An, and X. Zhang, "Autonomous navigation of a miniature mobile robot using real-time trinocular stereo machine," in *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP '03)*, vol. 1, pp. 417–421, Changsha, China, October 2003.

[19] G. van der Wal, M. Hansen, and M. Piacentino, "The Acadia vision processor," in *Proceedings of the 5th IEEE International Workshop on Computer Architectures for Machine Perception (CAMP '00)*, pp. 31–40, Padova, Italy, September 2000.

[20] J. S. Yi, J. S. Kim, L. P. Li, J. Morris, G. Lee, and P. Leclercq, "Real-time three dimensional vision," in *Proceedings of the 9th Asia-Pacific Conference on Advances in Computer Systems Architecture (ACSAC '04)*, vol. 3189 of *Lecture Notes in Computer Science*, pp. 309–320, Beijing, China, September 2004.

[21] R. Zabih and J. I. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proceedings of the 3rd European Conference on Computer Vision (ECCV '94)*, pp. 151–158, Stockholm, Sweden, May 1994.

[22] J. Banks and M. Bennamoun, "Reliability analysis of the rank transform for stereo matching," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 31, no. 6, pp. 870–880, 2001.

[23] H. Hirschmüller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '07)*, pp. 1–8, Minneapolis, Minn, USA, June 2007.

[24] J. D. Anderson, *Semi autonomous vehicle intelligence: real time target tracking for vision guided autonomous vehicles*, M.S. thesis, Brigham Young University, Provo, Utah, USA, 2007.

[25] B. Cyganek, "Comparison of nonparametric transformations and bit vector matching for stereo correlation," in *Proceedings of the 10th International Workshop on Combinatorial Image Analysis (IWCIA '04)*, vol. 3322 of *Lecture Notes in Computer Science*, pp. 534–547, Auckland, New Zealand, December 2004.

[26] P. Corke and P. Dunn, "Real-time stereopsis using FPGAs," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 1, pp. 235–238, Brisbane, Australia, December 1997.

[27] J. I. Woodfill and B. Von Herzen, "Real-time stereo vision on the PARTS reconfigurable computer," in *Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FPGA '97)*, pp. 201–210, Napa Valley, Calif, USA, April 1997.

[28] M. Kuhn, S. Moser, O. Isler, et al., "Efficient ASIC implementation of a real-time depth mapping stereo vision system," in *Proceedings of the of the 46th IEEE Midwest International Symposium on Circuits and Systems (MWSCAS '03)*, pp. 1478–1481, Cairo, Egypt, December 2003.

[29] Z. Zhang, "Determining the epipolar geometry and its uncertainty: a review," *International Journal of Computer Vision*, vol. 27, no. 2, pp. 161–195, 1998.

[30] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, vol. 1, pp. 195–202, Madison, Wis, USA, June 2003.

[31] J. I. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck, "The Tyzx DeepSea G2 vision aystem, a taskable, embedded stereo camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '06)*, p. 126, New York, NY, USA, June 2006.

[32] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.