Towards Accurate Hardware Stereo Correspondence:

A Real-Time FPGA Implementation of a Segmentation-Based Adaptive Support Weight Algorithm

C. Ttofis and T. Theocharides
KIOS Research Center for Intelligent Systems and Networks
Department of ECE, University of Cyprus
Nicosia, Cyprus

***Abstract*—Disparity estimation in stereoscopic vision is a vital step for the extraction of depth information from stereo images. This paper presents the hardware implementation of a disparity estimation system that enables good performance in both accuracy and speed. The architecture implements an adaptive support weight stereo correspondence algorithm, which integrates information obtained from image segmentation, in an attempt to increase the robustness of the matching process. The proposed system integrates optimization techniques that make the algorithm hardware-friendly and suitable for embedded vision systems. A prototype of the architecture was implemented on an FPGA, achieving 30 fps for 640x480 image sizes. The quality of the disparity maps generated by the proposed system is also better than other existing hardware implementations featuring fixed support local correspondence methods.**

Keywords-Computer Vision; Stereo Correspondence; FPGAs

I. INTRODUCTION

Stereo correspondence is one of the most extensively studied problems in computer vision [1]. It takes a pair of rectified stereo images as input, and computes a disparity map, from which information about the depth of objects in an image frame relative to the location of the camera(s) can be extracted [1]-[2]. Emerging applications of stereo correspondence such as robot navigation, space and avionics, and obstacle detection for autonomous vehicles, require real-time processing speed and low power consumption. In such cases, software implementations of stereo correspondence algorithms usually struggle to meet these constraints, and as a result, some form of hardware acceleration or even a complete custom hardware implementation is preferred [3].

Hardware acceleration of stereo correspondence algorithms has been done extensively using Digital Signal Processors (DSPs) and Graphics Processing Units (GPUs) [3]. These systems though, involve architectures that are generally not suitable for embedded applications; DSPs do not provide enough computational power to achieve parallel processing, while GPUs consume excessive power. Dedicated hardware approaches, on the other hand, can provide the necessary computational power and the energy efficiency. However, complex disparity estimation algorithms, such as local, based on an adaptive support weight [4], or global [2], are hard to implement in hardware. These algorithms usually achieve better quality compared to simple local algorithms, but are hardware-unfriendly and bandwidth-hungry. Consequently, most dedicated hardware implementations of disparity estimation algorithms adopt simple fixed support window methods that achieve relatively low disparity accuracy [3]-[4].

This work was partially supported by E! 5527 – RUNNER, an EU-funded project under the EUREKA's Eurostars Programme.

This paper presents a hardware implementation of an adaptive support weight (ADSW) disparity estimation algorithm that integrates information obtained from image segmentation in order to increase the accuracy of the resulting disparity maps. The proposed hardware system extends the initial idea proposed in [5] that was implemented in software, by integrating hardware-directed optimization techniques, which aim to make the algorithm hardware-friendly and able to reach real-time performance. Furthermore, edge-directed information is included in order to reduce the search space and further increase the processing speed, an idea successfully applied to a hardware implementation of a SAD-based fixed support algorithm [6]. The proposed system architecture was implemented on a Virtex-5 FPGA platform, yielding real-time performance (30 frames per second – fps) for a stereo image pair of 640x480. Moreover, the proposed system is able to improve the quality of the disparity maps compared to other hardware implementations that feature fixed support local stereo correspondence algorithms.

The rest of this paper is organized as follows. Section II presents background information and reviews related work. Section III describes the algorithm implemented by the proposed disparity estimation system. Section IV presents the proposed system architecture. Section V presents the experimental platform and implementation results. Section VI concludes the paper and discusses future work.

II. BACKGROUND & RELATED WORK

A. Stereo Correspondence Overview

Stereo correspondence infers depth information from a pair of rectified stereo images (called *reference* and *target images*) by locating corresponding pixels between the reference image I_r and the target image I_t. Given that the input images are rectified, the correspondence of a pixel at coordinate (x, y) can only be found at the same vertical coordinate y, and within a maximum horizontal bound, called *disparity range D* (d_m-d_M) [2]. The disparity is computed as the absolute difference between the coordinates of the corresponding pixels in I_r and I_t. The disparities of all corresponding pixels form a disparity map. The stereo correspondence process is detailed in [2].

According to [2], stereo correspondence algorithms mostly follow four steps: (1) compute a matching cost for each pixel at each disparity, (2) aggregate costs across pixels at the same disparity, (3) calculate the best disparities based on the aggregated costs, and (4) refine the disparity map. Moreover, [2] classifies stereo correspondence algorithms into two broad categories: *global* and *local*. Global algorithms can produce very accurate results but are computationally more demanding

978-3-9810801-8-6/DATE12/©2012 EDAA

compared to local algorithms, due to their iterative nature [3]. On the other hand, local algorithms are less computationally expensive, hence suitable for the majority of real-time applications. This work targets real-time embedded vision applications, thus employs a local algorithm.

Local algorithms determine the disparity of pixel p in I_r by finding the pixel of I_t that yields the lowest score of a similarity measure (see [3] for a review) computed on a support (correlation) window centered on p and on each of the d_M - d_m candidates defined by the disparity range. Early local algorithms have followed a simple approach that uses a fixed (typically square) support window during the cost aggregation step. However, this approach is prone to errors as it blindly aggregates pixels belonging to different disparities, resulting in incorrect disparity estimation at depth discontinuities and regions with low texture [3]. Recent local algorithms improve this basic approach by using multiple-window and ADSW methods [4]. The latter methods represent state-of-the-art in local stereo correspondence algorithms, as they can generate disparity maps that approach the accuracy of global algorithms [3]. These methods operate by assigning different weights to the pixels in a support window based on the proximity and color distances to the center pixel [4], or on information extracted from image segmentation [5], [7]. In this way, they aggregate only those neighboring pixels that are at the same disparity. Despite their improvements in accuracy, stereo correspondence algorithms based on ADSW are generally slower than other local algorithms.

B. Real-Time Dedicated Hardware Implementations

There is a considerable amount of work about real-time dedicated hardware implementations of disparity estimation algorithms. Most of them have been implemented on FPGAs, taking advantage of the inherent hardware flexibility, in order to exploit the intrinsic parallelism of the stereo correspondence algorithm. A stereo depth measurement system on an FPGA is introduced in [8]. It generates disparities on 512x480 images at 30 fps, implementing a window-based correlation search technique. Another system presented in [9] yields 20 fps on 640x480 image sizes. In [10], a survey of some FPGA implementations, [11]-[13], is presented, and the survey highlights the common use of the SAD similarity measure, a relatively simple technique suitable for hardware. Other dedicated hardware architectures suitable for real-time disparity map estimation are presented in [14]-[15]. These works implement local fixed support algorithms using the SAD similarity measure, and compute intermediate-sized disparity maps at a rate of 768 and 600 fps, respectively. A more complex disparity estimation algorithm based on locally weighted phase correlation is presented in [16], utilizing 4 FPGAs to produce dense disparity maps of size 256x360 at 30 fps. [17] presents a more recent FPGA implementation of a real-time stereo vision system, which generates dense disparity maps based on the Census transform. Lastly, the hardware implementation presented in [18] performs a modified version of the Census transform in both the intensity and the gradient images, in combination with the SAD correlation metric (SAD-IGMCT algorithm), achieving 60 fps on 750x400 images.

The dedicated hardware implementations in [14]-[18] indicate that the disparity map estimation can be effectively performed in real-time with high frame rates. However, many applications of disparity estimation require not only real-time processing but also accurate depth information. Unfortunately, the majority of the aforementioned implementations struggle to provide accurate depth information, as they mostly implement fast fixed-support [8]-[9], [11]-[12], [14]-[18] and multiple-window algorithms [13], which have difficulty in determining the best window size and shape for each pixel during the cost aggregation step [4]. As such, these implementations are prone to errors and tend to generate incorrect matches especially at depth discontinuities [5]. To the best of our knowledge, only the work in [19] implements a local ADSW stereo correspondence algorithm. That work proposes an accurate, hardware-friendly disparity estimation algorithm called mini-census ADSW, and its corresponding real-time VLSI architecture that achieves 42 fps on 352x288 image sizes.

III. Segmentation-based Adaptive Support Weight Algorithm –Hardware Implementation

The proposed stereo correspondence architecture is inspired by the ADSW algorithm proposed in [5]. This algorithm utilizes segmentation information within the weight cost function in order to increase the robustness of the matching process. The proposed architecture implements a modified version of the original algorithm, by integrating hardware-directed optimization techniques that aim to make the algorithm hardware-friendly. Additionally, an edge detection process is integrated, in an attempt to reduce the search space over which the matching process will be executed.

A. Algorithm Overview

The algorithm's steps towards computing a disparity value for each pixel p in I_r are given in Fig. 1. The algorithm starts by determining whether a pixel p corresponds to an edge or not. For each pixel p corresponding to an edge, the algorithm extracts an *mxm* support window W_r centered on p in I_r, and an *mxm* support window W_t centered on q in I_t (the coordinate of q is $(x+d, y)$, where d lies in the range d_m to d_M). The algorithm then applies a segmentation process over W_r and W_t, and uses the information obtained by the segmentation in the weight generation step, which generates the weight coefficients w'_r and w'_t (for each pixel falling in W_r and W_t) as in (1), where S_c is the segment where the central point p_c (or q_c) of the support window W_r (or W_t) lies, d_c is the Euclidean distance between two triplets in the CIELAB color space, and γ_c is a parameter of the algorithm. The next step is to compute a pointwise score for any pixel $p_i \in W_r$ corresponding to $q_i \in W_t$. The pointwise scores, which are selected as the Absolute Difference (AD) of p_i and q_i, are then weighted by a coefficient $w'_r(p_i, p_c)$ and a coefficient $w'_t(q_i, q_c)$. The final aggregated cost is computed by summing up all the weighted pointwise scores, and

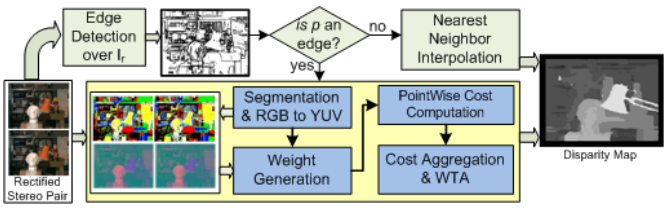

Figure 1. Overall flow of the algorithm implemented by the proposed disparity estimation system.

normalizing by the weights sum as in (2). The segmentation, weight generation, pointwise score computation and cost aggregation steps are executed for all disparity levels and the best disparity for the pixel p is found by locating the disparity with the minimum aggregated cost through a winner-takes-all (WTA) approach. In the case where pixel p does not correspond to an edge, the disparity is obtained by a simple nearest neighbor interpolation step.

$$w'_{r,t} = \begin{cases} 1.0 & p_i \epsilon S_c \\ exp\left(-\frac{d_c\left(I_{r,t}(p_i), I_{r,t}(p_c)\right)}{\gamma_c}\right) & otherwise \end{cases} \quad (1)$$

$$C(p_c, p_i) = \frac{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i) \cdot AD(p_i, q_i)}{\sum_{p_i \epsilon W_r, q_i \epsilon W_t} w'_r(p_c, p_i) \cdot w'_t(q_c, q_i)} \quad (2)$$

B. Hardware-Directed Optimization Techniques

The segment-based ADSW algorithm in [5] uses mean shift segmentation. However, the computational complexity and memory requirements make it unsuitable for embedded real-time applications. The k-means algorithm is simpler, and can be implemented in hardware more efficiently. In this work, we integrate an even simpler method, which partitions the image into segments using thresholding; we adopt this instead of the k-means, based on our observation that it has no negative impact on the overall disparity map accuracy. We also adopt YUV instead of CIELAB color representation in the weight generation step. This allows the use of unsigned integers instead of signed floating-point integers, which are complex and hardware-unfriendly. We also adopt Manhattan rather than Euclidean distance during the computation of the color distance between two YUV triplets. In this way, the square and square root operations are replaced by simple absolute difference and addition operations. Furthermore, the *exp(-x)* function is approximated by the $\lfloor 2^{8-x} \rfloor$ function, which assigns a maximum weight of 256 if the color distance is zero and a weight of 0 if the color distance is greater than 8. This function simplifies the circuits that implement the multiplication of the weight coefficients with the pointwise scores, as multiplications are reduced to left shift operations. The cost function is further simplified by setting γ_c to 16 instead of 22 (the value used in [5]). This converts the division to a right shift operation. Lastly, the denominator of (2) is approximated by the nearest power of 2 during the cost aggregation step, allowing the division to be replaced by a right shift operation.

We illustrate how the accuracy of the disparity maps is impacted by the aforementioned optimization techniques in Table I. We compare the error rate averaged over the Tsukuba, Venus, Cones and Teddy stereo images from Middlebury evaluation website [2] to a reference algorithm that works similarly with the one proposed in [5], but using the k-means instead of the mean shift segmentation. Moreover, the reference algorithm does not integrate any post-processing steps (for refining the disparity maps), as these are part of ongoing work. We observe that the overall error rate is reduced by ~2.9% after the integration of all optimization techniques.

TABLE I. IMPACT OF OPTIMIZATION TECHNIQUES ON ERROR RATE

Optimizations	1	1-2	1-3	1-4	1-5	1-6	1-7
Error rate	0%	-2.3%	-0.8%	-2.9%	-3.4%	-2.6%	-2.9%

1 - Thresholding segmentation, 2 - YUV color representation, 3 – Manhattan color distance, 4 - $\lfloor 2^{8-x} \rfloor$ function, 5 – $\gamma_c = 16$, 6 – approximation of the denominator by the nearest power of 2, 7 – edge detector

IV. PROPOSED HARDWARE ARCHITECTURE

The proposed architecture consists of two major pipeline stages: the *Input Stage (IS)* and the *Calculation Stage (CS)*. The IS fetches pixel values from the input images in RGB format and performs pixel-based operations on them, such as RGB to grayscale conversion, RGB to YUV color conversion and segmentation. The image values computed by the IS are temporarily stored into on-chip buffers (memory arrangements - *MAs*), ensuring that there is always sufficient data for the CS, which is responsible for the calculation of the disparities. The overall system architecture also consists of a control unit that coordinates data transfers and handshakes between the different system units. Fig. 2 shows a block diagram of the proposed architecture and the data flow between units.

A. Input Stage (IS)

The IS consists of a memory controller, 2 RGB to YUV color space converters (rgb2yuv), 2 RGB to grayscale converters (rgb2gray) and 2 segmentation units. The memory controller interfaces to an external memory and fetches the RGB color values corresponding to the support windows W_r and W_t in a column-wise fashion. Those values are then converted to grayscale by the two rgb2gray units, and to their corresponding 8-bit YUV representation by the rgb2yuv units. The image segmentation units receive the grayscale values and the number of segments k given as input to the system (maximum supported k is 32). A label (an unsigned integer in the range 1 to k) computed by a simple method that multiplies the input grayscale value by the value of k/256 is assigned to each input grayscale value. The multiplication is performed using fixed-point arithmetic with 8-bits of integer and 16-bits of fraction. The values of k/256 for all possible values of k are stored in a look-up table (LUT). The result is given by taking the 5 most significant bits of the multiplication operation.

B. On-chip Memory Arrangements (MAs)

The on-chip MAs temporarily store the pixels required to perform correlation between W_r in I_r and the d_M candidate support windows W_t in I_t. The system is provided with 5 MAs per input image, which store the Y, U, V color values, the grayscale values and the segments. Fig. 3 (a-b) shows the architectures of the on-chip MAs for the reference and target image, respectively. Both MAs consist of a column buffer, a

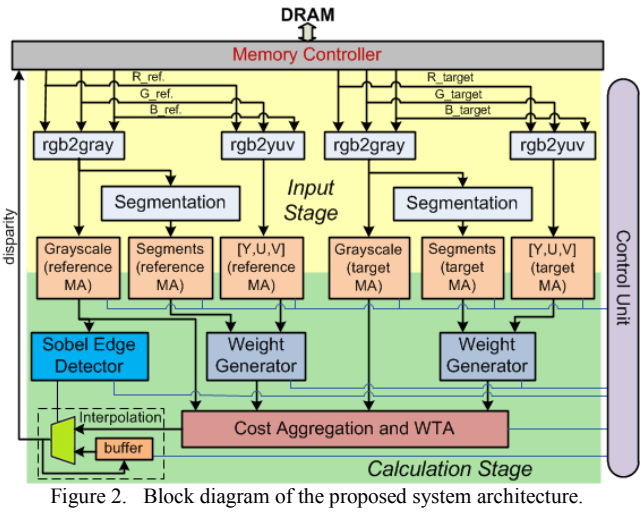

Figure 2. Block diagram of the proposed system architecture.

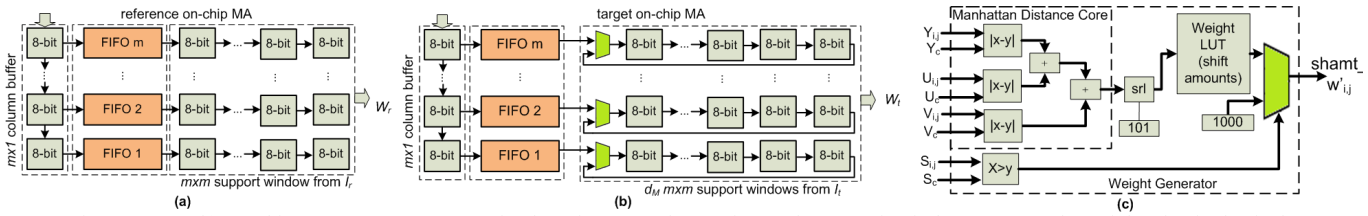

Figure 3. (a)-(b) On-chip memory arrangements for the reference and target image data, (c) Circuit that computes the weight of a single pixel.

series of FIFO queues and window buffers. The column buffer, which consists of m 8-bit registers, stores the pixels of an entire column of a support window. It receives one pixel per clock cycle and outputs a column every m cycles. The output column is stored in a series of m FIFO queues (1 pixel per queue), which are used to allow the memory controller to continuously fetch data from the external memory to the on-chip MAs (given that there is free space in the queues) irrespective of the data consumption rate of the CS. The CS consumes data at irregular rates; it consumes a column in d_M cycles if p is an edge and in a single cycle if p is not an edge.

The data from the queues is forwarded to the window buffers, which form the inputs of the CS. The window buffer of the reference MA consists of mxm 8-bit registers, while the window buffer of the target MA consists of $m \cdot (m + d_M - 1)$ registers (d_m is set to 0). The use of registers allows parallel access to the window buffers; after an initial delay of $m \cdot (m + d_M - 1)$ cycles per scanline (dominated by the cycles needed to fill in the window buffer of the target MA), both on-chip MAs can provide an mxm window per cycle. The window buffer of the target MA is organized in a cyclic structure, and is provided with a series of multiplexers at its input, which determine whether the input comes from the FIFO queues or from the rightmost registers of the window buffer. This structure is adopted to enable data reuse.

C. Calculation Stage (CS)

The CS consists of an edge detection unit, two units for the generation of the weight coefficients (weight generators), a unit that computes the aggregated costs and selects the disparity with the minimum cost, and a unit responsible for the nearest neighbor (NN) interpolation step.

1) Edge Detection Unit

The edge detection unit implements a Sobel operator, which involves convolution of a 3x3 window (fetched from the reference MA that stores the grayscale values) using two convolution masks; the masks hold data values between -2 and 2; thus the overall convolution does not involve a multiplier. The result of the convolution is compared to a predetermined threshold. The comparison returns 1-bit pixel values that indicate whether the pixel being processed is an edge or not.

2) Weight Generator

The weight generator computes the weight coefficients $w'_{r,t}$ for a support window $W_{r,t}$ in parallel. It receives the information about the segments and the YUV color values corresponding to the support window $W_{r,t}$ from the Segments and [Y,U,V] MAs, respectively, and computes the m^2 weight coefficients using m^2 instances of the circuit shown in Fig. 3 (c). That circuit consists of a comparator, a Manhattan distance core and a weight table (LUT). Since the multiplication of the pointwise scores by the weight coefficients (cost aggregation step) is performed using shifters instead of multipliers, each location x of the LUT stores the shift amount corresponding to the weight coefficient $\lfloor 2^{8-x} \rfloor$. This shift amount is equal to the binary logarithm of $\lfloor 2^{8-x} \rfloor$, except from values of x greater than 8, for which a binary logarithm does not exist. In that special case, the corresponding entries in the LUT are set to a number, which is large enough so that the result of a shift operation by that number is equal to zero. The comparator determines whether the pixel at location (i,j) in $W_{r,t}$ lies in the same segment with the central pixel of $W_{r,t}$. The result of the comparator specifies whether the shift amount that corresponds to the weight coefficient $w'_{r,t}(i,j)$ will be assigned to the shift amount corresponding to the maximum weight (8 in our case) or whether it will be looked up in the LUT using the color distance generated by the Manhattan Distance core as index.

3) Cost Aggregation & Winner-Takes-All (WTA)

The architecture of the cost aggregator and WTA unit is shown in Fig. 4. The unit utilizes m^2 absolute different circuits that compute the pointwise scores between corresponding pixels in W_r and W_t. Those scores are then shifted by the shift amounts corresponding to the weight coefficients w'_r and w'_t using a series of left shifters (equivalent to multiplying the scores by w'_r and w'_t). The final aggregated cost is computed by summing the outputs of the left shifters using a tree adder, and then normalizing (dividing) it by the weights sum, which, before being used for division, is rounded to the nearest power of 2 by using tree comparators. This enables a cost-effective implementation of the division using a right shifter. Finally, the WTA unit selects the disparity with the minimum cost.

4) Nearest Neighbor Interpolation

CS integrates a simple and fast nearest neighbor interpolation unit, used to assign each pixel p at (x, y) not corresponding to an edge, to the disparity value of the nearest edge point (x', y) in the same scanline, where (x'< x).

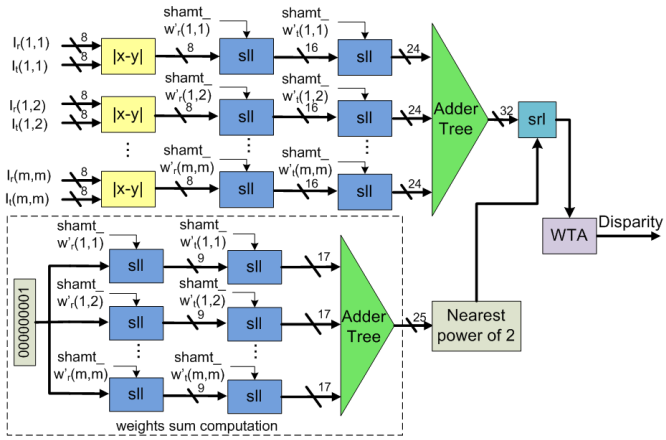

Figure 4. Architecture of the cost aggregator and WTA.

TABLE II. QUALITY COMPARISON OF DIFFERENT DEDICATED HARDWARE IMPLEMENTATIONS

	Tsukuba			Venus			Teddy			Cones		
	nonocc	**all**	**disc**	**nonocc**	**all**	**disc**	**nonocc**	**all**	**disc**	**nonocc**	**all**	**disc**
Georgoulas [14]	n.a	13.55	n.a	n.a	12.6	N/A	n.a	n.a	n.a	n.a	12.6	n.a
Darabiha [16]	19.59	n.a	37.62	10.51	n.a	31.52	n.a	n.a	n.a	n.a	n.a	n.a
Jin [17]	9.79	11.56	20.29	3.59	5.27	36.82	12.50	21.50	30.57	7.34	17.58	21.01
Ambrosch [18]	5.81	7.14	22.6	2.61	3.33	25.3	9.79	15.5	25.7	5.08	11.5	15.0
Chang [19]	n.a	2.80	n.a	n.a	0.64	n.a	n.a	13.7	n.a	n.a	10.1	n.a
Proposed	4.48	6.04	12.7	6.01	7.47	18.2	21.5	28.1	28.8	17.1	25.9	25.8

V. EXPERIMENTAL PLATFORM AND RESULTS

A. Experimental Platform

A prototype of the architecture shown in Fig. 2 was implemented on the Xilinx ML505 board, which features a Virtex-5 LX110T FPGA. The system was evaluated using rectified synthetic and real-world data, initially stored in the compact flash memory card. The synthetic data includes stereo images from the Middlebury database [2], and the real-world data includes stereo images taken in the lab. The images were loaded into the on-board DRAM using the Microblaze soft-processor, and were used as input to the system shown in Fig. 2. The resulting disparity maps were directed to a TFT monitor. The evaluation results of the synthetic images are shown in Fig. 5 (a) and Fig. 5 (c) (column 3). The evaluation results of the real-world images are shown in Fig. 5 (c) (columns 1 & 2).

B. Disparity Map Quality Analysis

The quality of the generated disparity maps was evaluated quantitatively using Middlebury stereo pairs, and by measuring the incorrect disparity estimates using the percentage of bad pixels, a commonly accepted metric [2]. Results are given in Table II, which also compares the quality of our implementation to other existing hardware implementations. We omit results from [8]-[9], [11]-[13], [15] because they do not present quality results or because a different quality metric is used. For qualitative comparisons, we include the disparity maps generated by different implementations in Fig. 5 (b).

The proposed implementation yields better quality than the ones listed in Table II when considering the Tsukuba image pair, excluding the implementation in [19]. This can also be observed in Fig. 5 (b) and particularly at depth discontinuity regions (indicated with green), and at regions with repetitive patterns (indicated with yellow). Moreover, the percentage of bad pixels at depth discontinuity regions is lower than all other implementations when considering the Venus stereo image pair. In the Teddy and Cones images, the proposed implementation yields a comparable quality to [14] and [17], but a lower quality when compared to [18] and [19]. It must be noted that the proposed implementation, along with [19], are the only ones that implement an ADSW algorithm. However, [19] does not provide quantitative results for the error rate at depth discontinuity regions. These regions are significantly important when implementing ADSW algorithms, as the idea of an adaptive support window is primarily motivated by the need to accurately detect depth borders (where depth discontinuities occur). As such, the effectiveness of the implementation in [19] cannot be directly compared to the proposed implementation. In addition, [19] focuses only on synthetic image data; in contrast, we provide evaluation results for real-world data in Fig. 5 (c), evidencing the effectiveness of the proposed implementation. We anticipate that the proposed architecture can achieve even better quality by integrating post-processing steps such as left-to-right consistency check, sub-pixel estimation, spike removal and interpolation of the occluded regions [2], [17]; the majority of the implementations listed in Table II already implement these steps.

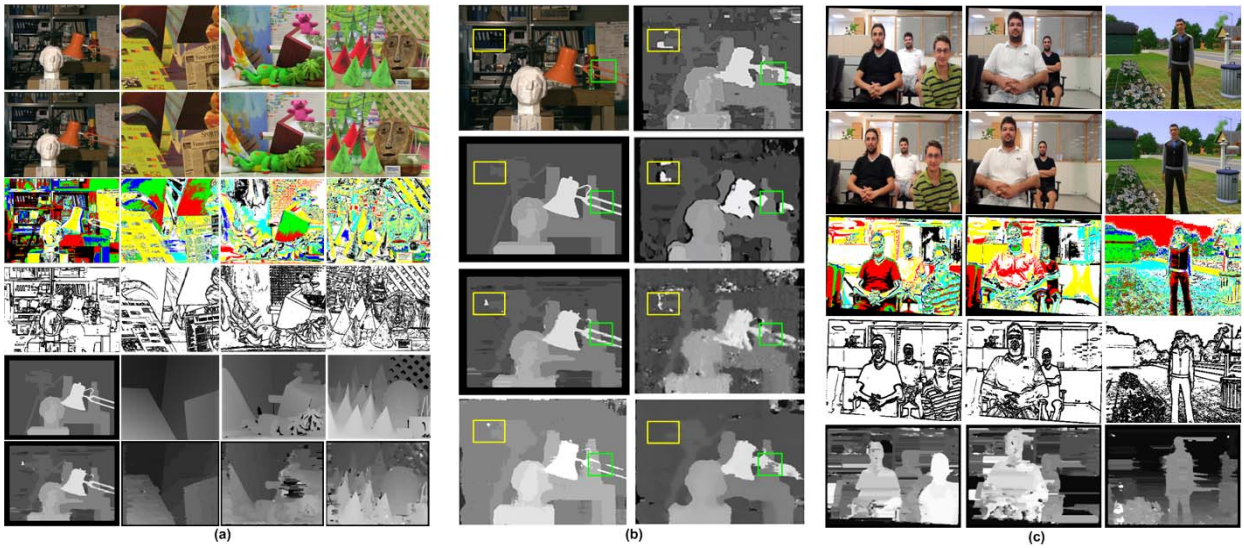

Figure 5.(a) Evaluation results using Middlebury stereo pairs for a correlation window size of 13x13 (from left to right: Tsukuba, Venus, Teddy and Cones). From top to down: reference image, target image, output of the segmentation step for the reference image, output of the edge detector for the reference image, ground truth, disparity map of the proposed FPGA implementation (b) Disparity map of the Tsukuba image pair for different implementations. From top to down: reference stereo image, ground truth, proposed, Chang et. al., Georgoulas et. al., Jin et. al., Darabiha et. al., and Ambrosch et. al., (c) Evaluation results for real-world images (columns 1 & 2) and a computer-generated image (column 3).

TABLE III. PROCESSING TIME (FPS) VS. DISPARITY RANGE
(Image Size = 320x240, Support Window Size = 13x13)

Disparity Range	8	16	32	64	128
fps w/ Edge Detector	590	361	207	115	66
fps w/o Edge Detector	239	124	65	34	19

TABLE IV. PROCESSING TIME (FPS) VS. IMAGE SIZE
(Max Disparity Range = 64, Support Window Size = 13x13)

Image Size	100x100	240x160	320x240	640x480	800x600
fps w/ Edge Detector	811	227	115	30	19
fps w/o Edge Detector	336	71	34	8	5

C. Processing Speed

We measured the processing speed (in fps) of the proposed hardware implementation using synthetic stereo image pairs from [2] as benchmarks. Tables III & IV illustrate how the frame rate is affected by the maximum disparity range and image size. We also give the frame rate of the system without the edge detector. The frame rate decreases as the disparity range and the image size increase. In both cases, the edge detector offers significant speedup (2.2 – 3.5) due to its ability to reduce the search data of the input images by an average of 55-85%, depending on the threshold used by the detector.

Table V presents a comparison between existing implementations and the proposed architecture. Performance is provided in frames per second (fps), as well as in Points x Disparity Estimates per second (PDS). The proposed system achieves 30 fps for an image size of 640x480 and a disparity range of 64. Such rates seem sufficient considering that the proposed architecture implements one of the most complex and accurate local stereo correspondence algorithms available in literature [20]. The majority of works listed in Table V implement simple and fast fixed support and multiple window algorithms; only [19] implements an ADSW algorithm. Even though the complexity of the algorithm implemented by the proposed architecture is twice as large as the complexity of the algorithm in [19] (as shown in [20]), the proposed system outperforms [19]. This is attributed to the reduction of the search space due to the integration of the edge detector.

D. Hardware Overheads

The proposed FPGA system was evaluated for relevant metrics such as area and frequency. Table VI gives the overall hardware demands of the FPGA prototype, which supports a window size of 13x13 and 64 disparity range levels. The system utilizes ~90% of the FPGA LUTs and ~80% of the FPGA slice registers, and can operate at 155MHz. The slice LUTs are dominated by the cost aggregator and the weight generators, which consume ~64% of the available LUTs. The slice registers are dominated by the on-chip MAs, which consume ~35% of the available slice registers.

TABLE V. PROCESSING SPEED COMPARISON FOR VARIOUS SYSTEMS

Work	Image size	Disparity Range	Frame rate (fps)	PDS (10^6)	Freq. (MHz)
Hile [8]	512x480	32	30	235.9	n.a.
Miyajima [9]	640x480	80	26	639	40
***Arias-Estrada* [11]**	320x240	16	71	87.2	66
Lee [12]	640x480	64	30	589	10
Hariyama [13]	64x64	64	5063	1327.2	86
Georgoulas [14]	800x600	80	550	21120	511
Ambrosch [15]	450x375	100	600	10125	110
Darabiha [16]	256x360	20	30.3	55.2	n.a.
Jin [17]	640x480	64	230	4522	93.1
Ambrosch [18]	750x400	60	60	1080	133
Chang [19]	352x288	64	42	272.5	n.a.
Proposed	640x480	64	30	589	155

TABLE VI. COMPLETE SYSTEM HARDWARE OVERHEADS
(Image Size=640x480, Max Disparity Range=64, Support Window Size=13x13)

Platform	Slice LUTs (69120)	Slice Registers (69120)	DSP48Es (64)	BRAMs (140)	Freq. (MHz)
Virtex-5 LX110T FPGA	62213 (~90.01%)	55594 (~80.43%)	16 (~25.0%)	30 (~20.3%)	155

VI. CONCLUSSION

This paper presented the hardware implementation of a segmentation-based ADSW disparity estimation algorithm. The presented system provides real-time performance and high disparity map quality when compared to other implementations featuring fixed support stereo correspondence algorithms. We plan to extend the proposed architecture to achieve better quality, by integrating post-processing steps and by combining the SAD with other correlation metrics (e.g. Cencus).

REFERENCES

[1] B. Cyganek, J. P. Siebert, *Introduction to 3D Computer Vision Techniques and Algorithms*, Wiley, John & Sons, 2009.

[2] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Inter. J. of Comput. Vision*, vol. 47, pp. 7–42, 2002.

[3] M. Z. Brown, D. Burschka, G. D. Hager, and S. Member, "Advances in computational stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, pp. 993–1008, 2003.

[4] K.-J. Yoon and I.-S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, Apr. 2006.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," *in Lecture Notes in Computer Science,* vol. 4872, Berlin, Germany: Springer, Dec. 2007, pp. 427–438.

[6] S. Hadjitheophanous, C. Ttofis, A. S. Georghiades, T. Theocharides, "Towards hardware stereoscopic 3D reconstruction: A real-time FPGA computation of the disparity map," *Design, Automation & Test in Europe Conference & Exhibition 2010 (DATE'10)*, pp.1743-1748, Dresden, Germany, 8-12 March 2010.

[7] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," *in Proc. 3rd Canadian Conf. Comput. Robot Vision*, Jun. 2006, p. 66.

[8] H. Hile, C. Zheng, Stereo Video Processing for Depth Map, Technical Report, University of Washington, 2004.

[9] Y. Miyajima and T. Maruyama, "A Real-Time Stereo Vision System with FPGA," *13th Int. Conf. on Field-Programmable Logic and Applications*, Lisbon, Portugal, 2003, pp. 448-457.

[10] L. Nalpantidis, G. Sirakoulis, A. Gasteratos, "Review of Stereo Matching Algorithms for 3D Vision,"*Proc. of the 16th Int. Symposium on Measurement and Control in Robotics (ISMCR 2007)*, Poland, 21-23 June 2007, pp. 116-124.

[11] M. Arias-Estrada, J. M. Xicotencatl, "Multiple Stereo Matching Using an Extended Architecture", *Field-Programable Logic and Applications*, vol. 2778, Springer Berlin/Heidelberg, 2003, pp. 203-212.

[12] S. H. Lee, J. Yi, J. Kim, "Real-Time Stereo Vision on a Reconfigurable System," *Embedded Computer Systems: Architectures, Modeling and Simulation*, vol. 3553, Springer Berlin/Heidelberg, 2005, pp. 299–307.

[13] M. Hariyama, Y. Kobayashi, H. Sasaki, M. Kameyama, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture, *48th Midwest Symp. on Circuits and Syst.*, vol. 2, Covington, KY, 2005, pp. 1219 –1222.

[14] C. Georgoulas, I. Andreadis, "A Real-Time Occlusion Aware Hardware Structure for Disparity Map Computation," *Image Analysis and Process. – ICIAP 2009*, vol. 5716/2009, pp. 721-730.

[15] K. Ambrosch, M. Humenberger, W. Kubinger, A. Steininger, "A SAD-based Stereo Matching Using FPGAs," *Embedded Computer Vision part II*, pp. 121-138, Spinger, London (2009).

[16] A. Darabiha, J. MacLean, J. Rose, "Reconfigurable hardware implementation of a phase-correlation stereo algorithm," *Machine Vision Appl.*, vol. 17, no. 2, pp. 116–132, 2006.

[17] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S-K. Park, M. Kim, J. W. Jeon, "FPGA Design and Implementation of a Real-Time Stereo Vision System," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 15 - 26, Jan. 2010.

[18] K. Ambrosch, W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, pp. 1303-1316, 2010.

[19] N.Y.-C. Chang, T-H Tsai, B-H Hsu, Y-C Chen, T-S Chang, "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," *IEEE Trans. Circuits Syst. Video Technol.*, vol.20, no.6, pp.792-805, June 2010.

[20] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," *IEEE Int. Conf. Comput. Vision Pattern Recognit.*, Jun.2008, pp. 1–8.