Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2014, Article ID 945926, 12 pages
http://dx.doi.org/10.1155/2014/945926

Research Article

Hardware-Efficient Design of Real-Time Profile Shape Matching Stereo Vision Algorithm on FPGA

Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James K. Archibald

Department of Computer Engineering, Brigham Young University, Provo, UT 84602, USA

Correspondence should be addressed to Beau Tippetts; beau@smartvisionworks.com

Received 7 October 2013; Revised 30 December 2013; Accepted 4 January 2014; Published 24 February 2014

Academic Editor: John Kalomiros

A variety of platforms, such as micro-unmanned vehicles, are limited in the amount of computational hardware they can support due to weight and power constraints. An efficient stereo vision algorithm implemented on an FPGA would be able to minimize payload and power consumption in microunmanned vehicles, while providing 3D information and still leaving computational resources available for other processing tasks. This work presents a hardware design of the efficient profile shape matching stereo vision algorithm. Hardware resource usage is presented for the targeted micro-UV platform, Helio-copter, that uses the Xilinx Virtex 4 FX60 FPGA. Less than a fifth of the resources on this FGPA were used to produce dense disparity maps for image sizes up to 450×375, with the ability to scale up easily by increasing BRAM usage. A comparison is given of accuracy, speed performance, and resource usage of a census transform-based stereo vision FPGA implementation by Jin et al. Results show that the profile shape matching algorithm is an efficient real-time stereo vision algorithm for hardware implementation for resource limited systems such as microunmanned vehicles.

1. Introduction

A variety of applications can benefit from dense disparity, robust stereo vision algorithms performed in real time on small, low-power processing systems. For example, microunmanned ground and aerial vehicles are required to carry sufficient batteries to power both movement and computation hardware but need to detect and avoid objects in its environment. Such a restriction limits the amount of computational resources that are available for processing of autonomous control and mission related sensor data, even before computationally intense stereo vision processing is considered. In addition to power restrictions, microunmanned vehicles (micro-UVs) have limited computational resources available for stereo vision processing for other possible reasons, including constraints on weight, size, and cost, or perhaps a requirement that the bulk of computing resources should be dedicated to a different primary task. Efficient use of light-weight, low-power image processing hardware to generate 3D information of their environment makes many applications for micro-UVs possible. Larger UVs have been suggested for several applications such as crop dusting, remote sensing [1], cinematography, aerial mapping [2], tracking [3], inspection [4], law enforcement, surveillance [5], search and rescue, and even exploration of the planet Mars [6]. With the ability to process stereo images in real time and produce 3D information for autonomous control, micro-UAVs would become more practical solutions than larger UAVs for many of these applications.

Many current solutions avoid the problem of computationally intense stereo vision processing by using ground stations to process images streamed wirelessly from the micro-UV. This dependency on a ground station for computation of sensors and control limits not only the operating range of a micro-UV, but also its reaction time. Transmission of images to a ground station for processing degrades their quality, causing less accurate results. As bin Ramli et al. [7] noted, on-board processing of all sensors related to control of the micro-UV increases robustness. Any vehicle that is dependent on a fixed ground station cannot be considered fully autonomous. A real-time vision sensor that meets the weight and power constraints of micro-UVs dramatically increases their functionality.

Table 1: Published FPGA implementation of stereo vision algorithms. FPGA resource usage is given, for those publications that provide it, in terms of percentage of configurable logic (%L) and percentage of memory (%M) for the specific device. The one exception is the BP implementation by Pérez et al. [28] which required a device with access to a total of 1.5 GB of memory.

Algorithm	fps	$W \times H$ (disp)	%L/%M	FPGA
SAD (FPGA) [29]	599	450 × 375 (60)	57/11	Quartus II
RTSVP [30]	303	640 × 480 (60)	34/26	Stratix II EP2S60
11 × 11 census [18]	230	640 × 480 (60)	57/95	Virtex4 XC4VLX200
SymDP (FPGA) [31]	30	1024 × 768 (128)	—/—	Stratix III
Tyzx [32]	200	512 × 480 (52)	—/—	DSP, FPGA, Tyzx
MeanCensus [33]	130	640 × 480 (60)	27/4	Stratix 1S40
7 × 7 SAD [34]	25	512 × 512 (255)	44/—	Virtex4 XC4VLX15
CensusVarCross [16]	62	640 × 480 (60)	—/—	Stratix III EP3SL150
SAD-IGMCT [17]	62	750 × 400 (60)	—/—	N/A
SAD left-right [35]	20	640 × 480 (80)	21/—	N/A
LWPC [36]	30	640 × 480 (36)	—/—	4 x Stratix S80
Trellis [37]	30	320 × 240 (128)	—/—	Virtex II pro-100
BP [28]	2	1280 × 720 (96)	12/1.5 G	Virtex 5 330VLX
Fasttrack DPML [38]	86	384 × 288 (16)	—/—	N/A
PARTS [39]	42	320 × 240 (24)	—/—	PARTS
Trinocular [40]	30	320 × 240 (32)	—/—	Virtex II XC2VP40
Adaptive census [41]	68	640 × 480 (—)	67/4	Virtex 6 VLX760

One low-power technology that offers large quantities of computational resources is field-programmable gate arrays (FPGAs). Another advantage to FPGAs is that they can be integrated into light-weight platforms. For robotic applications, this allows implementation of control and sensor algorithms that are fast and highly parallel, yet still alterable as algorithms and parameters are refined over time. Given recent advances in FPGA capacity and performance, researchers have naturally pursued hardware implementation of vision algorithms to obtain increased performance [8]. By implementing algorithms in FGPAs, more complex systems can run in real time, and image resolution can be increased, while less power and lower payload capacities are met.

An efficient stereo vision design implemented on an FPGA would minimize payload and power consumption on micro-UVs, while providing 3D information and still leaving computational resources available for other processing tasks. This work presents a hardware design of the efficient profile shape matching stereo vision algorithm [9], shown to produce disparity maps from real uncalibrated and unrectified images with accuracy sufficient for tasks useful to unmanned micro-UVs. The target platform was a micro-UV called Helio-copter [10], that has incorporated the Xilinx Virtex 4 FX60 FPGA for control and image sensor processing. The Helios board connects directly to a custom stereo camera board allowing stereo image data to be streamed directly to the FPGA in a pipelined structure. The resulting hardware design uses less than a fifth of the Virtex 4 FX60 resources, while producing real-time stereo vision disparity results at 30 fps of 450 × 375 images.

2. Background

There are several accurate stereo vision algorithms that have been implemented on FPGAs to achieve real-time performance. As can be seen in Table 1, many types of implementation have achieved real-time performance of 30 fps or more on relatively large images. Four of them are even able to process several hundred frames per second. Although there are applications that could take advantage of this, many real-time micro-UV applications are limited by the frame rate of the image sensor, which is most commonly 30 fps. An efficient FPGA design targeted at micro-UV applications would use as few resources as possible to achieve real-time performance and accuracy sufficient for desired tasks. Efficient types of stereo vision implementation have been achieved on other hardware such as the GPUs, for example, [11–14], but the power and size limitations of mirco-UVs make GPUs an impractical resource.

Only three types of the FPGA stereo vision implementation in Table 1 provided sufficient accuracy measures for a common comparison of accuracy. Using an accuracy measure based on Scharstein and Szeliski's bad pixel error [15], we can compare some of types of the FPGA stereo vision algorithm implementation. The average percent of all correct pixels on the four Middlebury stereo vision datasets is used, which is the complement ($100 - \text{error}_{\text{all}}$) of Schartstein and Szeliski's average error for all pixels. The census variable cross algorithm CensusVarCross [16] from Zhang et al. obtained an accuracy of 92.6% average correct pixels, while Ambrosch and Kubinger's SAD and gradient census algorithm SAD-IGMCT [17] achieved 90.6%, and the census-based algorithm 11 × 11 census [18] by Jin et al. achieved 86% accuracy. Since CensusVarCross [16] and SAD-IGMCT [17] have the same runtime, the extra accuracy of CensusVarCross [16] is an advantage, while a tradeoff of about 6 percentage points of accuracy can be made for nearly a 4X speed-up by going to 11 × 11 census [18]. Jin et al. provided accuracy and speed performance measurements as well as hardware resource usage, and therefore it is included for comparison in the results that will be presented in Section 5.

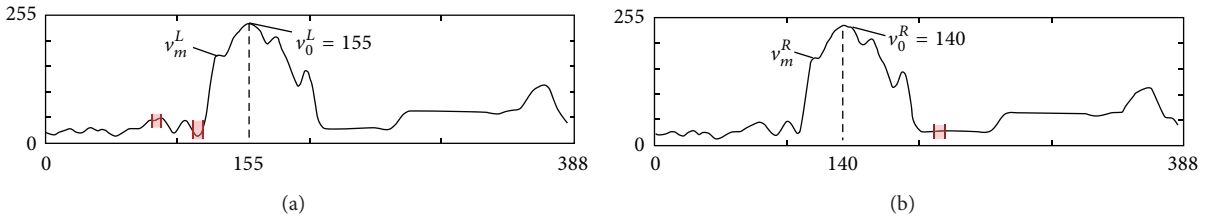

FIGURE 1: Gaussian filtered intensity profiles of corresponding rows from the left (a) and right (b) Tsukuba images, with two labeled shape vertices, and highlighted regions that represent discontinuities.

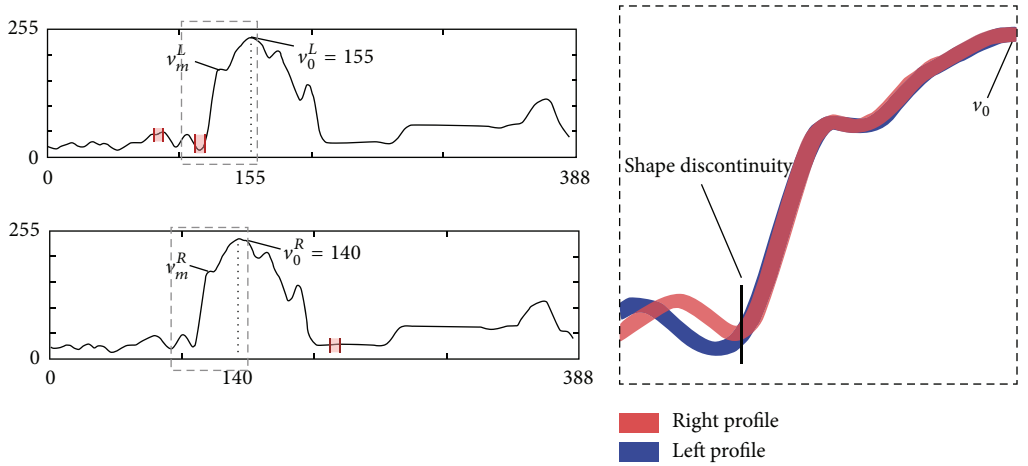

FIGURE 2: Enlargement of an actual section of the two intensity profiles from the Tsukuba images overlapped to emphasize the shape discontinuity and one of the missing highlighted sections of Figure 1.

3. Intensity Profile Shape Matching Algorithm

The hardware design presented in this paper is an implementation of the profile shape matching stereo vision algorithm [9]. The concept of this algorithm is based on observations of the intensity profile of an image. Looking at the intensity profile of a stereo image pair, one can see the similarities in shapes between the two images. For example, Figure 1 shows the Gaussian filtered intensity profile of corresponding rows from the left and right Tsukuba images. Looking at the entirety of the profiles allows one to recognize the similar shapes between the two images. Closer inspection reveals discontinuities in the shapes along the profile, where one of the profiles is missing a section that exists in the other profile. Examples of this are highlighted in red in Figure 1 and overlapped enlargement of one of the examples is shown in Figure 2. These discontinuities correspond to changes in the disparity map. This algorithm assumes that all pixels included in a shape, bounded by discontinuities, correspond to the same depth plane. This is based on similar assumptions in [19–22]. It also assumes that each shape does not violate the uniqueness constraint of matching only one shape in the other image. The algorithm consists of essentially three steps.

(1) Filter the input image pair using a Gaussian kernel.

(2) Match and identify shapes on a row-by-row basis. At the completion of this step, every pixel is part of a shape and has a disparity value.

(3) After the entire image is processed in this manner, perform a vertical smoothing pass to reduce streaking.

A Gaussian filter is a fairly common preprocessing step to reduce noise. It is used as part of this algorithm to smooth the profile shape, increasing shape matching accuracy.

Step two consists of matching shapes and finding the range of pixel indices, $[p, q]$, that make up each shape. This is summarized in Algorithm 1. First, vertices of the intensity profile shapes are found in both the left and the right images. Shape vertices are defined as local maxima in the intensity profile. The ordered set of vertices in a single row of an image is called V^k where $v_m^k \in V^k$. The value of v_m^k is the image row pixel index of the mth vertex, and k is either L (left image) or R (right image). The gradient of successive pixels, ∇^k, is then computed for the current row in both images according to

$$\nabla^k [x] = I^k [x + 1] - I^k [x] . \quad (1)$$

The algorithm starts at the vertex of the most prominent shape in the left image and compares it against all the shape vertices in the right image that are within the maximum disparity range, $d_{\max}$. The shape prominence is defined in terms of the vertex intensity value: the larger the intensity value is, the more prominent the shape is. Two vertices, v^L and v^R, are compared for matching by growing a region around each vertex. The region is grown until the difference of

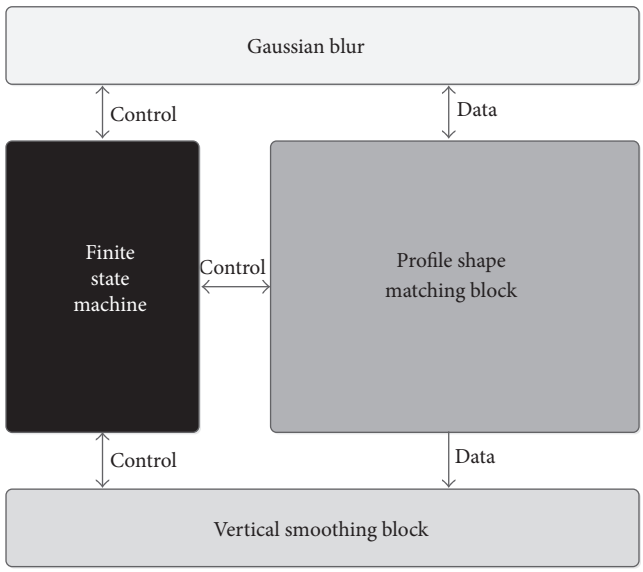

Figure 3: Profile shape matching high-level conceptual hardware block diagram.

gradients (1) is greater than or equal to the initial threshold, t_0, according to (2). After comparing all the vertices within $d_{\max}$, the two vertices that share the largest range of pixels are then labeled as the same shape, $S_n(v_n^L, p_0^L, q_0^L, v_m^R, p_0^R, q_0^R)$, where n is the shape label and p_0 and q_0 are the starting and ending pixel index values for each image, respectively. The disparity value of shape n, as well as all pixels included in it, is $d_n = v^L - v^R$. Consider the following:

$$S_n\left(v_n^L, p_i^L, q_i^L, v_m^R, p_i^R, q_i^R\right) = \left[p_i^k, q_i^k\right]$$
$$\text{s.t.} \quad \forall x : p_i \le x \le q_i, \quad \left|\nabla^R[x] - \nabla^L[x + d_n]\right| < t_i. \tag{2}$$

Once this first iteration is complete, each shape vertex will have been uniquely labeled and matched with a shape vertex in the other image. Due to a low initial threshold, however, most of the pixels of the current image row will not have been included as part of one of the shapes. Successive iterations are performed using an increased threshold, t_i, to expand the range $[p, q]$ of each shape until all pixels are included in a shape.

The use of the intensity gradient allows the algorithm to rely on intensity profile shape instead of raw intensity values like SAD, SSD, and other block matching algorithms. Focusing on the shape or gradient, versus intensity values, allows the algorithm to still provide a correct disparity map if one of the input images contained a constant offset of intensity values. Growing shapes incrementally allows pixels to be grouped with the most probable shape.

Once the shapes are matched, they do not continue to be compared to other shapes within the disparity range in subsequent iterations. Moreover, pixels that have been included in a shape are not reconsidered. These two characteristics significantly reduce the number of computations that the algorithm performs.

The final step of the intensity profile shape matching algorithm is a vertical smoothing pass. Since the shape matching is performed on a row-by-row basis, vertical information is not inherently leveraged. To smooth every pixel in the image, the 5 pixels above and below it are accumulated according to the disparity value. The pixel is then modified to take the disparity value with the greatest number of votes. This simple 11-pixel voting scheme produces the final disparity map.

4. Profile Shape Matching Hardware Design

Figure 3 shows a high level conceptual block diagram of the profile shape matching hardware design. The algorithm was partitioned into three main steps, as described in Section 3: a Gaussian blur step, the actual profile shape matching step, and a vertical smoothing step. The hardware design uses a finite state machine (FSM) to control the flow of data through these blocks. The design was intended to be an inline processing design where the pixel data is processed in a pipelined manner. This allows the output disparity data to be available at the same clock rate as the input pixel data coming out of the camera with the delay of a fixed latency. At the outset of the design process, the goal was to use only as much parallelization as was necessary to maintain this inline processing characteristic. Although more parallelization would allow for faster frame rates, as has been mentioned, the goal was to provide a design suitable for low-resource systems. This resulted in efforts to reduce the required FPGA resources as much as possible while still maintaining real-time performance of 30 fps. Each step of the design was tested to verify that a clock frequency of 50 MHz could be used, which allows images of up to 640 by 480 pixels to be processed.

4.1. Gaussian Blur Block. The only modification made to the original algorithm to reduce resources and processing latency was to reduce the Gaussian blur from two dimensions to just one. In the software version, the Gaussian blur was already separated and implemented in two 1D passes, but this modification removes the second pass. This reduced the amount of buffers required to store eleven rows of the image for a vertical pass, to just eleven registers for the horizontal pass. It also reduces the latency from eleven times the width of the image to just eleven pixel clock cycles total. This modification has minimal negative effects on the algorithm due to the 1D single row processing nature of profile shape matching, as well as the vertical smoothing step at the end. The amount of hardware saved through this modification is actually double that mentioned because a Gaussian blur block is instantiated for each of the left and right images.

The Gaussian blur block uses eleven built-in multipliers on the FPGA, which are then summed to obtain the result. These calculations are performed in two clocks cycles. These two cycles, plus those required to receive five pixels of image data for the left half of the Gaussian kernel, add 12 clock cycles of latency to the system. This block includes a counter that accounts for this latency and outputs the correct column index of the data being output.

```
for row in image do
  for vertex, v_n^L, in V_L do
    for vertex, v_m^R, in V_R where v_n^L − v_m^R ≤ d_max do
      while |∇^R[q_0^k] − ∇^L[q_0^k]| < t_0 do
        Increment q^k
      end while
      while |∇^R[p_0^k] − ∇^L[p_0^k]| < t_0 do
        Decrement p^k
      end while
    end for
    Set S_n(v_n^L, p_0^L, q_0^L, v_m^R, p_0^R, q_0^R) s.t. arg max_{v_m^R} (q_0^R − p_0^R)
  end for
  for t_i in thresholds do
    for vertex, v^R, in V^R do
      while |∇^R[q_i] − ∇^L[q_i]| < t_i do
        Increment q
      end while
      while |∇^R[p_i] − ∇^L[p_i]| < t_i do
        Decrement p
      end while
      if every pixel is assigned to a shape then
        Continue to next row of image
      end if
    end for
  end for
end for
```

Algorithm 1: Profile shape matching *step 2*.

Table 2: FPGA logic utilized by the profile shape matching hardware design capable of processing images as large as the Teddy and Cones images (450 × 375, 60 disp) on a Xilinx Virtex 4 FX60.

Logic	Used	Available	Utilization
Total number of slice registers	6,314	50,560	12.5%
Number of 4 input LUTs	9,637	50,560	19%
Number of RAMB 16 s	30	232	13%
Number of DSP 48 s	20	128	16%

4.2. Profile Shape Matching Block. After Gaussian blurring, the pixel data is then available for the profile shape matching block. Figure 4 shows the individual elements of this block. The incoming pixel data for the left and right images is written into the row buffer using the column address from the Gaussian blur block. The row buffer has two write ports so that the left and right Gaussian blurred image data can be written simultaneously and six read ports so each of the three row processing elements (RPEs) can read both left and right image data simultaneously as well. This does not actually require implementation of a single RAM with so many ports because the writing of data from the Gaussian and the reading of data by each of the RPEs are all happening on separate image rows of data at any given moment.

At the same time that the left image data is being read into the row buffer, it is also clocked into three registers that are used to find max points or shape vertices (block A in Figure 4). Another three registers and a subtractor are used to store the address of the vertices and calculate the distance to the previous vertex (block B). When a vertex is found, its address and the distance from the previous vertex are concatenated and enqueued as one value into the shift sort block. The shift sort block, which will be described in more detail in Section 4.2.2, sorts the values based on the distance between vertices. As discussed in Section 3, the distance between vertices correlates with the prominence of a shape, allowing the most prominent shapes to be matched and grown first.

At the beginning of each row the value of one is enqueued several times into the shift sort block to separate the sorted data of the current row from the previous row. While the data for the new row is being sorted, the data from the previous row is enqueued into a FIFO in one of the three RPEs. The FSM controls which of the three RPEs are active at any given time. The RPE does not read the data of a given row until it has been completely written into the row buffer. When a row has been completely written, the FSM enables the next available RPE to start processing the sorted vertices and also increments the write row address in the row buffer. Due to this structure, each row in the row buffer can be implemented with a separate BRAM in the FPGA.

4.2.1. Row Processing Element. The row processing element (RPE) is where the majority of the algorithm is implemented. Figure 5 shows the details of this block. Since the RPEs are

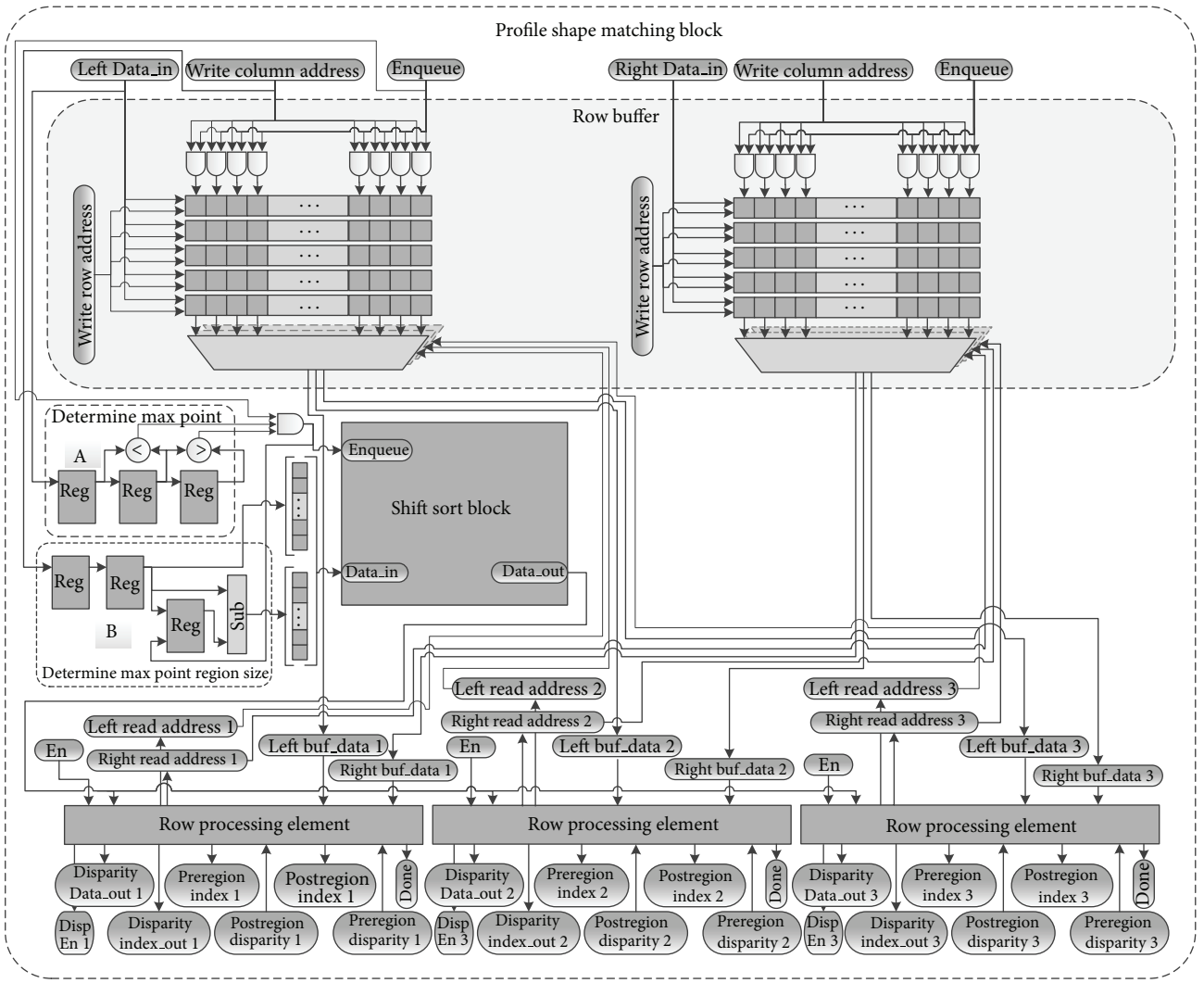

Figure 4: Profile shape matching hardware block diagram.

Table 3: Accuracy, runtime performance, and hardware resource usage of FPGA implementation of the census-based algorithm by Jin et al. [18] and the profile shape matching algorithm.

Algorithm		Nonocc.	All	Disc.	%Correct	Mde/s	Slices BRAMs
Census [18]	Tsukuba	9.79	11.6	20.3	86.02	4522	51,191 322
	Venus	3.59	5.3	37			
	Teddy	13	21.5	31			
	Cones	7.3	18	21			
Profile Shape Matching	Tsukuba	7.78	9.54	31.10	78.0	1500	4,854 30
	Venus	10.5	12.0	34.7			
	Teddy	34.0	40.8	51.6			
	Cones	16.2	25.5	35.2			

determining shape matches and the time to do so is data dependent, this step of the algorithm represented a challenge for hardware implementation. In previous work [10], the hardware architecture of a processing element was used in the cost aggregation step of a template matching implementation. It was observed that pipelining multiple passes of a specific step with varied timings can be efficiently handled with multiple processing elements controlled by a master finite state machine. This concept was successfully applied in development of the row processing elements (RPE).

Once all the vertex indices have been found and sorted for a given image row, the master FSM enables the next available

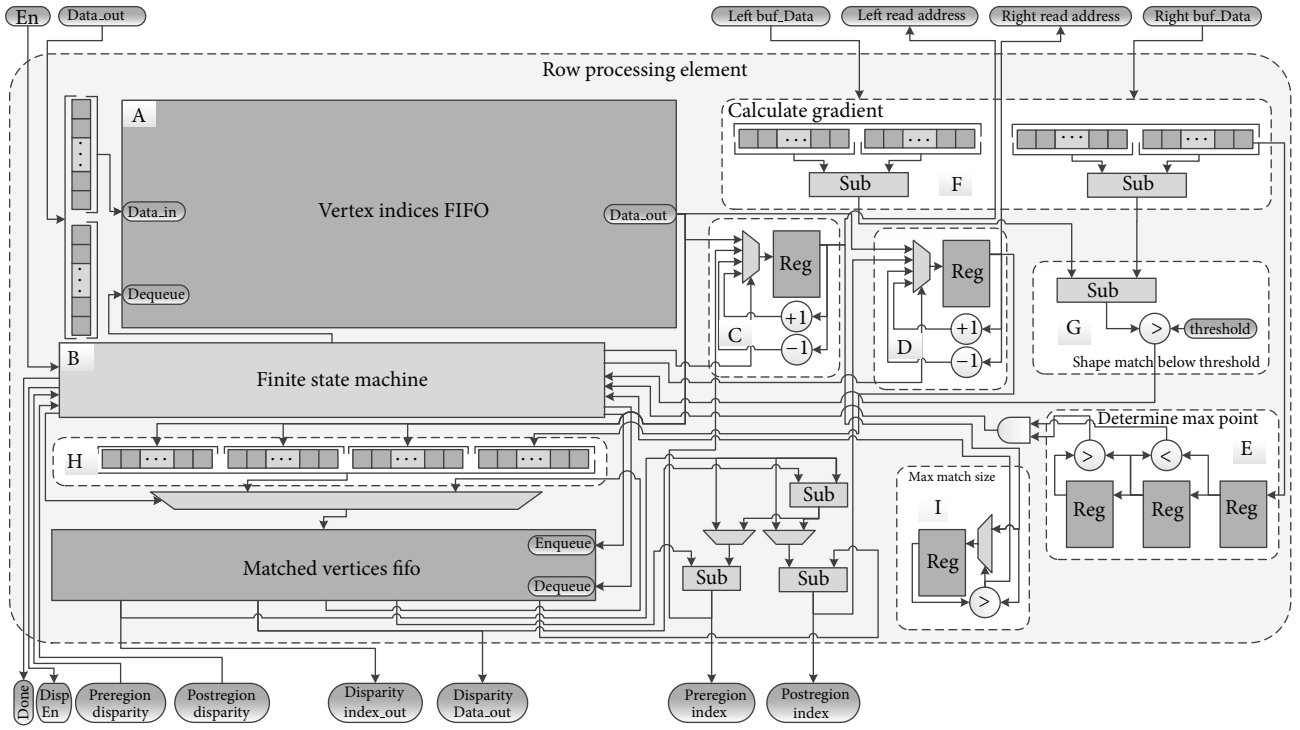

FIGURE 5: Row processing element (RPE) hardware block diagram.

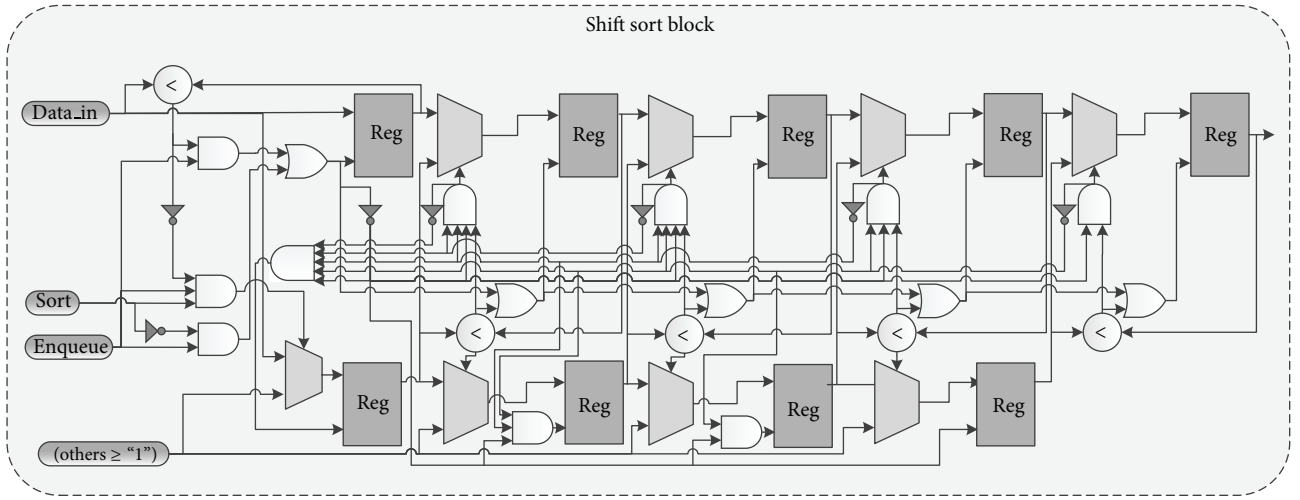

FIGURE 6: Shift sort hardware block diagram.

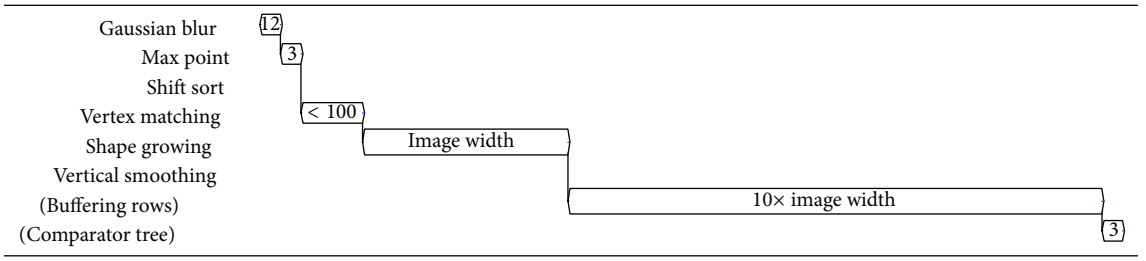

FIGURE 7: Average latency of pixel disparity results in clock cycles.

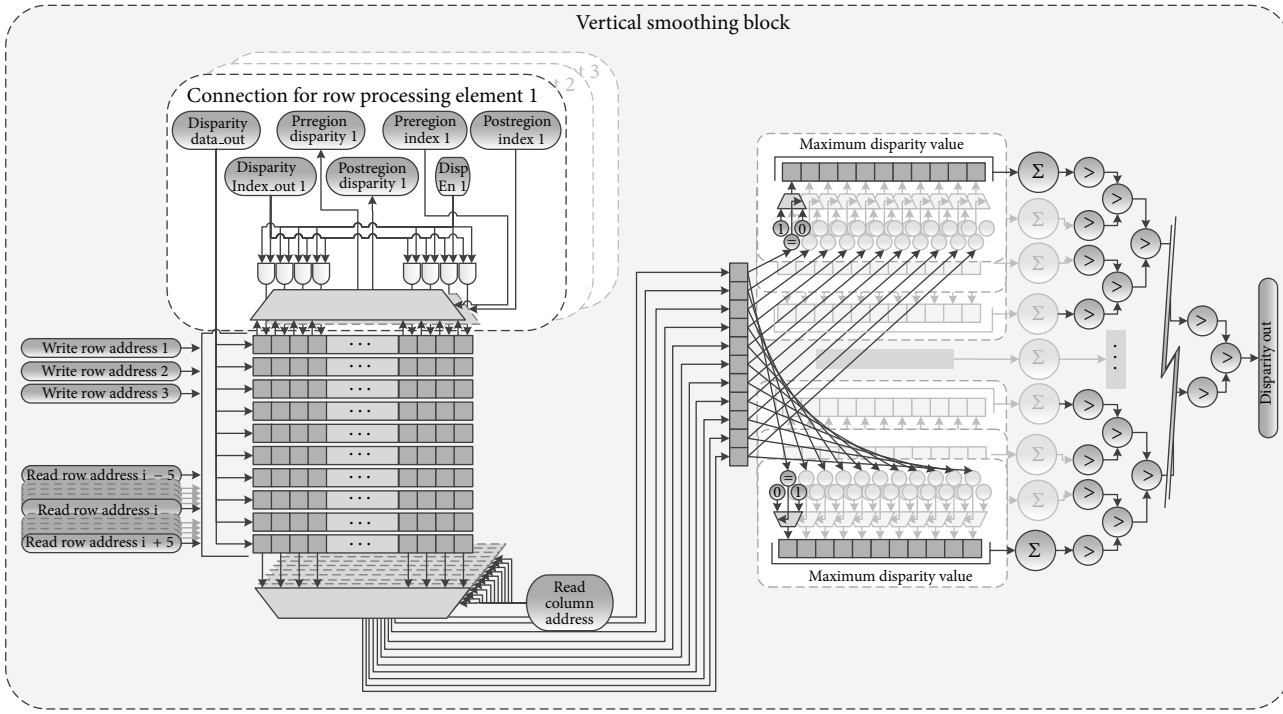

Figure 8: Vertical smoothing hardware block diagram.

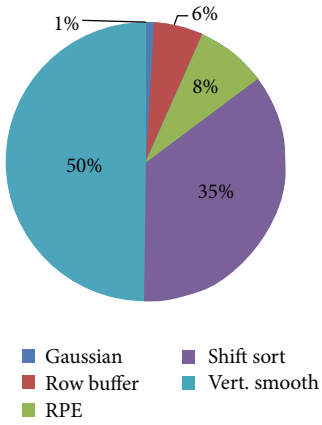

Figure 9: Approximate FPGA resource usage breakdown by block for the profile shape matching hardware design.

RPE. The enabled RPE enqueues just the index portion of the sorted vertex data into its own vertex index FIFO (block A in Figure 5). As many FIFOs as possible were implemented using the built-in FIFOs on the FPGA, because implementing them in raw fabric consumes many more resources. Each RPE then has its own internal FSM (block B) that controls the left (block C) and right (block D) column read addresses that access the row buffer. These addresses start at the same value, and the right address (block D) is decremented on each clock cycle until a vertex is detected (block E). The FSM then decrements both left and right addresses until the difference of gradients (Block F) is above the first threshold (block G). The left address is temporarily stored in a register as the left index of the shape (block H). The addresses (blocks C and D) are then reset and incremented, growing to the right now, on each clock cycle until the difference of the gradients is above the first threshold also. The stored left address and right address are then subtracted to find how much of the shapes match and allowing the max match size registers (block I) to be updated accordingly. By the end of the process, the max match size registers (block I) will have the starting and ending addresses of the shapes that have the largest matching area. After comparing the first right image vertex, the FSM then resets both addresses (blocks C and D) and decrements the right read address until the next vertex is found, at which point the process is repeated. This continues until all vertices within the disparity range have been compared to the current vertex in the left image row and the best match is in the max match size registers.

To reduce the number of FIFOs required for storing the matched vertices, maximum image sizes were assumed to allow for fixed bit lengths of pixel index addresses. The architecture of the Virtex 4 includes built-in FIFOs configurable up to 36 bits wide. The necessary data for matching vertices actually fit into this size perfectly. Using the addresses stored in the max match size registers, the disparity between the address of the left image vertex and the best matching right image vertex is found. This value (6 bits) is then concatenated with the left image vertex address (10 bits) and the starting (10 bits) and ending (10 bits) addresses of the matching shape region (Block H) and enqueued in the 36 bits matched vertices FIFO. The six bits containing the disparity value allows this design to handle disparity values up to 64. The

(a) (b) (c)

Figure 10: Disparity ground truth (a) and results from the profile shape matching (b) and census implementation by Jin et al. (c) of the Tsukuba stereo image pair.

ten bits pixel index addresses allow for images up to 1024 pixels wide. This matching process is repeated for each of the vertices in the shift sort block. Once matches have been found for each of the vertices, the FSM is incremented to the next threshold value and grows the shapes in the matched vertices FIFO.

As each shape is grown, the FSM (Block B) sends an enable signal along with the disparity value and address of each pixel to the buffers in the vertical smoothing block. The FSM also uses the starting and ending addresses of each shape to set the read address of the disparity buffers in the vertical smoothing block to know when a pixel has been assigned a disparity. While the current shape is being grown, as soon as the thresholds are met in both directions, the entry is dequeued and enqueued again into the FIFO and the next shape is then grown. When any given shape in the matched vertices FIFO cannot be grown any more it is permanently dequeued from the FIFO. The FSM continues this process until the FIFO is empty, at which point it sends a done signal to the master FSM and is ready to process a new row.

4.2.2. Shift Sort Block. Figure 6 shows the details of the shift sort block used by each RPE. The process of sorting a list of values is generally the simplest to perform in software. Several FPGA devices provide embedded processors to handle such tasks. The Virtex 4 FPGA on the Helios board has a PowerPC processor that could have been integrated into the design to perform the sorting step. This would have increased the latency of the design considerably to allow the transfer of data back and forth. Instead, a novel sorting architecture was developed to perform inline sorting. The design takes advantage of a faster system clock than pixel clock, as well as the fact that only local maximum pixel values and not all pixel values need to be sorted, to ensure that no pixel data is lost and that the local maximum pixel values are sorted in time to start processing the next image row.

The shift sort design consists of essentially two loadable shift registers, referred to as the main shift register and the pass shift register. The main shift register is the top row of registers in Figure 6 and the pass shift register is the bottom row. Conceptually, the new value to be enqueued can be added to either the first (leftmost registers in Figure 6) main register or the pass register. Each of the registers is initialized to a value of one, which represents the maximum unsigned value, and the new value is enqueued into the first main register only if it is less than the current first value in the main register. When this is the case, the control logic causes the pass register to "freeze" or maintain its current state, while the main registers all shift to the right.

On all clock cycles where a new value is not being enqueued, the pass shift register values at each stage are compared to the value in the main shift register of the following stage. If it is less, then it needs to be inserted at the current location; otherwise it will be shifted to the right in the pass shift register, so it can be compared at the next stage. If a value needs to be loaded into a given stage of the main shift register, all greater values (those to the right of the current stage in Figure 6) are shifted up and the value in the pass shift register is loaded into the main register. If more than one value in the pass shift register is at the appropriate stage to be inserted into the main shift register, only the greater one is inserted, while the lower stages of the main and pass shift registers are disabled.

Because of the need to insert values into the main shift register, this block could not be implemented using neither built-in FIFOs nor RAMB16s. Since only local max points are loaded into the shift sort registers, the main shift and pass shift registers each needs to contain no more than IMAGE_WIDTH/2 registers. Although practically the registers could be reduced in size, they were left at this maximum to avoid the slight possibility of losing data.

4.3. Vertical Smoothing Block. The vertical smoothing block, shown in Figure 8, implements the histogram vertical smoothing method described in Section 3. This block requires the buffering of several rows of disparity data, so that a histogram can be taken over each column. The values of the histogram are then compared to find the maximum value. To reduce the latency of this step and since comparators require relatively little FPGA fabric, a tree of comparators was chosen over a sorting or other linear comparison design.

Eleven BRAMs are used to implement the row buffers where disparity data from the profile shape matching block is written. The main FSM controls the read column address after eleven rows have been processed by the RPEs. As Figure 8 shows, there are eleven registers for each disparity value

in the disparity range and the eleven disparity values are used to determine whether or not to set the corresponding bit for each disparity value. Each of these eleven bits is summed to give the total bin count for each disparity value. Each bin count is then passed through a tree structure of comparators to determine the bin with the maximum value which determines the smoothed disparity value. This tree contains $n-1$ comparators, where n is the size of the disparity range, which for this design has been limited to 64, based on the Middlebury datasets Teddy and Cones. Larger disparity ranges than this would require an increase in the number of comparators in the tree and an increased latency. This tree structure helps limit the latency of this block to three clock cycles.

4.4. Hardware Timing Analysis. The design met timing constraints for a minimum clock period of 19.797 ns, allowing a clock frequency of 50 MHz. Along with this system clock frequency, the VGA standard clock frequency of 25 MHz for 640×480 images was assumed for determining the latency and throughput of the system. Figure 7 summarizes the latency of the hardware design. The first block of the design, containing hardware for the Gaussian blur, has a latency of twelve clock cycles. There are two for the multiplication and addition of the Gaussian kernel and additional ten cycles to receive the five pixels for the first half of the kernel. The max point registers at the beginning of the profile shape matching phase add three more cycles before enqueueing data into the shift sort block. The latency of the shift sort block depends on the image content. The worst case would be an image row where every other pixel is a max point and the second-to-last max point found in a row is the most prominent shape, which would result in added latency of IMAGE_WIDTH/2 clock cycles. For example, this worst case would occur in an image of extreme salt and pepper static. Such an image would provide very little useful information for most micro-UV applications and should be ignored. Excessive latency would cause the master FSM to drop the frame. The best case would be that this step does not add any latency.

If all the vertices have been sorted by the time the last pixels of a row are read into the block, there is no added latency. Empirical tests showed that the worst case was never encountered and that the best case represents the timing of most rows in the test images. For example, the average number of vertices per row is about 30 for the Teddy and Cones images, with a max of 70 vertices. Once the vertices have been sorted, they are matched, the latency of which is determined by the number of vertices found in the right image within the disparity range of the left image vertex. For the Teddy and Cones images that have a larger disparity range, there is an average of 4 vertices that are processed. Extra clock cycles are needed to repeatedly compare the vertex from the left image with each of these vertices, but a low initial threshold keeps this in tens of cycles. To grow each shape until all pixels in the row are included in a shape requires IMAGE_WIDTH clock cycles, which must be added to cycles used to compare the vertices that ended up not being matches, to get a total latency for the RPE block. The last step like the profile shape matching phase also has a considerable latency due to the need to buffer five of the eleven rows before producing a disparity value. This together with the histogram addition and comparator operations creates a latency of $10 \times$ IMAGE_WIDTH+3 clock cycles. Even in the extreme cases, it is unlikely that the total latency would exceed $22 \times$ IMAGE_WIDTH clock cycles, but even for small images like Tsukuba thirty rows is less than a tenth of the image size, which should be an acceptable latency for most applications.

5. Results

5.1. FPGA Resource Utilization. All timing and performance results were verified using Mentor Graphics Modelsim 6.3 and resource usage was collected from Xilinx synthesis reports. As seen in Table 2, the profile shape matching hardware design uses less than fifth of the resources (combined slice registers, LUTs, RAMB 16s, DSP 48s) on Virtex 4 FX60 FPGA. The distribution of these resources among the various blocks of the design is shown in Figure 9. The vertical smoothing block with its comparator tree and histogram summing arrays and the shift sort block with its large shift registers account for the bulk of the resource usage.

The whole Helios board with the FPGA configured with even larger designs than this consumes less than 3 watts of power, making it very suitable for micro-UV platforms with their power limitations [10].

5.2. Performance. Table 3 shows a comparison of accuracy, runtime, and resource usage of the 11×11 census FPGA implementation by Jin et al. [18] and this profile shape matching FPGA implementation. Although Table 3 shows that the accuracy of the census implementation of Jin et al. is 8 percentage points of correct pixels higher than profile shape matching on the Middlebury datasets Tsukuba, Venus, Teddy, and Cones, Figures 10(c) and 10(b) show tradeoffs in detail and noise in the disparity results. For micro-UV tasks such as obstacle avoidance, consolidated, less noisy disparity maps are preferred. The 10X lower resource usage of the profile shape matching design not only reduces the power consumption of the FPGA, but also allows other vision processing tasks to be implemented. The nature of the hardware design allowed the disparity results to exactly match those achieved by the original software implementation. Future research can determine if the extra hardware resources could also be utilized to improve disparity accuracy results.

As previously mentioned, the profile shape matching hardware runs at 50 MHz with a pixel clock of 25 MHz. It is currently capable of handling stereo images with disparity ranges of up to 60 pixels. Although this means that the design has a max rate of 1500 million disparity evaluations per second (Mde/s), if more BRAMs were utilized, the design could be scaled to handle images larger than the Teddy or Cones Middlebury images which are 450×375 with a disparity range of 60. Since this design is intended for micro-UV applications, the speed is more than sufficient to obtain 30 fps real-time performance, with a latency of less than

a tenth of a frame, while still using relatively few FPGA resources.

As discussed in [9], the quality of the disparity map results is comparable to many stereo vision algorithms, and the algorithm is resilient to types of noise that is common when using real images from basic image sensors. This gives the algorithm advantages over the common SAD block matching technique, which has been implemented for many unmanned vehicle applications [23–25]. Disparity results from the Tsukuba image pair, as seen in Figure 10, are similar to those used in other micro-UV applications [26, 27].

6. Conclusion

A hardware design of the profile shape matching algorithm was presented. Hardware resource usage was presented for the targeted Helio-copter platform that uses the Xilinx Virtex 4 FX60 FPGA. Less than a fifth of the resources on this FGPA were used to produce dense disparity maps for images up to 450 × 375 and the ability to scale up easily by increasing BRAM usage. A census transform based stereo vision FPGA implementation by Jin et al. uses more than 10X as many logic and memory resources. These observations show that the profile shape matching algorithm is an efficient real-time stereo vision algorithm for hardware implementation.

Although the profile shape matching algorithm offers real-time stereo vision results to resource limited systems, future work will include research to make it more efficient through increased accuracy while maintaining similar runtime performance. This would include considerations such as using color gradients instead of grayscale for matching shapes and/or adding techniques to better handle disparity discontinuities.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] F. Archer, A. M. Shutko, T. L. Coleman, A. Haldin, E. Novichikhin, and I. Sidorov, "Introduction, overview, and status of the Microwave Autonomous Copter System (MACS)," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium Proceedings: Science for Society: Exploring and Managing a Changing Planet (IGARSS '04)*, pp. 3574–3576, Anchorage, Alaska, USA, September 2004.

[2] R. Sugiura, T. Fukagawa, N. Noguchi, K. Ishii, Y. Shibata, and K. Toriyama, "Field information system using an agricultural helicopter towards precision farming," in *Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM '03)*, vol. 2, pp. 1073–1078, July 2003.

[3] S. Rock, E. Frew, H. Jones, E. LeMaster, and B. Woodley, "Combined CDGPS and vision-based control of a small autonomous helicopter," in *Proceedings of the American Control Conference*, vol. 2, pp. 694–698, June 1998.

[4] E. Altuğ, J. P. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*, pp. 72–77, May 2002.

[5] L. Jun, X. Shaorong, G. Zhenbang, and R. Jinjun, "Subminiature unmanned surveillance aircraft and its ground control station for security," in *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics*, pp. 116–119, Kobe, Japan, June 2005.

[6] C. A. Patel, *Building a testbed for mini quadrotor unmanned aerial vehicle with protective shroud [M.S. thesis]*, Wichita State University, Department of Mechanical Engineering, 2006.

[7] M. A. bin Ramli, C. K. Wei, and G. Leng, "Design and development of an indoor UAV," in *RSAF Aerospace Technology SemInar*, 2007.

[8] W. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, p. 131, June 2005.

[9] B. J. Tippetts, D.-J. Lee, J. K. Archibald, and K. D. Lillywhite, "Dense disparity real-time stereo vision algorithm for resource-limited systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 10, pp. 1547–1555, 2011.

[10] B. J. Tippetts, D.-J. Lee, S. G. Fowers, and J. K. Archibald, "Real-time vision sensor for an autonomous hovering micro unmanned aerial vehicle," *Journal of Aerospace Computing, Information and Communication*, vol. 6, no. 10, pp. 570–584, 2009.

[11] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1180–1202, 2010.

[12] Q. Yang, L. Wang, and N. Ahuja, "A constant-space belief propagation algorithm for stereo matching," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '10)*, pp. 1458–1465, San Francisco, Calif, USA, June 2010.

[13] L. Jiangbo, S. Rogmans, G. Lafruit, and F. Catthoor, "Real-time stereo correspondence using a truncated separable laplacian kernel approximation on graphics hardware," in *Proceedings of the IEEE International Conference onMultimedia and Expo (ICME '07)*, pp. 1946–1949, Beijing, China, July 2007.

[14] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen, "Hardware-efficient belief propagation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 525–537, 2011.

[15] 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.

[16] L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest, "Real-time high-definition stereo matching on FPGA," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '11)*, pp. 55–64, ACM, New York, NY, USA, March 2011.

[17] K. Ambrosch and W. Kubinger, "Accurate hardware-based stereo vision," *Computer Vision and Image Understanding*, vol. 114, no. 11, pp. 1303–1316, 2010.

[18] S. Jin, J. Cho, X. D. Pham et al., "FPGA design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, 2010.

[19] A. Klaus, M. Sormann, and K. Karner, "Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure," in *Proceedings of the 18th International*

Conference on Pattern Recognition (ICPR '06), pp. 15–18, Hong Kong, August 2006.

[20] A. F. Bobick and S. S. Intille, "Large occlusion stereo," *International Journal of Computer Vision*, vol. 33, no. 3, pp. 181–200, 1999.

[21] R. Brockers, "Cooperative stereo matching with color-based adaptive local support," in *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns (CAIP '09)*, pp. 1019–1027, Springer, Berlin, Germany, 2009.

[22] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV '06)*, p. 66, June 2006.

[23] W. Van Der Mark and D. M. Gavrila, "Real-time dense stereo for intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 38–50, 2006.

[24] S. B. Goldberg and L. Matthies, "Stereo and IMU assisted visual odometry on an OMAP3530 for small robots," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '11)*, Colorado Springs, Colo, USA, June 2011.

[25] L. Matthies, A. Kelly, T. Litwin, and G. Tharp, "Obstacle detection for unmanned ground vehicles: a progress report," in *Proceedings of the Intelligent Vehicles Symposium*, pp. 66–71, September 1995.

[26] J. Byrne, M. Cosgrove, and R. Mehra, "Stereo based obstacle detection for an unmanned air vehicle," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pp. 2830–2835, Orlando, Fla, USA, May 2006.

[27] K. Konolige, "Small vision systems: hardware and implementation," in *Proceedings of the 8th International Symposium on Robotics Research*, pp. 111–116, 1997.

[28] J. M. Pérez, P. Sánchez, and M. Martínez, "High memory throughput FPGA architecture for high-definition belief-propagation stereo matching," in *Proceedings of the 3rd International Conference on Signals, Circuits and Systems (SCS '09)*, pp. 1–6, Medinine, Tunisia, November 2009.

[29] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, "SAD-based stereo matching using FPGAs," in *Embedded Computer Vision*, B. Kisaanin, S. S. Bhattacharyya, and S. Chai, Eds., series Advances in Pattern Recognition, pp. 121–138, Springer, London, UK, 2009.

[30] 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.

[31] R. Kalarot and J. Morris, "Comparison of FPGA and GPU implementations of real-time stereo vision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition—Workshops (CVPRW '10)*, pp. 9–15, San Francisco, Calif, USA, June 2010.

[32] J. I. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck, "The Tyzx DeepSea G2 Vision System, A taskable, embedded stereo camera," in *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops*, p. 126, New York, NY, USA, June 2006.

[33] A. Naoulou, J.-L. Boizard, J. Y. Fourniols, and M. Devy, "An alternative to sequential architectures to improve the processing time of passive stereovision algorithms," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–4, Madrid, Spain, August 2006.

[34] 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.

[35] Y. Miyajima and T. Maruyama, "A real-time stereo vision system with FPGA," in *Field-Programmable Logic and Applications*, vol. 2778 of *Lecture Notes in Computer Science*, pp. 448–457, Springer, Berlin, Germany, 2003.

[36] D. K. Masrani and W. J. MacLean, "A real-time large disparity range stereo-system using FPGAs," in *Proceedings of the 4th IEEE International Conference on Computer Vision Systems (ICVS '06)*, p. 13, January 2006.

[37] S. Park and H. Jeong, "Real-time stereo vision FPGA chip with low error rate," in *Proceedings of the International Conference on Multimedia and Ubiquitous Engineering (MUE '07)*, pp. 751–756, Seoul, Republic of Korea, April 2007.

[38] S. Sabihuddin and W. J. MacLean, "Maximum-likelihood stereo correspondence using field programmable gate arrays," in *Proceedings of the 5th International Conference on Computer Vision Systems*, 2007.

[39] J. Woodfill and B. Von Herzen, "Real-time stereo vision on the PARTS reconfigurable computer," in *Proceedings of the 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 201–210, April 1997.

[40] M. Li and Y. Jia, "Stereo vision system on programmable chip (SVSoC) for small robot navigation," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pp. 1359–1365, Beijing, China, October 2006.

[41] S. Perri, P. Corsonello, and G. Cocorullo, "Adaptive census transform: a novel hardware-oriented stereovision algorithm," *Computer Vision and Image Understanding*, vol. 117, no. 1, pp. 29–41, 2013, http://www.sciencedirect.com/science/article/pii/S1077314212001336.