Machine Vision and Applications (2006) 17(2): 116–132
DOI 10.1007/s00138-006-0018-2

ORIGINAL PAPER

Ahmad Darabiha · W. James MacLean · Jonathan Rose

Reconfigurable hardware implementation of a phase-correlation stereo algorithm

Received: 24 May 2005 / Accepted: 24 December 2005 / Published online: 7 March 2006

Abstract This paper describes the implementation of a stereo-vision system using Field Programmable Gate Arrays (FPGAs). Reconfigurable hardware, including FPGAs, is an attractive platform for implementing vision algorithms due to its ability to exploit parallelism often found in these algorithms, and due to the speed with which applications can be developed as compared to hardware. The system outputs 8-bit, subpixel disparity estimates for 256 × 360 pixel images at 30 fps. A local-weighted phase correlation algorithm for stereo disparity [Fleet, D. J.: Int. Conf. Syst. Man Cybernetics **1**:48–54 (1994)] is implemented. Despite the complexity of performing correlations on multiscale, multiorientation phase data, the system runs as much as 300 times faster in hardware than its software implementation. This paper describes the hardware platform used, the algorithm, and the issues encountered during its hardware implementation. Of particular interest is the implementation of multiscale, steerable filters, which are widely used in computer vision algorithms. Several trade-offs (reducing the number of filter orientations from three to two, using fixed-point computation, changing the location of one localized low-pass filter, and using L1 instead of L2 norms) were required to both fit the design into the available hardware and to achieve video-rate processing. Finally, results from the system are given both for synthetic data sets as well as several standard stereo-pair test images.

Keywords Stereo disparity estimation · Frame rate implementation · Field Programmable Gate Arrays (FPGAs) · Reconfigurable hardware implementation · Phase correlation

1 Introduction

High-level computer vision tasks such as robot navigation and collision avoidance require 3-D depth information about the surrounding environment at frame rate. At present, few high-performance implementations of stereo-vision algorithms exist, and the fastest of these involve the use of special hardware features to achieve their high performance. Implementations based on general-purpose microprocessors have had to be of low computational complexity in order to achieve processing speeds above a few frames per second (fps), and exclude algorithms of moderate or high computational complexity. One solution to implementing complex algorithms at frame rates is to build custom hardware. This approach can exploit the inherent parallelism in image processing and vision problems since it is possible to build hardware to perform costly operations, for example 2-D convolutions, in the time it takes to acquire a single frame. Even ignoring parallelism, it is possible to implement operations that might take many CPU cycles in software, much faster in hardware. The downside to hardware-based approaches is that design of custom hardware has typically been a lengthy and expensive process. It may take months to develop and verify a design, and fabrication of boards and Application-Specific Integrated Circuits (ASICs) incurs costs ranging from hundreds to hundreds of thousands of dollars.

There is a third option available that bridges the gap between the ease of design associated with software and the performance associated with hardware. The advent of reconfigurable logic hardware in the form of Field-Programmable Gate Arrays (FPGAs) allows designs to be quickly developed and prototyped at relatively low cost. These devices contain large quantities of logic elements and have the ability to flexibly interconnect these elements to implement virtually any logic design imaginable. This can be done by downloading a sequence of bits to the device that describe the desired connectivity. The interconnections can be easily and rapidly changed to allow the same device to be used repeatedly for multiple purposes. High-level Hardware Description Languages (HDLs) such as VHDL and Verilog bring the process of hardware design a step closer to the ease of software design. New, high-level synthesis tools allow designers to work with familiar simulation tools, such as Matlab, and have the results easily translated into hardware. This

A. Darabiha · W. J. MacLean (✉) · J. Rose
Department of Electrical and Computer Engineering,
University of Toronto, Toronto, Canada M5S 1A1
E-mail: {ahmadd, maclean, jayar}@eecg.toronto.edu

shortens the time of the design-prototype cycle and allows the algorithm designer to work directly with the hardware as opposed to passing the design off to a hardware specialist.

This paper describes an FPGA-based implementation of a computationally complex stereo disparity algorithm based on locally weighted phase correlation [1]. Disparity is the apparent shift in the image positions of a scene point viewed by two cameras, and is a function of the depth of the scene point along the optic axis and the distance between the camera's optic centers. This particular algorithm was chosen due to an interest in phase-based recovery of disparity, and the system is the first hardware implementation of a phase-based stereo algorithm. Our motivation for implementation of this algorithm in hardware stems from a belief that reconfigurable hardware allows for high-performance implementations of computer vision algorithms that are impossible to match using software alone. Many current implementations of vision algorithms are far too slow to be used for frame-rate processing. Those algorithms that do run at frame rate on serial CPUs are often too simplified to give meaningful results, or use the entire resources of the CPU so that no other processing is possible. Some frame-rate systems require a cluster of serial CPUs and are therefore unsuitable for mobile applications. Finally, some frame-rate algorithms require the fastest serial CPUs currently available, and these may be unsuitable for particular environments, such as space exploration. In this case, the slower clock speeds of reconfigurable devices may allow them to be used, even without special efforts to make them "radiation hard."

An obvious drawback to using FPGAs is that, presently at least, the implementation of complex algorithms is strongly tied to the target architecture, and therefore not easily ported to newer, more powerful devices. FPGA manufacturers are working to remedy this by developing high-level programming environments to allow abstracting algorithm implementations from the target hardware. Commercial packages such as System Generator™ [2] and DSP Builder™ [3] are already simplifying this sort of design. FPGAs are already popular in embedded systems, and will likely become standard hardware on workstations and graphics cards in the not-too-distant future. Finally, while there is much interest in implementing vision algorithms on commodity graphics hardware [4, 5], it is important to note that FPGAs still provide a more flexible framework for processing than GPUs.

A final and important consideration is that reconfigurable hardware is, like software running on a serial CPU, *reusable*. It is reasonable to expect that a high-level vision system may comprise multiple task-specific modules, and that not all modules are in use at any given time. In this case, reconfigurable hardware allows modules to be swapped in and out on an as-needed basis. For example, a robotic vision system might have a navigation module for use while the robot is in motion, an object recognition module when the robot is stationary and manipulating objects in its workspace, and perhaps a face and gesture recognition module to allow it to obey commands of human co-workers. Total hardware can be kept to a minimum by swapping in only the module(s) required by the current task. Keeping hardware to a minimum has both monetary and spatial benefits. Unlike software, processing capacity is expanded by increasing the number of FPGA chips, with the added overhead of sharing signals between the devices.

An outline of this paper is as follows. We start with a brief description of the FPGA system used in this work in Sect. 2. Previous approaches to implementing hardware-based and hardware-accelerated stereo algorithms are reviewed in Sect. 3. Next is a description of the stereo algorithm implemented in this work. Then comes a description of our design in Sect. 5, including an overview of our proposed system architecture in Sect. 5.1, and several specific implementation issues in Sect. 5.2–5.4. Section 6 contains a discussion of the results, with the imaging system specified in Sect. 6.1. System performance measures are discussed in Sect. 6.2, and results for synthetic and several standard stereo-pair images are given in Sect. 6.3. We conclude in Sect. 7 and provide some suggestions for future directions.

2 Transmogrifier-3A

The Transmogrifier-3A (TM-3A) [6, 7], shown in Fig. 1, is a reconfigurable board built at the University of Toronto containing four Xilinx Virtex2000E FPGAs [8]. A 98-bit bus exists between each pair of FPGAs, allowing for designs larger than will fit into a single device. Each chip is also connected to a 256K $\times$ 64 bit synchronous SRAM memory, an I/O connector, and a bus which allows communication with a housekeeping FPGA. The housekeeping chip communicates with the host computer for download and control functions. Video encoder/decoder chips are also integrated into the TM-3A board, which give the ability to receive an NTSC video stream and to send output results directly to an SVGA display. Circuits have been built on the TM-3A operating at frequencies of 50 MHz. The system as used was configured

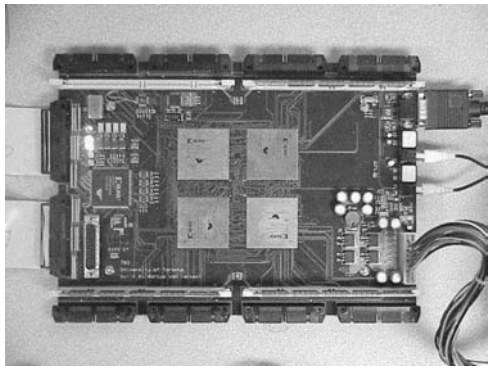


Fig. 1 The TM-3A board. The four large chips are Xilinx Virtex 2000Es. On the left ribbon cables connect the TM-3A to the host computer via an S-bus. On the right are video in/out connections and power cables. I/O connectors and memory modules are seen along the top and bottom edges of the board

with three SRAM modules, giving a total of 6 MB of SRAM memory.

3 Review of previous work

A variety of reconfigurable stereo machines have been introduced in recent years [9–13]. The PARTS reconfigurable computer [12] consists of a 4 × 4 array of mesh-connected FPGAs with a maximum total number of about 35,000 4-input LUTs. A stereo system was developed on PARTS based on the census transform, which mainly consists of bitwise comparisons and additions [13]. Kanade et al. [11] describe a hybrid system using C40 digital signal processors together with programmable logic devices (PLDs, similar to FPGAs) mounted on boards in a VME-bus backplane. The system, which the authors do not claim to be reconfigurable, implements a sum-of-absolute-differences along predetermined epipolar geometry to generate 5-bit disparity estimates at frame-rate. One interesting feature of this system is that it can accept inputs from up to six cameras. In Faugeras et al. [9], a 4 × 4 matrix of small FPGAs is used to perform the cross-correlation of two 256 × 256 images in 140 ms. In Hou et al. [10], a combination of FPGA and Digital Signal Processors (DSPs) is used to perform edge-based stereo vision. Their approach uses FPGAs to perform low level tasks like edge detection and uses DSPs for higher level integration tasks.

Not all previous hardware approaches have been based on reconfigurable devices. Burt [14] describes an ASIC (nonreconfigurable) designed to act as a front-end for video processing algorithms. The chip contains a stereo module which generates 8-bit disparity measurements with subpixel accuracy. A "sum of absolute difference" (SAD) algorithm is used, with programmable regions ranging from 7 × 7 to 13 × 13. Disparity estimates are generated at frame rates. The chip clocks at 100 MHz and is capable of 80 GOPS, although not all of this processing is used by the stereo module. This work is continued in [15]. Yang and Pollefeys [5] report an implementation of a multiresolution sum of squared differences (SSD) stereo algorithm, with a throughput reported at 6–8 fps on 256 × 256 images (although no performance data with respect to accuracy is given). Their solution relies on a serendipitous match between their algorithm's requirements and available fast operations on the graphics processor, so it is not clear if other vision algorithms (or even other stereo algorithms) will be as easy to map onto graphics hardware.

A number of fast algorithms that do not use reconfigurable hardware also exist in the literature. Some of these take advantage of special hardware, specifically SIMD (single instruction, multiple data) instructions in Intel MMX processors [16, 17]. A number of intensity-based cross-correlation techniques are reported [16–20]. Approaches to speed up algorithms include the use of image pyramids [14, 19, 21, 22] and the use of simplified algorithms [23].

The two fastest algorithms both use MMX processors. Hirschmuller et al. [16] achieve 4.7 fps on 320 × 240 images using intensity correlation in 7 × 7 windows with Gaussian prefiltering. Their method includes both rectification and left–right consistency checking (without these the frame rate is estimated at 5.7 fps). Their hardware is a 450 MHz Pentium running Linux. Muhlmann [17], using an 800 MHz MMX processor, achieves less than 5 fps on 348 × 288 color images, again using intensity correlation. A trinocular disparity system is implemented by Mulligan et al. [18] on a 4-processor system utilizing Intel IPL libraries. This system takes 456 ms to compute disparity for three 320 × 240 images up to a maximum disparity of 64 pixels. Sun [19] computes disparity (up to 10 pixels) in 450 ms on a 500 MHz Pentium processor based on fast intensity correlation in an image pyramid.

Birchfield and Tomasi [23] use a simplified algorithm that minimizes a 1-D cost function based on pixel intensity differences. They report speeds of 4 s/frame on a 333 MHz Pentium for 640 × 480 images. Given a faster processor their algorithm would doubtlessly perform better than 1 fps. It has the advantage of explicitly accounting for occlusion as well as propagating disparity information between scan lines, a property typically found in global algorithms according to [24].

A number of commercial stereo-vision systems are currently available. Point Grey Research [25] offers IEEE1394-based two- and three-camera systems with software development kits for Windows platforms. They claim performances of 26 fps for 320 × 240 images and a disparity range of 48 pixels with subpixel interpolation on a 2.4 GHz P4 system, although this drops to 4.3 fps for 640 × 480 images. The algorithm includes rectification, and cross validation, but is based on SAD. The Small Vision System [26], based on the SRI stereo engine and the Small Vision System by Konolige [27] is based on SAD performed on prefiltered images, and claims performance of 340 × 240 images for disparity of 32 pixels at 30 fps on a 700 MHz P3 system, although the example on their website is listed as 340 × 240 images for disparity of 24 pixels at 15 fps. The system requires MMX technology for best performance, and slows down as the disparity range is increased and as the correlation area is increased. The CanestaVision Electronic Perception Development Kit [28] gives depth maps for a 64 × 64 image array based on "time of flight" using an IR laser to actively illuminate the image. The main advantage here is the lack of requirement for texture in the scene, but it has limited range (on the order of a few metres), is slow (on the order of 5 fps) and can be strongly affected by ambient light.

4 Local weighted phase correlation stereo algorithm

The heart of any stereo-vision system is stereo matching, the goal of which is to establish correspondence between points in the left and right images arising from the same element in the scene. Stereo matching is complicated by factors such as lack of texture, occlusion, depth discontinuity and image noise. Techniques proposed to solve the

correspondence problem and improve the performance of stereo matching can be categorized into three major groups: (1) Intensity-based; (2) Feature-based; and (3) Phase-based. Intensity-based techniques assume that image intensity corresponding to a 3-D point remains the same in binocular images. These techniques usually lead to window-based and coarse-to-fine strategies. Intensity-based methods are often confused by changes in image scale between the left and right images, and also by deformation between the two. Changes in intensity are also problematic, but this effect can be minimized through the use of normalization techniques, albeit with considerable additional computational effort. Feature-based techniques use sparse primitives such as edges [29] or straight line segments [30]. The major limitation of all feature-based techniques is that they cannot generate dense disparity maps, and hence they often need to be used in conjunction with other techniques. In phase-based techniques, the disparity is defined as the shift necessary to align the phase value of band-pass filtered versions of two images. It has been shown that phase-based methods are robust to smooth lighting variations and modest deformations between stereo images [31, 32]. It has also been shown that phase is predominantly linear [32], and hence reliable approximations to disparity can be extracted from phase displacement [31].

The stereo system described in this paper is based on a phase-based stereo matching technique called "Local Weighted Phase-Correlation" (LWPC) [1]. This algorithm combines the robustness of phase-difference methods with the simple control strategy of phase-correlation methods. The LWPC algorithm has four major steps:

1. Create a Gaussian pyramid with total number of S scales for both left and right images. Then apply spatially-oriented quadrature-pair filters [33] to each scale of the pyramid. If $K_j(x)$ is the filter impulse response of the jth orientation, then we can write the complex-valued output of the convolution of $K_j(x)$ with each scale of the left and right images, $I_l(x)$ and $I_r(x)$, as

$$O_l(x) = \rho_l(x)e^{i\phi_l(x)} = K_j(x) \otimes I_l(x) \quad \text{and}$$
$$O_r(x) = \rho_r(x)e^{i\phi_r(x)} = K_j(x) \otimes I_r(x)$$

   in polar representation, where $\rho(x) = |O(x)|$ is the amplitude and $\phi(x) = \arg[O(x)]$ is the phase of the complex response.
2. For each scale and orientation, compute local voting functions $C_{j,s}(x, \tau)$ as

$$C_{j,s}(x,\tau) = \frac{W(x) \otimes [O_l(x)O_r^*(x+\tau)]}{\sqrt{W(x) \otimes |O_l(x)|^2}\sqrt{W(x) \otimes |O_r(x)|^2}}, \tag{1}$$

   where $W(x)$ is a smoothing, localized window and τ is the preshift of the right filter output.
3. Combine the voting functions $C_{j,s}(x, \tau)$ over all orientations, $1 \le j \le F$, and scales, $1 \le s \le S$, where F is the total number of orientations, to get the cumulative voting function

$$V(x,\tau) = \sum_{j,s} C_{j,s}(x,\tau).$$

4. For each image position x, find the τ value corresponding to the peak in the real part of $V(x, \tau)$ as a good estimate for the true disparity.

The cumulative voting function is expected to be a more accurate indicator of disparity as spurious peaks at different scales and orientations will be uncorrelated, but the true disparity will create a peak in the same location in each local voting function, and this peak will be dominant in the cumulative voting function.

Two major features make this algorithm a good candidate for hardware implementation. First, it is primarily composed of linear operations that are easily implemented in hardware. Operations such as addition and multiplication can be represented efficiently in terms of number of logic blocks required, and can be computed in few, or even one, clock cycles. Second, there is no iteration nor any explicit coarse-to-fine control strategy. This property makes the real-time flow of data possible through the hardware. In the next sections, we will describe the hardware of the system and then modifications applied to the original LWPC method.

5 System design

5.1 System overview

When implementing a complex algorithm on re-programmable hardware, the most important issue is that there is a fixed amount of hardware available as described in Sect. 2. These hardware resources include logic capacity, on-FPGA and off-FPGA available memory, memory access bandwidth and chip-to-chip communication bandwidth. Achieving the best overall performance requires efficient usage of all hardware resources.

In this work, for parallel and efficient hardware implementation of the stereo depth measurement, some modifications are introduced to the original LWPC algorithm. Three major modifications are: (1) Employing fixed-point data representation *vs.* floating-point representation; (2) Changing the location of a low pass localized filter; and (3) Using the L1 norm instead of the L2 norm in calculating phase correlation. In this section, we first describe the major building blocks of the system and the distribution of the tasks over four FPGAs available on the TM-3A board. Then we will discuss the advantages and effects of modifications on the overall system performance.

The architecture of the stereo-vision system is illustrated in Fig. 2. It consists of four major units: the Video Interface Unit, the Scale-Orientation Decomposition unit, the Phase-Correlation unit and the Interpolation/Peak Detection unit. Each of these units is implemented on one of the Xilinx V2000E FPGAs available on the TM-3A.

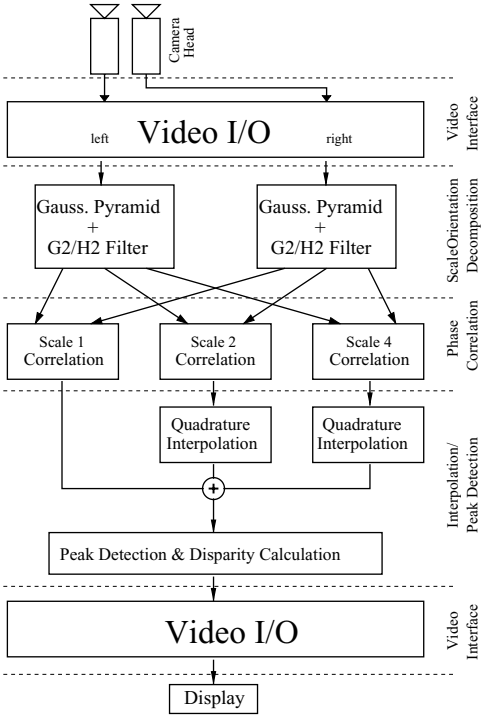


Fig. 2 This diagram shows a high-level view of the system architecture. The system is divided into four major blocks, each of which is implemented on one of the four FPGAs on the TM-3A

The *Video Interface Unit* receives the video signal from cameras in composite NTSC format with an image size of 512 × 720 pixels and immediately downsamples to 256 × 360. Since there is only one video input channel available at a time on the TM-3A board, we switch between synchronized camera signals after each frame such that we receive 15 fps from each camera. However, the stereo-vision system itself processes and outputs 30 fps using buffered images. If there were two video inputs available,[1] the system would in fact be processing 30 unique image-pairs each second. The video is de-interlaced as it is acquired, causing a 1/60th second delay as the first field is stored in memory. After frame buffering, the video interface unit sends two noninterlaced gray scale image streams to the Scale/Orientation decomposition unit.

In the *Scale-Orientation Decomposition Unit*, a 3-level Gaussian pyramid is built for both left and right images. Each level of the pyramid is built by low-pass filtering with a 3-tap FIR filter and subsampling by a factor of two. Each of the pyramid levels is then decomposed into +45° and −45° orientations using G_2/H_2 steerable filters [33]. While the original algorithm [1] also calls for a third orientation of 0°, this orientation was omitted due to resource limitations. Based on results given in Sect. 6.3, this omission has not caused a large degradation in system performance. The G_2/H_2 filters use a set of seven separable 7 × 7 FIR filters as basis filters. The tap coefficients in the filter were quantized to 8-bits. By choosing proper coefficients for the linear combination of the basis filters, one can synthesize filters of arbitrary orientation. One important advantage of using G_2/H_2 filters for hardware implementation is that they are X–Y separable and therefore require less hardware resources than nonseparable filters of the same size. The vertical component of each filter is applied first, generating a series of 16-bit values that are then passed to the horizontal filter components. The order of application of the filter components does not affect the result of the computations, but it does have an impact on resource allocation. All FIR filter designs in hardware require the input data to be stored in a series of delay buffers prior to multiplication by filter coefficients and summation. Since video data arrives row-by-row, the delay buffers for the horizontal filter components are quite small, while those for the vertical components must store as many scan lines of data as the filter has taps. By applying the vertical filter components first, a common set of buffers can be used for all seven vertical filter components, thus saving on memory resources. The output of each vertical filter component must be buffered separately, but these buffers are much smaller by comparison as they feed the horizontal filter components. This arrangement is shown in Fig. 3. At the end of this unit, filter outputs are reduced to 16-bits before being sent to the phase-correlation unit. The issues related to the effects of fixed-point representation will be discussed in Sect. 5.2.

Adders and multipliers implemented in logic blocks may be optimized either for maximum speed or minimum area (logic blocks used). In this system, since the system clock speed of 48 MHz is much higher than the arrival of pixels in the data stream (roughly 5.6 MHz), we have spread addition and multiplication computations over multiple cycles wherever possible in order to save on FPGA resources.

The *Phase-Correlation Unit* is the heart of the algorithm. For each pixel in the left image, it computes the real part of the voting function $C_{j,s}(x, \tau)$ as mentioned in Eq. (1) for $0 \leq \tau \leq D$, where τ is a preshift in the right image and D is the maximum allowed disparity. The value for D for the finest scale ($s = 1$) is set to 20 pixels. Hence, for the next coarser scales, D is set to 10 and 5, respectively.

Figure 4 shows the architecture of the phase correlation block as dictated by the original algorithm. The voting functions $C_{j,s}(x, \tau)$ are reduced to an 8-bit representation before being sent to the interpolation unit.

The *Interpolation/Peak-Detection Unit* interpolates the two coarser scale voting functions, $C_{j,2}(x, \tau)$ and $C_{j,3}(x, \tau)$, in both x and τ domains such that they can be combined with the finest scale voting function $C_{j,1}(x, \tau)$. It performs quadrature interpolation in the τ domain and constant interpolation in the x domain. The interpolated voting functions are then added together to produce the cumulative voting function $V(x, \tau)$.

The final step in this unit is peak detection. For each pixel x in the image, it detects the value of τ for which $V(x, \tau)$ is maximum. By performing subpixel interpolation of the disparities, this unit produces disparity values with

[1] As is the case in the newly designed Transmogrifier-4.

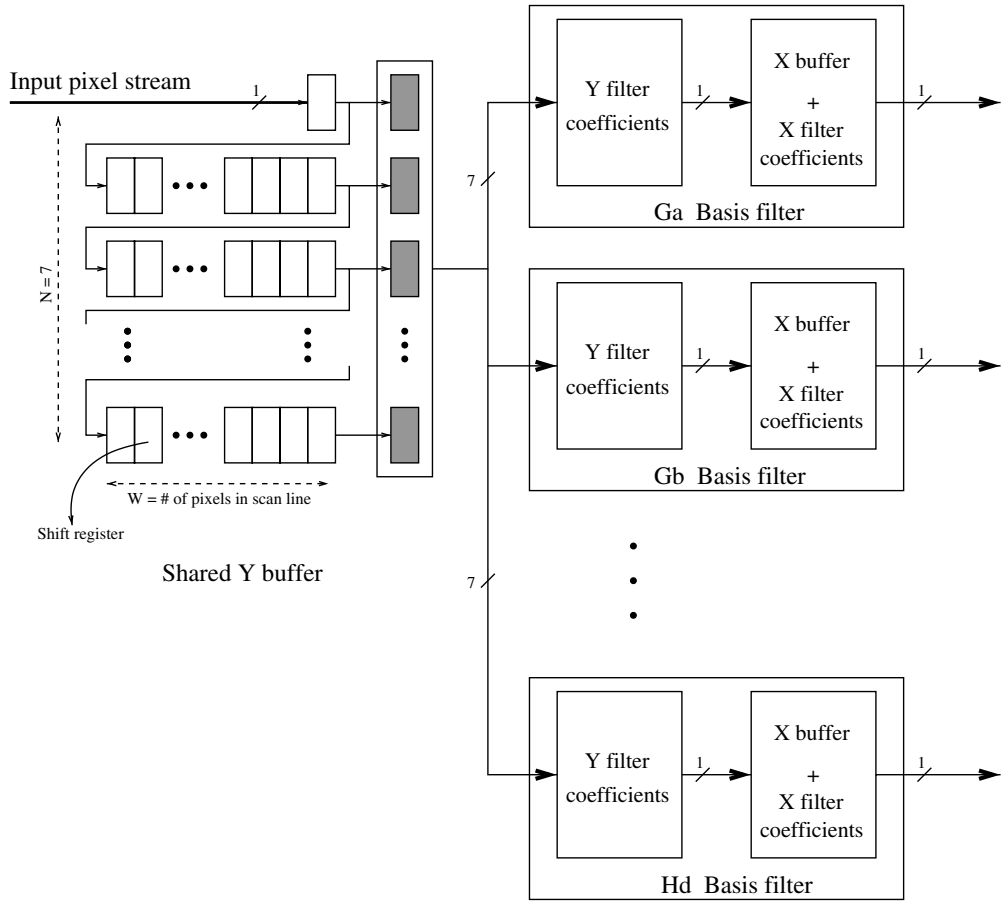


Fig. 3 The arrangement of delay buffers for the horizontal and vertical components of the G_2 and H_2 basis filters is shown. By applying the vertical component first, the larger delay buffer structure can be shared among all seven basis filters, thus saving resources

8-bit resolution from the 20-pixel disparity range. The final disparity results are sent back to the video interface unit to be written in the video output buffer and displayed on a monitor. Table 1 lists the hardware resources used in each unit in terms of number of Look Up Tables (LUTs), flip-flops and the number of on-chip fast memory banks. Each row represents resource utilization on one of the TM-3A's FPGAs. In the design of the hardware, we have assumed that the two cameras have identical focal length and are vertically aligned such that no rectification [34] is required. Also, there is no postprocessing stage such as left-to-right/right-to-left validation or smoothing/gap filling implemented in hardware. As will be pointed out in Sect. 6, these steps have been left out due to space limitations, but are currently being added in a new version of the system.

5.2 Fixed-point representation

There is always a trade-off between the accuracy of fixed-point representation and the hardware efficiency: minimum quantization error requires using wide fixed-point representations, but wider signals require larger mathematical operators (dividers, multipliers, adders, *etc*.). In addition, more hardware resources are needed for data path circuits. As an example, Fig. 5 shows the relation between the required number of LUTs to create parallel (one clock cycle) multipliers or dividers versus the input width of the multiplier or divider. The number of LUTs increases with the square of the input width.

We need to make good decisions for the precision of the variables and operations in each stage of the algorithm. This

Table 1 Resources consumed by each section of the design

Unit name	4-Input LUTs	Flip-flops	On-chip memory (bits)
Video interface	169	71	–
Scale/orientation decomposition	23,151	18,020	614,400
Phase correlation	16,709	30,961	–
Interpolation peak detection	26,615	33,974	172,032

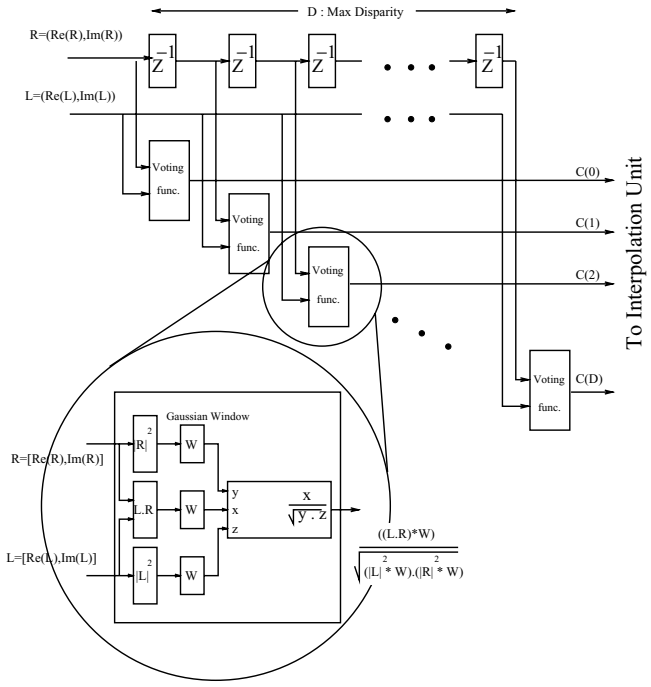


Fig. 4 This diagram shows the architecture of the phase correlation block as required by the original algorithm

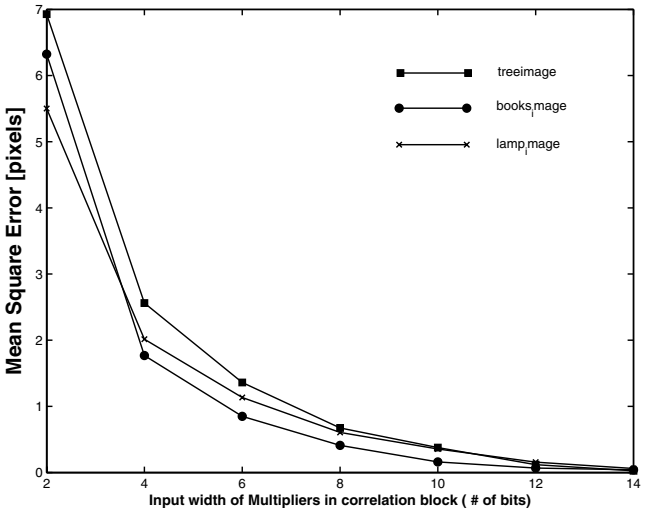


Fig. 6 This graph shows the error in the final disparity estimate (as compared to the floating point algorithm implementation) resulting from different multiplier widths in the phase correlation block. Since multiplier cost increases as the square of the input width, 8-bit multipliers were chosen as they allowed the disparity error to be kept below 1 pixel

analysis requires both knowledge of the target hardware and the algorithm itself. Few tools have been developed to solve this problem. For example, Chang and Hauck [35] present a framework for automatically determining fixed-point precision of floating-point calculations. Our approach has been to optimize width of data paths using real data together with knowledge of the constraints imposed by the target hardware to choose appropriate data widths. Before moving to hardware, we conducted simulations in order to find the most efficient precisions for each stage of the system. To find the optimum precision, we have simulated our hardware system for a range of precisions and calculated the quantization error for each.

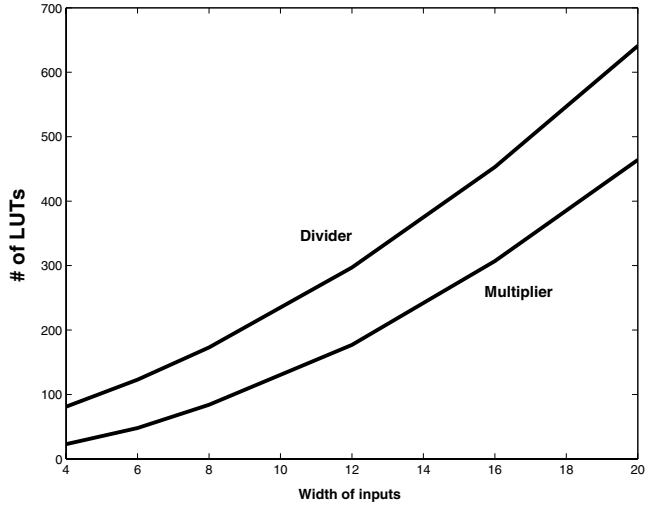


Fig. 5 The cost of multipliers and dividers, in terms of the number of LUTs required to implement them, as a function of the width of the operands in bits. The plots are for implementations that complete the operation in a single clock cycle – implementations that "pipeline" the operation over several clock cycles can reduce the required number of LUTs

Figure 6 shows the Mean Square Error between using full-precision and a fixed-point multiplier in the correlation unit for a range of precisions (assuming that all other stages are in full precision). In this case, for example, based on Figs. 5 and 6, an 8-bit multiplier was chosen for the phase-correlation unit. Since resolution need not be fixed across all stages of the system, we performed similar analyses for each of the units, and we have chosen 16- and 8-bit widths for Orientation Decomposition Unit and Interpolation Unit multipliers, respectively.

5.3 Relocation of weighting function

The implementation of the correlation block based on the architecture of Fig. 4 requires hardware resources beyond the capacity of a Virtex 2000E FPGA. To shrink the size of the circuit, we have modified the correlation block as shown in Fig. 7. In this revised architecture we have placed the Gaussian window, W, at the end of voting function blocks. This change allows us to extract the common portions of the computations out of voting function units as much as possible. Extracting the two normalization blocks in Fig. 7 leaves a simple inner product calculation and a single Gaussian window inside each voting function block. Each normalization block receives a complex-valued input and divides it by its magnitude such that the output has the same phase but with unit magnitude. Changing the location of Gaussian

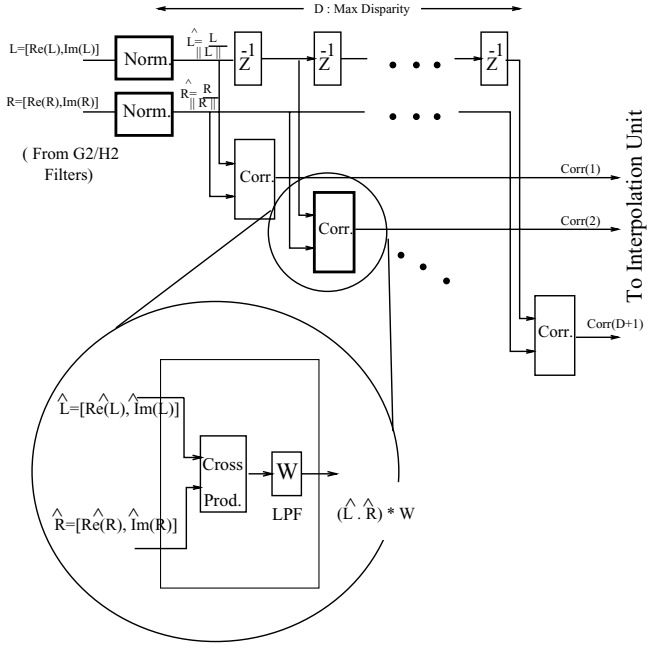


Fig. 7 Revised structure of the correlation block. Each complex value is normalized before entering the correlation block, and the weighting function is applied after normalization and computation of the inner product

window along with sharing the normalization block reduces the total number of multipliers, dividers and square roots in the correlation unit by more than 65%. Since the output of the normalization blocks is in the limited range of $[-1, +1]$, it is possible to use a small number of bits in representing their output. At present the output is 8 bits.

The drawback of this architecture is that the Gaussian window relocation to the end of voting function block is not an exact analytical equivalent of the original method. In fact, by comparison with [31], it is seen that the revised computation closely resembles a previously proposed phase differencing method, except that the difference function is now windowed and the voting structure has been retained. One can show that for FIR filters close to an impulse function, this revision is a reasonable approximation [36] to the original algorithm. Since in our stereo-vision system a 3-tap Gaussian LP filter is used as W, this trade off seems to be reasonable.

5.4 Normalization of complex values

A further reduction in the size of the phase-correlation unit is achieved through modification of the architecture of the normalization block. This block is used to compute the magnitude of the complex-valued inputs. The L2-norm of a complex number A is defined as $||A||_2 = \sqrt{\Re\{A\}^2 + \Im\{A\}^2}$. The hardware implementation of $||A||_2$ is expensive because it requires two multipliers, one square root and one adder. Instead, we replace it with the L1-norm of A, defined as $||A||_1 = |\Re\{A\}| + |\Im\{A\}|$. Figure 8 shows the effect of using $||A||_1$ instead of $||A||_2$ to normalize the output. When using L2, all the normalized vectors are located on the unit circle in the Real-Imaginary plane, but in L1 they are projected on a square as shown in Fig. 8. This is because the sum of absolute real and imaginary parts of L1-normalized vectors is always unity and therefore they are projected to straight lines in each quadrant which form a square instead of a unit circle. This technique provides enough accuracy for our application and may also be used in other applications requiring computation of vector norms. To improve the accuracy of the normalization operation and still avoid implementation of $||A||_2$, there exists a different solution to the one we implemented: since $\Re\{A_{u1}\}$ and $\Im\{A_{u1}\}$, shown in Fig. 8, always lie between -1 and 1, one can use a memory block as a look up table with appropriate values to replace current A_{u1} vectors with corresponding points on unit circle. These look up tables can be built using on-chip memory, which is available in most current FPGA devices, without the need to use logic elements of the device. An example of the effect of replacing the L2 norm with an L1 norm in our system is given in Fig. 9.

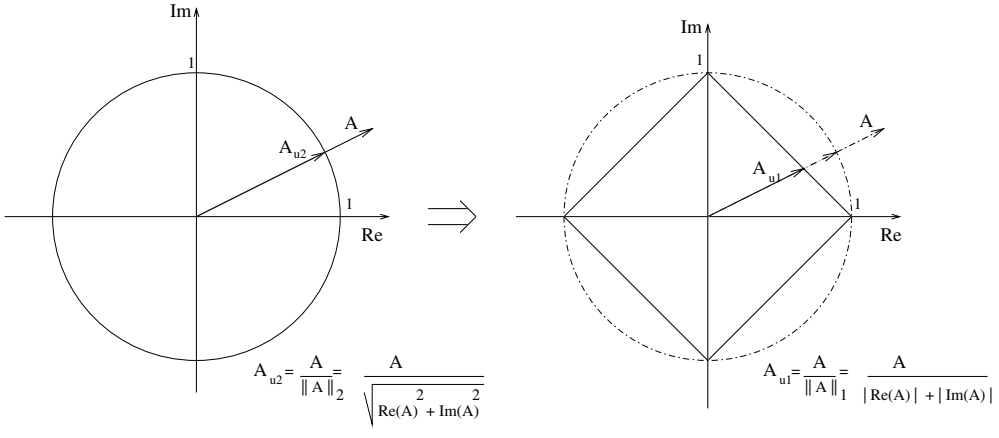
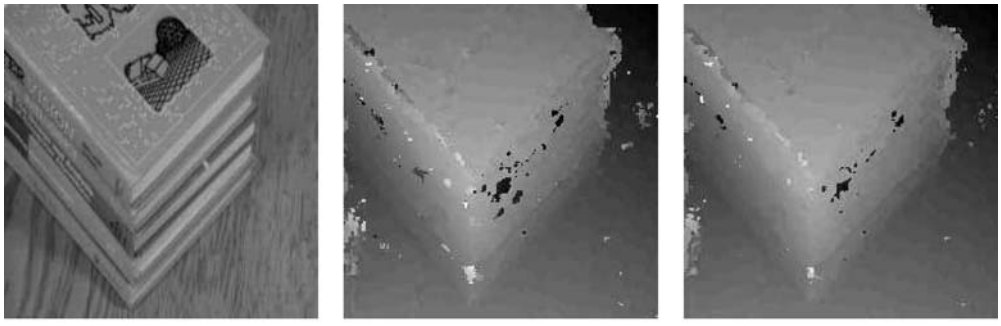


Fig. 8 The effect of using an L1 norm in computations instead of an L2 norm



(a) Left Image (b) Disparity with L1 Norm (c) Disparity with L2 Norm

Fig. 9 This figure compares the performance of the L1 norm against that of the L2 norm. The differences between the two methods appears minor, and can be characterized in a manner similar to quantization noise

Finally, it is worth considering the resource requirements to implement an L2 norm. Each magnitude requires (i) computing $\Re\{A\}^2 + \Im\{A\}^2$, and then (ii) computing the square root of this value. Step (i), assuming 16-bit multipliers operating over four clock cycles, would require roughly 200 4-LUTs. Step (ii) would require a further 760 4-LUTs to compute the square root at the required precision. Since 12 magnitude units would be required, this entails roughly 11,500 4-LUTs. While the FPGA implementing the phase correlation has 35,000 4-LUTs and only 16,709 are utilized, the chip is at 99% slice utilization, so it would not be possible to implement the L2 magnitude units. While a CORDIC implementation [37, 38] would be possible and would take less LUTs, the trade-off is LUTs for clock cycles and the resulting design would not be fast enough, assuming enough slices were available.

5.5 Subpixel peak detection

Another modification made to the original algorithm is the addition of a simple subpixel peak detection scheme. Once the integer-valued location of the voting function peak is found, a quadratic function is fitted locally to the peak and its two neighboring values. This requires two shift operations, four additions and one division. Since the integer peak location can be encoded in five bits, we can encode the subpixel correction in three bits to end up with an 8-bit output. Others have followed a similar procedure, for example Kanade et al. [11] perform the same interpolation on the C40 processor in their system.

5.6 Data communication between FPGAs

As mentioned previously, our design is spread across four FPGA devices mounted on a single circuit board. A data path 98-bits wide exists between each pair of chips. We found that 98-bits was often not enough to transmit the required data between chips. For example in the Scale/Orientation unit each image-pair has three pyramid levels, with two orientations each, each producing both a complex and real part, giving 24 16-bit values to be communicated to the phase correlation unit. This requires 384 bits to be transmitted for each clock cycle in the data pipeline. The transmission is achieved using a time-division multiplexing (TDM) scheme. Since the overall system clock runs at 48 MHz, we have at least six time slices in which to transfer data and achieve an overall throughput of 8 MHz. It should be noted that the use of TDM comes with a cost, as hardware resources are required to buffer data on both sides and manage the transfer.

6 Discussion

In Sect. 5 the implementation of the algorithm in FPGA hardware was described, and issues surrounding this implementation were discussed. In this section we evaluate the performance of the implementation through comparison with other stereo disparity implementations and through comparison with the original floating point software implementation of the algorithm.

6.1 Imaging system

Image acquisition for the stereo system uses two Hitachi KP-D51 color CCD cameras, with a CCD size of 7.55 mm × 6.45 mm and 12 mm lenses. The cameras are rigidly fixed on a common mounting bracket, and there is a separation of 70 mm between the lens centers. In the stereo-vision system, the size of the phase-correlation and interpolation hardware design is directly proportional to the maximum disparity that we specify between two images. We have set the maximum allowed disparity to 20 pixels. This corresponds to a minimum allowed distance of about 2 m from the objects to the stereo head. Other parameters such as separation between the cameras, CCD size and resolution can reduce the minimum distance, but they are usually restricted by optical and physical constraints. For example, to reduce the minimum distance, wide-angle lenses can be used but lenses with smaller focal length start to introduce radial distortion, that will degrade performance.

Table 2 Comparison of PDS performance for various systems

	Image size $n \times m$	Disparity D	Frame time (ms)	PDS ($\times 10^6$)	Algorithm	Platform
INRIA [9]	256 × 256	32	280	7.5	Intensity correlation	PeRLe-1 board (23 Xilinx XC3090 FPGAs
PARTS Engine [12]	240 × 320	24	23.8	77	Census	16 Xilinx 4025 FPGAs
CMU Stereo Machine [11]	200 × 200	30	33	36	SAD	Special purpose hardware (c40 DSP + real-time processor)
Yang/Pollefeys[a] [5]	512 × 512	20	71.4	58.9	multi-resolution SSD	NVIDIA GeForce4 Graphics card
Point Grey [25]	240 × 320	48	38.5	101.8	SAD	2.4 GHz P4
This work	256 × 360	20	33	55.2	LWPC (phase correlation)	TM-3A board (4 Xilinx Virtex 2000E FPGAs)

[a]This result is for a single image, and does not include real-time capture and rectification of images. The PDS value drops considerably when processing a captured sequence.

6.2 Performance

The FPGA stereo system described in this paper performs multiresolution, multiorientation disparity estimation based on local weighted phase-correlation. It produces a dense disparity map of size 256 × 360 pixels with 8-bit subpixel accuracy disparity results at the rate of 30 fps. In the metric of Points × Disparity Estimates per second (PDS), this system achieves a performance of 55.2×10^6 PDS, which is among the highest rates reported [12]. Taking into account the relative complexity of SAD vs. LWPC, the Point Grey result seems less dominant. Table 2 compares several stereo-vision systems. While the PDS metric reflects the density and the speed of the system, it does not reflect the complexity or accuracy of the implemented algorithm. An important feature of our system in comparison with other hardware stereo machines is its high-performance, phase-based algorithm. To realize a phase-based algorithm in video rate, the system performs the equivalent of more than 10 billion 16 × 16-bit multiplications per second and the four Virtex devices communicate in a data rate of up to 200 Mbytes/s. Our algorithm is the most computationally complex stereo algorithm yet implemented in hardware. The results given in Sect. 6.3 justify the use of such a computationally intensive algorithm.

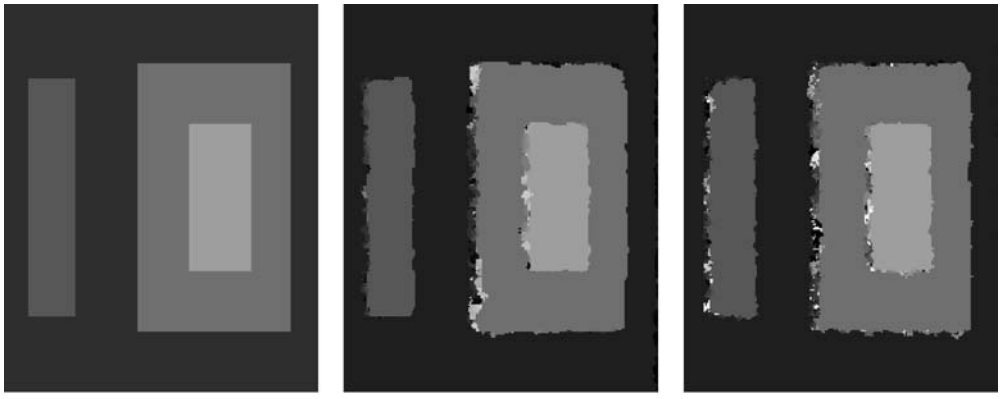
By comparison, the floating point version of the LWPC algorithm, implemented in Matlab scripts, takes approximately 30 s to compute disparity results for 256 × 360 images when run on a Sun UltraSPARC-III 750 MHz workstation with 2.5 GB of memory. While a comparison between hardware and an interpreted language may seem unfair, Matlab can be very computationally efficient so long as certain structures, such as loops, are avoided (as they have been in our scripts). Porting it to C would result in some speed improvement, perhaps a factor of ten, but certainly not enough to make it anywhere near frame-rate.

The disparity output by the system is just the output of the Peak-Detection unit without any postprocessing such as left-to-right/right-to-left validation [39] or smoothing/gap filling. Also, when the images are input to the system, they are not preprocessed to achieve rectification, although this should be done. We estimate that rectification and left-to-right/right-to-left validation can be added to the current system by increasing resources by approximately half of the size of one Xilinx Virtex2000E FPGA. Given the capacity of FPGA devices now available, this poses no problem. For example, as suggested in [11], rectification can be implemented using a simple memory look-up to transform image coordinates before doing the phase-correlation. The transform values can be computed off-line using standard calibration techniques and downloaded during the programming of the FPGAs. As another solution, rectification can be avoided as prerectified images can be directly generated using special optical setups [40]. An efficient technique to integrate the left–right, right–left validation feature to the current system is to alternate between left and right images after each subsequent frame. Since left–right and right–left matching is performed in different time slices, they can share the same blocks and hence there is no need for extra hardware. Even if we want to perform both matchings at the same time slice, we can still share the filtering blocks and need only implement the correlation and peak detection separately.

6.3 Results

In this section we examine the accuracy of the system. This will be done in two ways. First, we present stereo image pairs for which "ground truth" (or a good, independent estimate of it) is known. Results are presented for both a synthetic image pair containing a random-dot stereogram and also a real image-pair with hand measured depth at selected points. Our second method of evaluation will be to compare the hardware performance against output of the floating-point, software implementation of the LWPC algorithm. This will be done for several stereo-pairs commonly used to evaluate stereo disparity algorithms. This comparison uses images that have been previously rectified, and so removes this source of error from the reported results.

Figure 10 shows the disparity map from a random dot stereogram (RDS) extracted by the original LWPC



(a) Ground Truth Disparity (b) Floating Point Disparity (c) Hardware Disparity

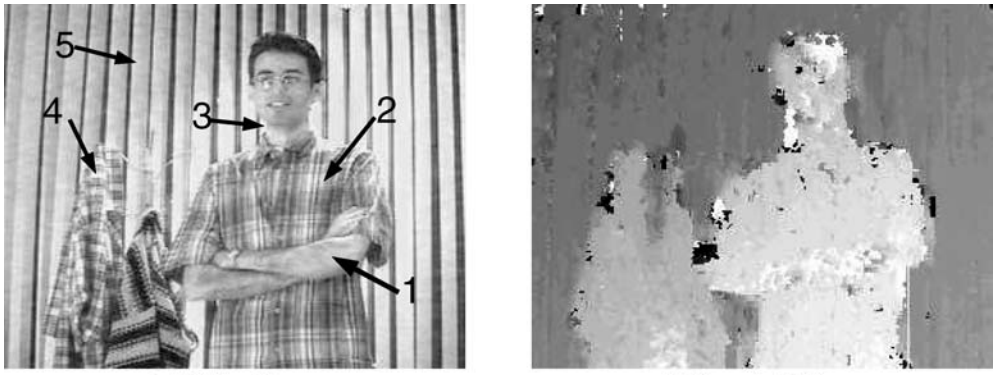
Fig. 10 Comparison of the performance of the software version of LWPC against the performance of the hardware. The input is a random-dot stereogram image pair whose ground-truth disparity is shown in (**a**). The performance of the software implementation and the hardware implementation are shown in (**b**) and (**c**) respectively. While they are not identical, the differences are confined to the left-edges of the depth discontinuities which are expected to be noisy (see text)

algorithm and also by the hardware. The RDS was generated by shifting blocks of pixels to the right. This explains the noise seen along the left edges of the disparity discontinuities: in these regions there are pixels that exist in one image but not the other, similar to occlusion that would be experienced in real scenes. In these regions the phase correlation attempts to find matches for pixels where none exists, so we do not expect the algorithm to perform well. Left–right/right–left validation, currently not part of the system, would allow this type of error to be labeled as such [39]. It should also be noted that the algorithm does not include either a texture test or other false-matching tests, which would at least allow it to give a certainty measure for the estimates produced. Figure 11 shows a sample image from the camera head and the depth map generated by the hardware.

In Table 3, the depth estimates obtained by our system are compared with hand-measured depths. Since the hardware outputs disparity, and not depth, estimates, depths were computed according to

$$d = \frac{fT}{D}$$

where d is depth, f is the focal length of the imaging system, T is the baseline between the cameras, and D is the disparity. Since we measure D in pixels, it is necessary to estimate the focal length in pixels via camera calibration. This formula assumes that the cameras are perfectly aligned, which of course they are not. A simple calibration was performed to estimate the value of the product fT prior to making the estimates shown in Table 3. As in most of the other stereo-matching algorithms, the phase-based stereo-matching algorithm is sensitive to depth discontinuity and lack of texture. In regions with sufficient texture, most of the depth measurement errors are less than 3%, but in regions with little or no texture or at depth discontinuities, where occlusion and dis-occlusion become an issue, the depth value estimates are less reliable. An example is the depth estimate for point 3 in Fig. 11a). This point lies on an occlusion boundary (where local algorithms are expected to do poorly, [24]), and inspection of the disparity map (b) shows that it is obviously in error. It is worth noting that despite the good results shown in Table 3, they would be even better if a full camera

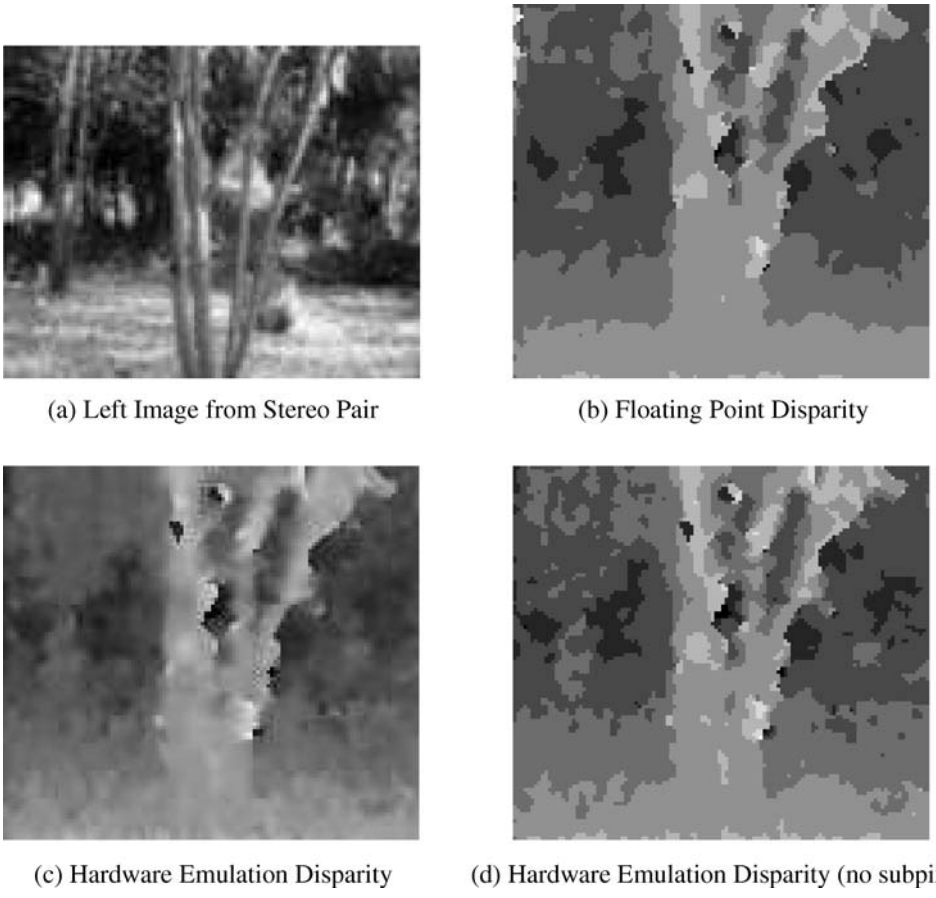


(a) Original Image (b) Disparity Estimates

Fig. 11 This figure compares the estimates of the hardware implementation against hand-measured ground truth shown in Table 3. In **a** is shown one of the original images from the pair used to generate the disparity map shown in **b**

Table 3 Depth measurements for points labeled in Fig. 11

Point #	1	2	3	4	5
Ground truth depth (cm)	300	315	320	354	396
Hardware depth estimate (cm)	309 (3%)	320 (1.6%)	276 (13.7%)	355 (0.3%)	402 (1.5%)



(a) Left Image from Stereo Pair (b) Floating Point Disparity

(c) Hardware Emulation Disparity (d) Hardware Emulation Disparity (no subpixel)

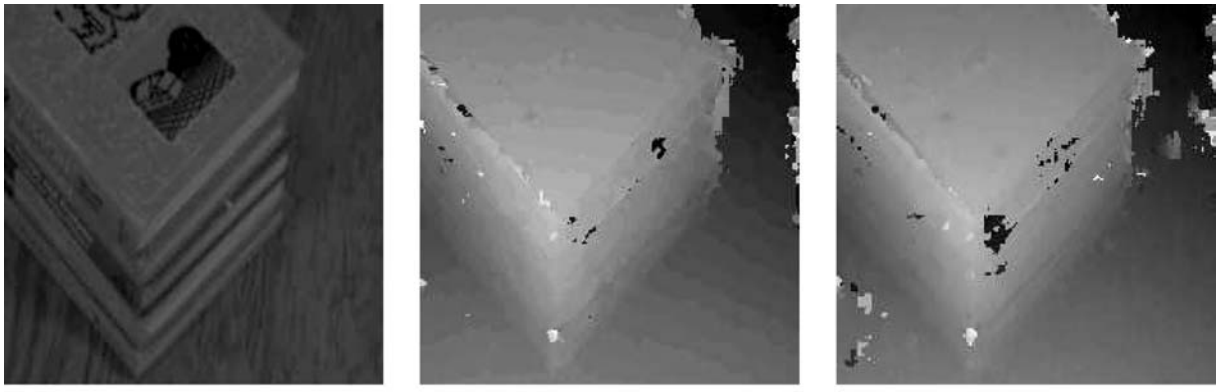
Fig. 12 Comparison of performance between the floating-point and fixed-point implementations of the algorithm. The disparities have been rescaled to the range [0, 255] for display purposes. Since the hardware emulation result (**c**) includes subpixel peak detection, it shows a greater number of different disparity values than are evident in (**b**). Hardware emulation results without subpixel peak detection are shown in (**d**)

calibration had been performed and the images had been rectified.

In Figs. 12 and 13, we see comparisons between the floating point version of the algorithm, and the hardware emulation. In these comparisons, we will see the effect of changes made to the algorithm, including the addition of subpixel peak detection. Figure 12 shows results from a stereo pair where the disparity is in the range $[-4, 4]$.[2] The measured error (root mean square error) between the software and hardware-emulation disparities is about 0.49 pixels (0.52 pixels with subpixel disparity turned off). The agreement between the two implementations is very good, despite the necessary changes in the hardware version.

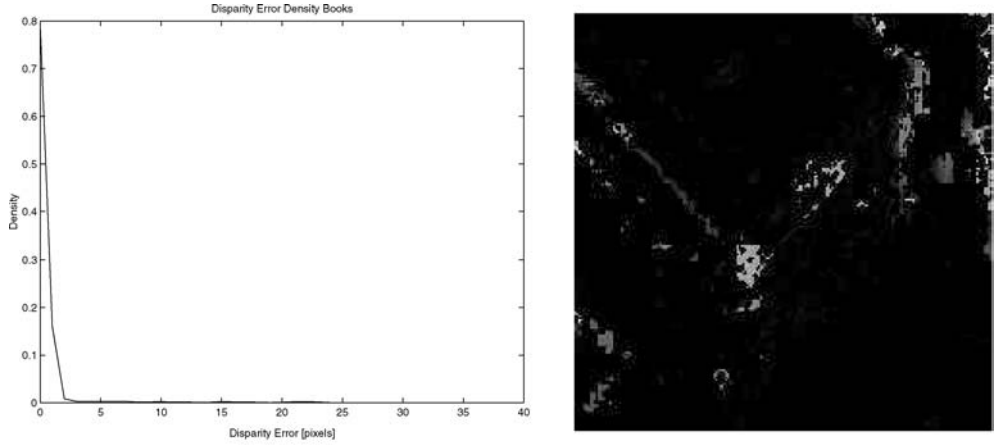
In Fig. 13, we see results from another stereo image pair, this time with disparities over a much larger range, $[-20, 20]$. The RMSE between the software and hardware-emulation in this case is 3.09 pixels. The error is 3.08 pixels with subpixel disparity turned off, since the original algorithm does not include subpixel estimation. In Fig. 14 we see a histogram of the absolute values of the errors. The distribution has a dominant peak at zero, indicating that most pixels have an error less than 1 pixel. The distribution also has a long tail, which may indicate a uniform distribution of errors in cases where no good match was found by either algorithm, but the software and hardware-emulation chose different values. In this sense the performance of the hardware implementation is probably much better than the RMSE would suggest. Figure 14b shows the distribution of errors over the entire image,

[2] Anytime results outside the range [0, 20] are quoted in this paper, the results are from an exact emulation of the fixed-point computation performed by the hardware.



(a) Left image (b) Floating Point Disparity (c) Emulation Disparity

Fig. 13 Another test sequence is shown here. The left image from the original stereo pair is shown in (**a**), the disparity calculated by the floating point version of the algorithm is shown in (**b**), and the hardware emulation version of the disparity is shown in (**c**)



(a) (b) Absolute Error Displayed by Pixel

Fig. 14 In (**a**) is shown the error distribution for the books test image. We see that the majority of the error is concentrated below 1 pixel, although the distribution has a long tail. The root-mean-square error between the hardware emulator and the floating point versions is 3.09 pixels. The reason for this large value is evident in Fig. 13, in the noisy section in the top-right corner. Both algorithms report disparity estimates varying between ± 20 in this region when it should be closer to zero. In some cases the floating point algorithm returns $+20$ when the hardware emulation reports -20, and *vice versa*. This leads to outliers in the error distribution, as can readily be verified from (**b**)

confirming that there are indeed localized regions in which the error is quite large, but that it is otherwise quite small.

In order to provide a further comparison of the hardware implementation with the original, floating-point LWPC implementation, we used a standard test set of stereo-pair images provided by Scharstein and Szeliski [24].[3] These data sets also allow for a direct comparison against other stereo implementations. The results from three of these tests are shown in Figs. 15 and 16, and Table 4. It can be seen that the hardware implementation provides performance that is comparable to the floating point version for the Sawtooth and Map images, although it performs somewhat worse for the Tsukuba and Venus images. An error image and its associated error density for the Tsukuba image is shown in Fig. 16, and it is seen that much of the error again comes from attempting to match occluded pixels. The figures in Table 4 do not include occluded pixels (Fig. 16 does), but shows that nontextured regions and depth discontinuities are still problematic, as is to be expected with a correlation-based algorithm. For example, the lamp shade has very low texture, and can be seen to have large disparity errors in Fig. 16b). In all cases similar performance is found in the discontinuity regions: while the performance is not the best in either case, Scharstein and Szeliski admit that these regions are expected to be troublesome for pure "local" algorithms [24]. The hardware implementation is consistently poorer on the untextured regions, but this is not unexpected given the loss of fine detail through the fixed precision of the hardware system. In comparison with other algorithms, the hardware implementation delivered performance that was comparable to [19] and the Dynamic Programming methods [24] for the Venus and Map test images and [17]

[3] These images, along with a numerical comparison with leading stereo algorithms for the same data sets, can be found at http://bj.middlebury.edu/ schar/stereo/web/results.php.

Table 4 Numerical results from the standard test images [24] from Fig. 15[a]

Tsukuba			Sawtooth			Venus			Map	
All	Untex	Disc	All	Untex	Disc	All	Untex	Disc	All	Disc
LWPC										
14.16	14.46	38.98	5.15	5.95	23.54	6.32	10.59	26.68	3.54	17.31
Hardware LWPC										
19.59	24.90	37.62	6.93	11.71	22.68	10.51	18.11	31.52	3.87	20.12

[a]Values indicate percentage of pixels with disparity error above one pixel.

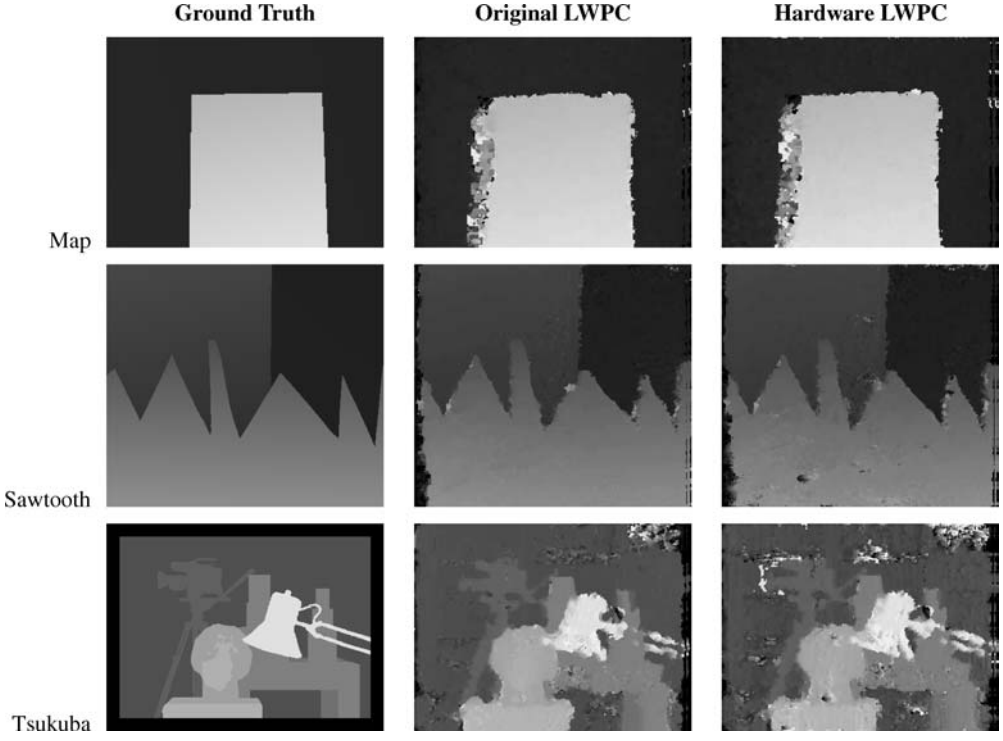


Fig. 15 The comparative results of running the original LWPC algorithm along side the hardware for three images from Scharstein and Szekiski's standard test set [24]. The ground-truth disparities are shown in the first column. While the hardware results are "noisier" than the original algorithm, it is usually in regions where the original algorithm encountered difficulty. Numerical results are shown in Table 4

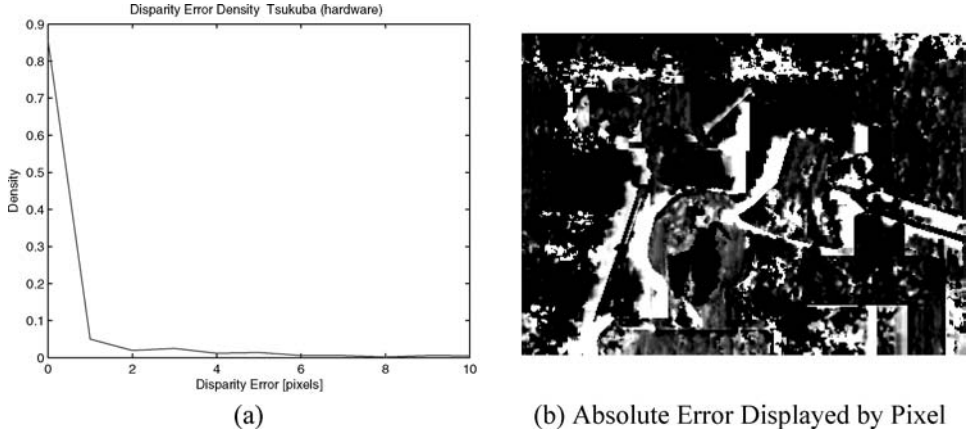


Fig. 16 In (**a**) is shown the error distribution for the Tsukuba test images. Again, the majority of the error is concentrated below 1 pixel, and the distribution tail is again long. The absolute disparity error is shown in (**b**), with black representing zero (no error) and white representing maximum error (10 pixels). It is evident that the algorithm has had difficulty at left-occlusion boundaries (the left image is used as reference), due to trying to match pixels that do not exist in the right image. The algorithm essentially guesses in these cases, giving rise to the distribution's rather flat tail. The root-mean-square error between hardware and the ground truth is 1.59 pixels

for the Venus image. This is significant since none of these are hardware implementations and all are much slower. No comparative data is known for the commercial stereo systems.

7 Conclusion

In summary, it should be seen that our system compares favorably with the systems described in Sect. 3. While some of the systems are starting to achieve frame rates above 5 fps, none of them are doing the amount of computation that our system is, and we expect our effective accuracy (assuming rectified images) to be significantly better. For example, while others are using correlation methods, none are computing complex filter responses at multiple scales and orientations, and then recovering and correlating the phase responses. While others are using image pyramid approaches, they use the pyramid to simplify computation at subsequent scales, as opposed to combining the information across scales. Finally, while we have attempted to compare running speeds of other algorithms based on the same size of image that our system currently processes, it must be mentioned that we are able to handle larger images at 30 fps if the algorithm is moved to larger FPGAs. Speed is thus dependent on the logic capacity of the FPGA, and not just its clock speed. FPGAs with capacities greatly exceeding that of the FPGAs used in this work now exist.

This paper demonstrates the feasibility of implementing challenging vision applications on FPGAs to achieve frame-rate performance. It illustrates the trade-offs required for a fast and efficient hardware implementation. Although this paper is specifically focused on a particular phase-based stereo-vision FPGA implementation, most of the design issues are common among other DSP and image-processing applications. Many of the components of this system can be reused in other vision algorithms. prefiltering and building the image pyramid are basic operations used by many algorithms doing tasks such as stereo, motion analysis and object recognition and localization. In particular, our implementation of steerable filters has great potential for reuse. Examples of algorithms based on these filters include optic flow computation [41, 42], object tracking [43] and recovery of rotation and illumination invariant features [44], which can be used for object tracking and/or object recognition. The implementation of the correlation units can be adapted for use in other systems that require correlation, such as template matching.

Finally, it must be acknowledged that the LWPC algorithm is not among the top performers in the Middlebury comparative results. At the time this system was conceived this comparative data was not available, and had it been a different choice of algorithm might have been considered. However, in defense of the system, it has given rise to a number of components that can be reused in other vision systems (as noted in the preceding paragraph).

7.1 Future work

A number of avenues for future work exist. The most obvious is porting this system to larger FPGA devices that would allow for restoring the third orientation to the Scale/Orientation unit, add preprocessing to rectify the image, and postprocessing to perform validation tasks such as left–right/right–left validation on the output. The size of images that can be processed will be increased, and computation of a *confidence* measure for each disparity estimate should be included. This measure can be based in part on estimates of local texture energy and probability of false matches. Given the increase in capacity in newer FPGA devices these goals should not be difficult to accomplish.

Another improvement is to increase the range of allowed disparities, and this work (along with the items mentioned in the previous paragraph) has already begun [45]. Although this could be achieved by merely increasing hardware resources, there are other possible solutions. One solution to increasing the system's disparity range lies in preshifting the phase correlation to cover the expected range of disparity for a particular section of the image. Between successive frames, the expected disparity is not expected to change rapidly, so disparity values from the previous frame could be used to determine the required amount of prewarping for individual pixels. At present the system has no memory of the results from the previous frame when starting computations on a new image pair.

We would also like to look into methods to automate the search for optimal fixed-point widths. This could be done through exhaustive search over a predetermined range of bit-widths, or through a simplified search where each stage in the pipeline is optimized on its own, assuming previous stages have already been optimized and subsequent stages are still in floating point (this approach lends itself well to the hardware-emulation stage). Also, it should be possible to simultaneously optimize with respect to target device space limitations; this will require development of integrated fixed-point width search and synthesis/place & route algorithms.

Finally, recent advancements in FPGA technology have made it possible to include simple processor cores to run software within an FPGA device. Such processors can be customized for a particular application, and are well suited for higher-level control tasks. The application of such cores to implementations of vision algorithms warrants further investigation.

Acknowledgements The authors would like to thank Marcus Van Ierssel and David Galloway for their technical assistance during the course of this work, and Allan Jepson for his thoughts and suggestions. The authors would also like to acknowledge financial support from Micronet R&D and NSERC.

References

1. Fleet, D.J.: Disparity from local weighted phase correlation. Int. Conf. Syst. Man Cybernetics **1**, 48–54 (1994)

2. Xilinx Inc.: Design tools center, cited May 3, 2005. [Online]. Available: http://www.xilinx.com/products/design_resources/design_tool/index.htm
3. Altera Corporation: DSP Builder, cited May 3, 2005 [Online]. Available: http://www.altera.com/products/software/products/dsp/dsp-builder.html
4. Fung, J., Mann, S.: Using multiple graphics cards as a general purpose parallel computer: Applications to computer vision. In: Proceedings of the 17th International Conference on Pattern Recognition, Cambridge, UK, pp. 805–808 (2004) online: http://www.eyetap.org/papers/docs/procicpr2004/
5. Yang, R., Pollefeys, M.: Multi-resolution real-time stereo on commodity graphics hardware. In: Proceedings of the 2003 IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, Madison, Wisconsin, pp. 211–218 (2003) [Online]. Available: http://citeseer.ist.psu.edu/article/yang03multiresolution.html
6. van Ierssel, M.: TM-3 documentation (2002) [Online]. Available: http://www.eecg.toronto.edu/ tm3/
7. van Ierssel, M., Galloway, D., Chow, P., Rose, J.: The Transmogrifier 3-a: Hardware and software for a 3 million gate rapid prototyping system. In: Micronet Annual Workshop (2001)
8. Xilinx Inc.: Xilinx data sheets (2002) [Online]. Available: http://direct.xilinx.com/bvdocs/publications/ds022-1.pdf
9. Faugeras, O., Hotz, B., Mathieu, H., Viéville, T., Zhang, Z., Fua, P., Théron, E., Moll, L., Berry, G., Vuillemin, J., Bertin, P., Proy, C.: Real time correlation-based stereo: Algorithm, implementations and applications. INRIA Sophia Antipolis, Tech. Rep. Research Report 2013 (1993) [Online]. Available: ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RR/RR-2013.ps.gz
10. Hou, K.M., Belloum, A.: A reconfigurable and flexible parallel 3d vision system for a mobile robot. In: IEEE Workshop on Computer Architecture for Machine Perception New Orleans, Louisiana (1993)
11. Kanade, T., Yoshida, A., Oda, K., Kano, H., Tanaka, M.: A stereo machine for video-rate dense depth mapping and its new applications. In: Proceedings of the 15th IEEE Computer Vision & Pattern Recognition Conference, pp. 196–202. San Francisco (1996)
12. Woodfill, J., Herzen, B.V.: Real time stereo vision on the parts reconfigurable computer. In: 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 201–210 (1997)
13. Zabih, R., Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In: Proceedings of the 3rd European Conference on Computer Vision, pp. 150–158 (1994) http://www.cs.cornell.edu/rdz/Papers/Archive/neccv.ps, http://www.cs.cornell.edu/rdz/Papers/Archive/nplt-journal.ps.gz
14. Burt, P.J.: A pyramid-based front-end processor for dynamic vision applications. In: Proceedings of the IEEE **90**(7), 1188–1200 (2002)
15. van der Wal, G., Hansen, M., Piacentino, M.: The acadia vision processor. In: Proceedings of the Fifth IEEE International Workshop on Computer Architecture for Machine Perception, pp. 31–40. Padua, Italy (2000)
16. Hirschmüller, H., Innocent, P.R., Garibaldi, J.: Real-time correlation-based stereo vision with reduced border errors. Int. J. Comput. Vis. **47**(1–3), 229–246 (2002), stereo, intensity correlation, MMX, fast
17. Mühlmann, K., Maier, D., Hesser, J., Anner, R.M.: Calculating dense disparity maps from color stereo images, an efficient implementation. Int. J. Comput. Vis. **47**(1–3), 79–88 (2002), stereo, intensity correlation, MMX, fast
18. Mulligan, J., Isler, V., Daniilidis, K.: Trinocular stereo: A real-time algorithm and its evaluation. Int. J. Comput. Vis. **47**(1–3), 51–61 (2002), stereo, trinocular, DFW-V500
19. Sun, C.: Fast stereo matching using rectangular subregioning and 3d maximum-surface techniques. Int. J. Comput. Vis. **47**(1–3), 99–117 (2002)
20. Veksler, O.: Stereo correspondence with compact windows via minimum ratio cycle. IEEE Trans. Pattern Anal. Machine Intell. **24**(12), 1654–1660 (2002)
21. Burt, P., Adelson, E.: The laplacian pyramid as a compact image code. IEEE Trans. Commun. **31**, 532–540 (1983)
22. Meerbergen, G.V., Vergauwen, M., Pollefeys, M., Gool, L.V.: A hierarchical symmetric stereo algorithm using dynamic programming. Int. J. Comput. Vis. **47**(1–3), 275–285 (2002)
23. Birchfield, S., Tomasi, C.: Depth discontinuities by pixel-to-pixel stereo. Int. J. Comput. Vis. **35**(3), 269–293 (1999)
24. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Int. J. Comput. Vis. **47**(1–3), 7–42 (2002)
25. Point Grey Research: Point Grey Research homepage. cited May 3, 2005 [Online]. Available: http://www.ptgrey.com/
26. Videre Design: SRI small vision system, cited May 3, 2005 [Online]. Available: http://www.videredesign.com/svs.htm
27. Konolige, K.: Small vision systems: Hardware and implentation. In: Proceedings of the Eighth International Symposium on Robotics Research (Robotics Research 8), Hayama, Japan, pp. 203–212 (1997) [Online]. Available: citeseer.ist.psu.edu/konolige97small.html
28. Canesta Inc.: Canesta ep development kit. cited May 3, 2005. [Online]. Available: http://www.canesta.com/devkit.htm
29. Baker, H.H., Binford, T.O.: Depth from edges and intensity based stereo. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence, pp. 631–636. Vancouver (1981)
30. Ayache, N., Faverjon, B.: Efficient registration of stereo images by matching graph descriptions of edge segments. Int. J. Comput. Vis. **1**(2), 107–131 (1987)
31. Fleet, D.J., Jepson, A.D., Jenkin, M.R.M.: Phase-based disparity measurement. CVGIP: Image Understanding **53**(2), 198–210 (1991)
32. Wang, J.J.: Image matching using the windowed fourier phase. Int. J. Comput. Vis. **11**(3), 211–236 (1993), read
33. Freeman, W.T., Adelson, E.H.: The design and use of steerable filters. IEEE Trans. Pattern Anal. Mach. Intell. **13**(9), 891–906 (1991)
34. Trucco, E., Verri, A.: Introductory Techniques for 3-D Computer Vision. Prentice Hall (1998)
35. Chang, M.L., Hauck, S.: Precis: A design-time precision analysis tool. In: 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 229–283 (2002) [Online]. Available: http://www.ee.washington.edu/faculty/hauck/publications/PrecisFCCM2002
36. Darabiha, A.: Video-rate stereo vision on reconfigurable hardware. Master's thesis, Department of Electrical & Computer Engineering, University of Toronto (2003)
37. Andraka, R.: A survey of CORDIC algorithms for FPGAs. In: Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA '98), Monterey, CA, pp. 191–200 (1998) [Online]. Available: http://www.andraka.com/files/crdcsrvy.pdf
38. Valls, J., Kuhlmann, M., Parhi, K.K.: Evaluation of CORDIC algorithms for FPGA design. J. VLSI Signal Process. **32**, 207–222 (2002)
39. Fua, P.: A parallel stereo algorithm that produces dense depth maps and preserves image features. Mach. Vis. & Appl. **6**(1), 35–49 (1993), INRIA Research Report 1369
40. Gluckman, J., Nayar, S.K.: Rectified catadioptric stereo sensors. IEEE Trans. Pattern Anal. Machine Intell. **24**(2), 224–236 (2002), stereo, mirros, rectified
41. Fleet, D.J., Black, M.J., Jepson, A.D.: Motion feature detection using steerable flow fields. In: Proceedings of the 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 274–281. Santa Barbara (1998)

42. Fleet, D.J., Jepson, A.D.: Computation of component image velocity from local phase information. Int. J. Comput. Vis. **5**(1), 77–104 (1990)
43. Jepson, A.D., Fleet, D.J., El-Maraghi, T.F.: Robust on-line appearance models for visual tracking. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 415–422 Kauai, (2001) [Online]. Available: http://www.cs.toronto.edu/ tem/cvpr01.pdf
44. Carneiro, G., Jepson, A.D.: Local phase-based features. In: Proceedings of the 2002 European Conference on Computer Vision, vol. 1, pp. 282–296. Copenhagen, Denmark (2002)
45. Masrani, D.K., MacLean, W.J.: Expanding disparity range in an FPGA stereo system while keeping resource utilization low. In: The First IEEE Workshop on Embedded Computer Vision, CVPR 2005 (2005)