

Real Time Rectification for Stereo Correspondence

Khurram Jawed¹ John Morris^{1,2} Tariq Khan¹ Georgy Gimel'farb²

¹Electrical and Computer Engineering, ²Computer Science, The University of Auckland, New Zealand.
Email: [mjaw002@aucklanduni,j.morris@,g.gimelfarb@auckland].ac.nz

Abstract

Duplicating the full dynamic capabilities of the human eye-brain combination is a difficult task but an important goal because of the wide application that a system which can acquire accurate 3D models of a scene in real time. Such a system must be able to correct images to remove lens distortion and camera misalignments from high resolution images at video frame rates - 30 fps or better. The images then need to be matched to determine the distance to scene objects. We have constructed a system which uses reconfigurable hardware (FPGAs) to handle the very large number of calculations required and is capable of processing 1 Mpixel images for disparity ranges of ~ 100 (allowing $\sim 1\%$ depth accuracy) at 30 fps. This paper focuses on the use of lookup tables in the hardware to correct images in real time with latencies that are determined more by the quality of the optics and mechanical alignment than by calculation demand. Sample results from the full system (which uses the Symmetric Dynamic Programming Stereo matching algorithm) operating at 30 fps are also shown.

1 Introduction

A system that provides high resolution stereovision in real time will have wide application - ranging from collision avoidance and path planning in moving vehicles through monitoring arbitrary scenes for security purposes to flexible industrial control. We have successfully built a system that produces '3D movies' with 1Mpixel frames at 30 fps. With this system, disparity ranges of more than 100 can be handled, permitting depth resolution of 1% and precise control, path determination or accurate scene modeling. The system consists of three main modules: (a) rectification, (b) stereo correspondence and (c) scene interpretation. Although the individual computations required in the first two modules are basically simple, the large number of computations required when high resolution images (our project target was at least 1 Mpixel) stream in from a pair of cameras at 30 fps.

The rectification module removes distortion and misalignment of images to simplify stereo matching. The correspondence module determines the most likely matches between pixels in the left and right camera images and outputs the disparity - the shift in pixels between corresponding points in the left and right images. Four 'images' are then fed to a host computer (corrected left and right images, disparity map and an occlusion map, which identifies pixels for which no good match can be detected, *i.e.* pixels which are not visible in both cameras and for which no direct depth

measurement is possible). The host computer is then responsible for the (application specific) interpretation of these 'images' and determination of strategies for navigation, control - or just the excitation of alarms for dangerous situations. Dataflow through the system is shown in Figure 1.

Related work Distortion removal and rectification has been extensively studied. For distortion removal, it is usual to assume a simple radial distortion model with the addition of tangential distortion to allow for misalignment of lenses with optical axes [1]. Most work has focussed on devising simple and efficient calibration techniques which determine the parameters of the distortion model, but Gribbon *et al.* proposed a distortion removal method based on lookup tables which could be implemented in an FPGA [2]. This work extends that work to include correction of camera misalignment and describes the incorporation of the rectification module into a full stereovision system: it focuses on the rectification module and only briefly covers the correspondence module, which is described in detail elsewhere [3]. Some sample output from the full system is shown in Section 6.

2 Rectification Module

For effective real time control, threat analysis or danger warnings, latencies in the system must be reduced to the minimum. In particular, a tradi-

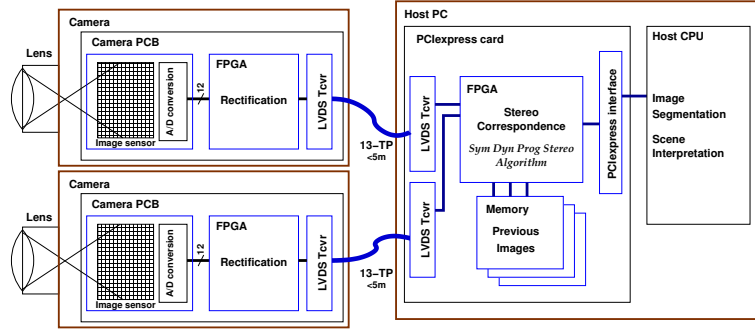


Figure 1: Block diagram of our final system showing small FPGAs associated with each camera.

tional approach which acquires images, then starts to process them, incurs a latency of $\sim 30ms$ plus the image interpretation time (which could add another $30ms$) at 30fps. A state-of-the-art processor can do a lot of work in $30ms$ to detect hazards or determine optimal paths earlier. Thus our system design attempts to reduce latency to much less than a frame time: with good optics and mechanical alignment, we are able to reduce it to several scan lines $\lesssim 1ms$ for 1 Mpixel images.

2.1 Calibration

An off-line calibration procedure was used to determine the distortion model parameters. In principle, any one of the plethora of calibration procedures in the literature could be used. We imaged a plane chequer board in ~ 25 different poses and positions in the scene and used the functions from the OpenCV `cvCalibFilter` class [4] to determine distortion parameters [5] and system extrinsic parameters [1]. Manual checks that corresponding points lie on the same scan line in corrected images showed that the procedure reduced distortion and misalignment errors to less than 1 pixel over the whole image.

Conventional (*i.e.* software) rectification procedures solve the equations for the distortion model using a simple, but slow, iterative approach. The lookup tables needed to support the rapid rectification procedure described in the next section are generated as part of the calibration procedure: the calibration program emits tables which are directly incorporated into the rectification circuit when it is synthesized.

3 Real time rectification

The rectification module (Figure 2) consists of lookup tables, two pixel address generators, a scan line buffer address generator, a scan line buffer and an intensity calculator. It removes lens distortion and camera misalignment in a single step. The

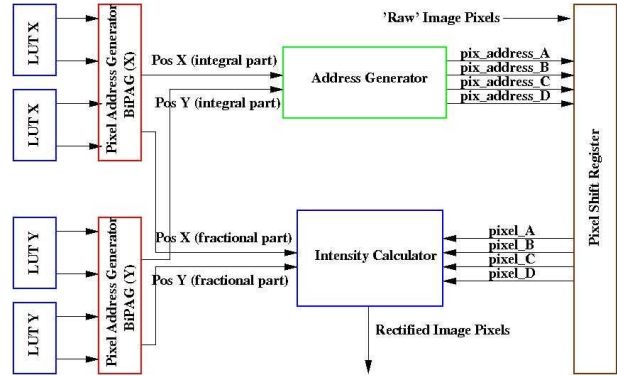


Figure 2: Rectification module: Pixel Address Generator interpolates lookup table entries to generate distorted image addresses; Address Generator converts a base address to addresses in the pixel shift register for four neighbours; Intensity Calculator calculates intensity by bilinear interpolation

calibration procedure determines displacements of pixels in the distorted image from their ideal positions if the cameras were perfectly aligned pinhole cameras. These displacements are stored in lookup tables which become part of the rectification circuit. The rectification module waits for sufficient pixels to be buffered to compensate for the worst displacement in any point of the distorted image. Then, for each pixel of the ‘ideal’ image, it computes the actual position from four points in the lookup table (see Figure 4), fetches the four nearest neighbours of the target point in the distorted image, compute the intensity of the ideal image pixel by bilinear interpolation.

3.1 Rectification

Akeila and Morris showed that a lookup table based approach could produce real-time performance in a compact circuit efficiently implemented on an FPGA [6]. The system buffers all the incoming pixels in a shift register, until it can start producing rectified images. This delay (and the size of the shift register) is determined by the pixel having the largest displacement in the undistorted

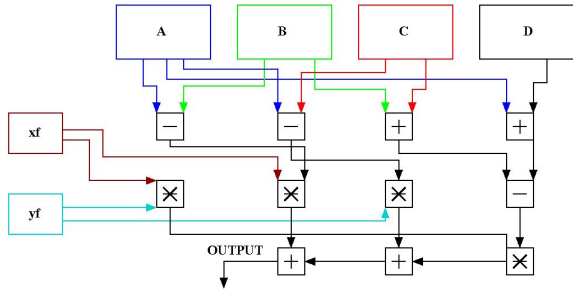


Figure 3: Pixel Address generator fetches the four values A, B, C and D from the lookup tables and using the offsets x_f and y_f computes the final output. Essentially the same circuit is used by the intensity calculator where A,B,C and D are the four neighbour pixel intensities and x_f, y_f are the fractional parts of the displacements.

image. Since the displacement lookup tables have been reduced to save space, the Pixel Address Generator must load the displacement values for the four gridpoints surrounding the current point and determine the actual displacement for the current location by bilinear interpolation. Since the displacement from the ideal position to the position in the distorted image will, in general, point to a location which is not conveniently in the centre of any pixel. The Pixel Address Generator separates the displacement into integral and fractional parts. The integral part is fed to the Address Generator which generates addresses into the shift register for the four neighboring pixels which are fetched and passed to the intensity calculator. The fractional part is fed directly to the intensity calculator.

3.2 Lookup Table Reduction

The lookup tables that are generated by the calibration procedure are too big to fit directly into the memory available on state-of-the-art FPGAs. However, since the distortion and misalignment correction is a smooth function, it is not necessary to store the full table. The tables are iteratively reduced until the maximum error introduced by interpolation exceeds a tolerance level defined by the intensity calculator - the final interpolated intensity should differ by less than 1 intensity unit from the intensity that would be calculated by using the full lookup table. For the experiments reported here, it was possible to reduce the original 1024×768 entry lookup table to one with 65×65 entries.

3.3 Pixel Address Generator

The Pixel Address Generator (PAG) uses bilinear interpolation to compute the position of the pixel in the distorted image. It need four values to compute the pixel position:

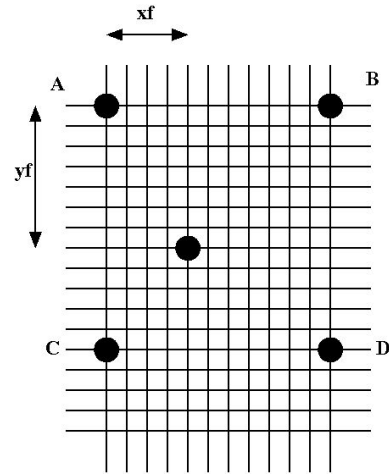


Figure 4: Address computation in the PAG module. A,B,C and D are the four displacement values from the lookup table. The current position is x_f, y_f .

$$d = (B - A)x_f + (C - A)y_f + (D + A - B - C)x_f y_f \quad (1)$$

where d is the output value for the x (y) displacement, A, B, C and D are the values read from the x (y) lookup table and x_f and y_f are the x and y positions in the current 'cell' - see Figure 4. Because the FPGAs we are using do not provide four-port memory, we have replicated the data in two dual-port memories for both X and Y pixel positions. This allows us to load the four values needed in one cycle and use the bilinear interpolation circuit(see Figure 3) to compute the pixel position. The lookup tables are already so small (typically a $\sim 65 \times 65$ entry table will suffice), so this solution uses a small amount of memory but simplifies the interpolation circuitry.

3.4 Address Generator

The address generator uses the integral part of the pixel position and calculates the address of the four neighboring pixels in the pixel shift register. Pixel Shift Register is implemented using three-port memory(one write port, two read ports).Two pixels are read in the first cycle and the two other pixel are read in the second cycles. These four pixels are then sent to the intensity calculator, along with the fractional part of the pixel position, to generate the final rectified pixel.

3.5 Intensity Calculator

The intensity of the pixel in the 'ideal' image is computed by bilinear interpolation from the intensities of four neighbouring pixels - see Figure 4. These pixels are fetched from the pixel shift register by the addresses generator. The fractional positions, x_f and y_f , are calculated by the Pixel Address generator. The intensity calculator also



Figure 5: Pair of rectified images: Inspection shows that corresponding points lie on the same scanline, satisfying the epipolar constraint.

Table 1: Rectification module resources for various pixel widths and scanlines; pixwth = pixel width; scanline length = 1024 pixels; SL = number of scan lines; DSP = number of DSP blocks; Mem = Memory bits

pixwth (bits)	SL	Logic (LEs)	Regs	DSP	Mem (bits)
12	64	1167	309	27	1,877,064
12	32	963	287	27	1,090,632
12	16	931	277	27	697,416
12	8	895	267	27	500,808
10	64	1003	283	26	1,614,920
10	32	933	272	26	959,560
10	16	902	263	26	631,880
10	8	869	253	26	468,040
8	64	901	259	23	1,342,776
8	32	878	249	23	828,000
8	16	349	239	23	566,344
8	8	901	259	23	435,272

All configurations will fit on a small Altera Cyclone FPGA.

evaluates Equation 1 (where A, B, C and D are neighbour pixel intensities) and uses essentially the same circuit as the Pixel Address Generator - see Figure 3 - to compute the intensity of the rectified image pixel.

Camera misalignment means that the rectified images have missing edges of various shapes and sizes - mainly determined by mechanical misalignment, but lens distortion also contributes. It was found necessary to extend ‘valid’ pixel intensities into these regions to prevent the correspondence algorithm from producing false matches on these very strong edges. The images in Figure 5 show the success of the rectification process: the extended edges are also recognizable.

3.6 Rectification Subsystem Architecture

For each camera, distortion can be removed independently. Rectification of images to those taken by cameras perfectly aligned to the optical axes of the canonical configuration (parallel optical axes perpendicular to the baseline; images planes parallel and scan-lines collinear) can also be performed

independently on both cameras. Thus our final system will consist of an image sensor and a small FPGA directly coupled to it in the camera head connected by CameraLink-style twisted pairs to the main FPGA in the host processor [7]. For the results shown in Section 6, we used CameraLink cameras connected to an Altera FPGA (on a Gidel ProcStar III card) in the host and placed the rectification modules on this FPGA.

4 Stereo Correspondence

An almost complete spectrum of stereo correspondence algorithms has been published and it is still augmented at a significant rate! Scharstein provides a dynamic catalogue and performance review [8]. The best algorithms in terms of matching performance [8]¹ perform global optimizations and are generally slow, so that they are unsuitable for real time performance. On the other hand, correlation algorithms, being regular, simple and local in nature, are good candidates for hardware implementations, but they have poor matching performance. Dynamic programming algorithms have better matching performance: they achieve this by averaging over scan lines and are thus potentially well suited to systems which stream pixels from cameras scan line by scan line over serial links. Inter-scan line smoothing [9] enhances performance, but, due to the resource cost, we have not considered such optimizations yet, but we have added additional memory to the FPGA card that we are designing to allow this to be tested. Our system uses Gimel’farb’s Symmetric Dynamic Programming Stereo (SDPS) algorithm [10] which has an extremely compact (and thus readily replicated many times to handle large disparity ranges, Δ) basic calculation block, see Figure 7. The SDPS algorithm exploits constraints on visibility state changes in the Cyclopæan view to save storage in the predecessor array - a large and unavoidable evil in any dynamic programming approach - making it superior to other dynamic programming implementations [11, 12].

An SDPS implementation is even more compact than an, ostensibly much simpler, correlation algorithm [13], because of the large number of adders needed to sum over correlation windows. A brief outline of the SDPS algorithm follows: more detailed descriptions are available [14, 15, 16].

4.1 Symmetric Dynamic Programming Stereo Algorithm

The SDPS algorithm places a virtual camera midway between the left and right cameras - this Cy-

¹At the date of writing: the Middlebury table changes regularly.

Table 2: Notation

d	Disparity
Δ	Maximum disparity
$\mathbf{g}^{\mathbf{L R}}$	Array of pixel intensities from L R images
x	Pixel index in a Cyclopæan image
\mathbf{g}	Array of pixel intensities in Cyclopæan image
$x_{L R}$	Pixel index in a L R image
MR	Monocularly visible (Right)
ML	Monocularly visible (Left)
B	Binocularly visible
\mathbf{c}	Cost array
π	Predecessor array

clopæan ‘eye’ views a scene half-way between that viewed by the left and right cameras. A very useful consequence of taking this view is that transitions between disparity levels must pass through monocularly visible states. The algorithm naturally detects occlusions and our system outputs them. As can be seen in the results (Figure 9), occlusions neatly outline objects in the scene: we expect that the occlusion map will provide a very fast way to outline and identify scene objects as it is a very simple map with elements which represent the three possible visibility states: B - Binocularly visible; MR - Monocular right (visible only by the right camera) and ML - Monocular left.

Figure 8 shows the canonical arrangement, with a Cyclopæan image (CI) in the centre (green rays). A simple profile is shown, with the visibility state of each point on the profile marked. Looking at Figure 8, you can see that any large change in disparity along a profile will be accompanied by at least one ML or MR point for each unit change in disparity.

For simplicity, we assume

- that allowable disparity values d lie in the range $0.. \Delta$.
- a canonical stereo configuration (parallel optical axes and image planes with collinear scan lines), so that matching pixels are always found in the same scan line² and
- cameras are electrically and optically matched.

We process each scan line in turn, so that the y index in the pixel array is always constant. This allows efficient processing of images streamed from cameras pixel by pixel along scan lines. To reduce subscript clutter in the expressions following, the y index is dropped and we consider an - effectively

²Concurrent work in our laboratory has resulted in efficient techniques which can remove distortion and rectify images with low latency [6].

one-dimensional - array of pixels from the left image, $\mathbf{g}^{\mathbf{L}} = \{g_i^{\mathbf{L}} | i = 0..w - 1\}$ and the right image, $\mathbf{g}^{\mathbf{R}} = \{g_i^{\mathbf{R}} | i = 0..w - 1\}$. These pixels are used to ‘construct’ a conjugate line in the Cyclopæan image,

$$\mathbf{g} = \{g_x | x = 0, 1, 2, 3, \dots, 2w - 1\}$$

Note that \mathbf{g} has twice as many points as $\mathbf{g}^{\mathbf{L|R}}$ and that we have chosen to use integral indices rather than the half-integral ones used by Gimel’farb [10]. For a point, x , in the Cyclopæan image which represents a conjugate pair, *i.e.* is in state, B (binocularly visible), the corresponding coordinates in the left and right images are:

$$x_L = \frac{x + d}{2} \quad x_R = \frac{x - d}{2}$$

Thus positions in the Cyclopæan image with even indices, $x = 0, 2, \dots$ represent points with even disparities and those with odd indices, $x = 1, 3, \dots$, odd disparities.

In a traditional dynamic programming approach, we fill two $2w \times 3(\Delta + 1)$ arrays. The cost array, \mathbf{C} , contains costs, $c_{x,d,s}$, for a ‘path’ from the pixel at $x = \Delta/2$ to pixel x in the Cyclopæan image ending at disparity, d , with visibility state, s , where $s \in \{MR, B, ML\}$. A predecessor array, π , stores pointers to the predecessor of each state in the cost array: it is used for constructing the final disparity list, with $2w - \Delta$ points per line. Note the double width disparity and occlusion maps in Figure 9. The circuit is driven by a master clock which is at least twice as fast as the pixel clock (90MHz for a 29MHz pixel clock in our prototype). Thus, for each incoming pixel, there are two computation phases: one computes even disparities and the other odd disparities. A similarity function, ΔI , measures pixel mismatches (a simple absolute difference function was used for these results but an adaptive function will be built into the next version [16]). New costs, c_{dxs} , are the sum of this mismatch and the best term from the two preceding cost array columns allowing for visibility constraints in the Cyclopæan image - fundamentally, each Cyclopæan image point is no more than one disparity unit different from its predecessor. Costs of paths are: to a binocularly visible point (B) -

$$c_{x,d,B} = \Delta I(x, d) + \min(c_{x-2,d,B}, c_{x-2,d,MR}, c_{x-1,d-1,ML}) \quad (2)$$

and to a monocular point (ML,MR) -

$$c_{x,d,ML} = \min(c_{x-1,d-1,ML}, c_{x-2,d,B}) + occ.term \quad (3)$$

$$c_{x,d,MR} = \min(c_{x-1,d+1,MR}, c_{x-1,d+1,B}) + occ.term \quad (4)$$

The index (x_{pr}, d_{pr}, s_{pr}) of the cost value chosen by the *min* operator is despatched to the predecessor array, π . x_{pr} and d_{pr} are uniquely defined by and computed from s_{pr} and s using a small set of rules, so only the previous visibility index, s_{pr} , is retained. This is a major advantage of the SDPS algorithm in a hardware implementation: s values can be encoded by two bits, saving a large amount of space in the predecessor array. Further saving is possible by compressing a set of values, $\{s_{MR}, s_B, s_{ML}\}$, into 5 bits as there are only 18 unique values, but the predecessor array used in our experiments to date easily fits in the FPGA available (see Section 6), so the additional circuit complexity was not warranted.

When two full scan-line have been received from the cameras, the back-track module activates. It starts with the last π value for which a full set of disparities can be computed³:

$$\pi(w-1, \arg \min_{d,s}(c_{2w-1,d,s}))$$

and generates a list of disparities for this scan line. Note that the disparity list is generated in reverse order, *i.e.* right \rightarrow left when the normal scan is left \rightarrow right, but this will usually have negligible effect on subsequent host processing because it is likely to take a more global view of the data sent to it and thus can work with a scans generated in any order. In the current implementation, 32-bit words are sent to the host which unpacks left and right image pixels as well as disparity and occlusion map components, so that the cost to re-arrange lines to a more natural order is negligible using the host caches.

5 Hardware design

Corrected pixels from the rectification module are fed to shift registers in the correspondence: a $\frac{\Delta}{2}$ entry shift register first delays right image pixels until they can be fed to the disparity calculators. Then left and right pixel streamed in a contra-rotation pattern through the disparity calculators. Left pixels enter the bottom ($d = (\Delta - 1)/\Delta$) disparity calculator while right pixels enter the top ($d = 0/1$) disparity calculator. This ensures that potentially matching pixels arrive in the appropriate disparity calculator in every cycle.

Although the cost array, \mathbf{C} , has $2w$ values for x , only $x-1$ and x are used at any time, so it suffices to use a pair of registers, one for $\mathbf{C}_{x-1,d,s}$ and the

³The symmetric Cyclopæan image misses two ‘edges’ of Δ pixels for which no corresponding pixels can be found. Contrast with conventional approaches which reconstruct the left (or right) image and cannot compute a band of Δ pixels on the left (or right) due to the absence of corresponding pixels.

other for $\mathbf{C}_{x-2,d,s}$. A total of only $2 \times (\Delta + 1)$ entries are actually needed. The cost register update block is clocked at twice the pixel clock rate: it computes even disparity values in one cycle and odd ones in the next. One half of a cost register block is shown in Figure 7: it computes costs for odd d in odd clock cycles. The circuitry to compute even d values is identical⁴. $\frac{\Delta}{2}$ such blocks are needed to evaluate all the entries in \mathbf{C} .

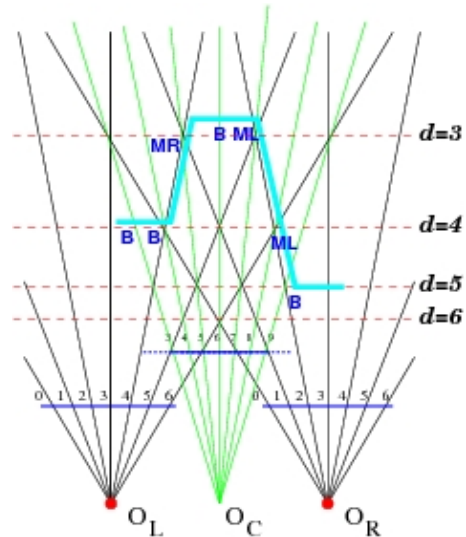


Figure 8: Camera configuration, including the virtual ‘Cyclopæan’ camera (centre) and the profile of a scene object.

Note that only a portion of the Cyclopæan image is shown - see the indices. The state (ML, B or MR) in which an object point appears in the Cyclopæan image is marked on the profile.

The largest storage cost occurs for the predecessor array, π : a total of $3w(\Delta + 1)$ values are needed. However, each entry requires 2 bits only to encode the three values which can be taken by s_{pr} . In the Cyclopæan image, a change in disparity must be accomplished by one or more monocularly visible points. For example, in Figure 8, the first disparity change has one MR points between B points, indicating a change, $\delta d = -1$. The next disparity change has two ML points between B points and $\delta d = +2$. Backtracking from right to left in Figure 8, one would start with a minimum in \mathbf{C} at $d = 5, s = B$. The predecessor is B, so at $x = \Delta - 2, d = 5$ also. The full sequence (from right to left) and the inferred values of d are:

⁴We could reduce the total number of adders in the circuit by re-use of the ‘odd’ circuits in an even cycle - at the expense of some multiplexors. However, the circuit is already compact (*cf.* Table 3) and is not limited by available FPGA sizes, so this was not tried.

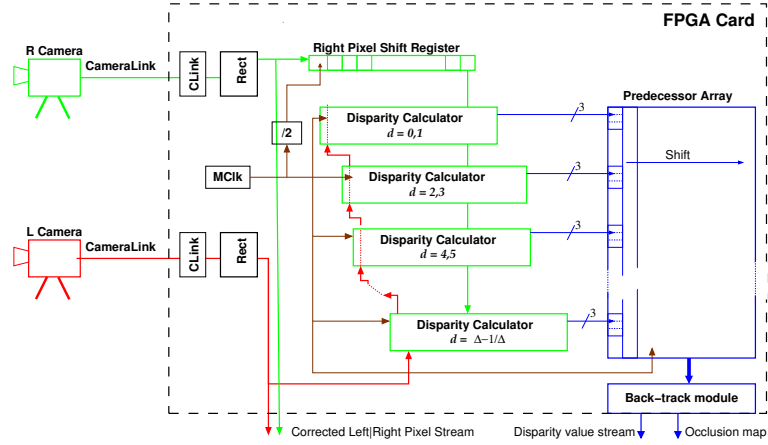


Figure 6: Overall layout of disparity map generator.

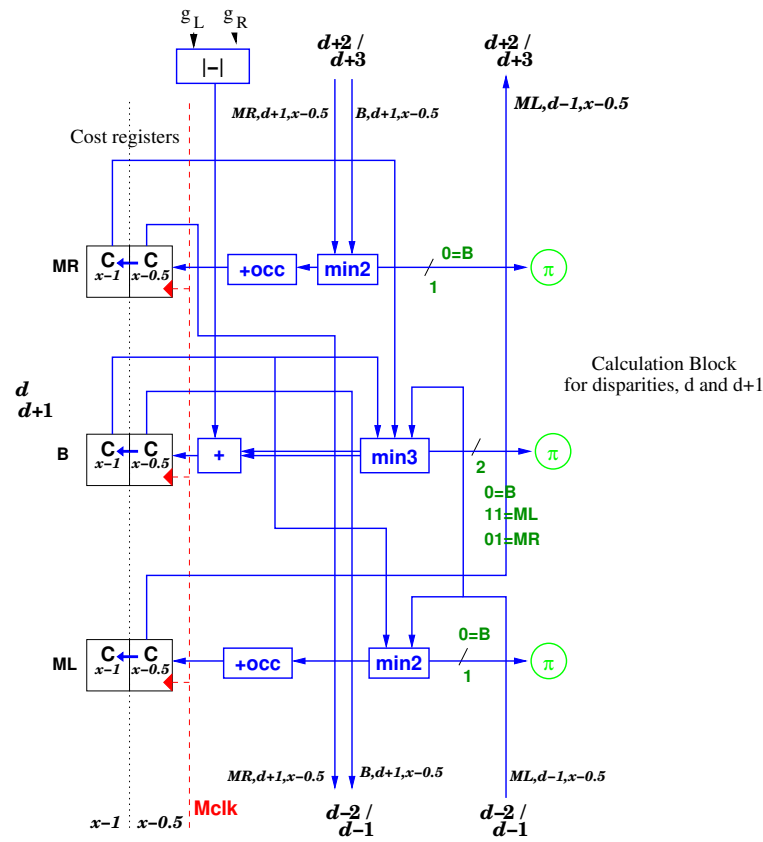


Figure 7: Disparity calculator for odd disparities, $d + 1$. This circuit is replicated (sharing registers) to calculate even disparities, d . $\frac{\Delta}{2}$ such blocks are instantiated in a fully parallel implementation - see Figure 6.

x	12-10	9	8	7	6	5	4	1
State,	B	B	ML	ML	B	MR	B	B
π_{pr}								
Infer'd d	5	5	(4)	(3)	3	(4)	4	4

For monocularly visible points, d values in brackets are the disparity to be assigned to the next binocularly visible point.

6 Results

Our current prototype system uses a Gidel Prostar III card with CameraLink interface and an Altera Stratix III FPGA. Two Sentech CL83A monochrome cameras (1024×768 pixel frames at 30fps) connect directly to the FPGA. The FPGA card communicates to a host PC via an 8-lane PCIexpress bus. Some images selected from video se-

Table 3: Selection of resource requirements

Pixel (bits)	Scan line (pixels)	Scan line buffer	Δ pixels	Logic Util ⁿ <i>out of</i>	ALUT 113600	Registers 113600	Memory (10 ⁶ bits) 5.6
8	1024	64	40	23%	17840	12204	3.5
8	512	128	40	22%	17736	12189	3.4
8	1024	64	64	28 %	24056	14315	3.6
8	1024	64	100	39%	17966	17966	3.8

quences of two scenes are shown in Figure 9. The first sequence shows our actor walking forward and placing a box in front of him. The actor becomes lighter as he approaches. Note that an outline of the main scene objects is clearly visible in the occlusion maps - even when he is not clearly visible in the disparity map. Note that this is mainly an illusion due to low image contrast: there is a clear disparity difference! In the second sequence a ball is thrown from the rear of the scene and our system tracks it in flight at 30fps: it becomes lighter as if flies forward.

Table 3 shows the resources required on the Altera FPGA for the *full system* - CameraLink interface, rectification module and correspondence module. The only resource which is near capacity is memory. Most of this is needed to buffer scan lines in the rectification module and the number of lines buffered is large - 64 or 128. With good mechanical alignment this can be reduced to less than 10: lens distortion contributes 2-3 lines at most. In our final system, the rectification module will be moved to the small FPGAs shown in Figure 1. The Altera '150' FPGA is only 'medium' size in the scale of today's technology, showing that, using the SDPS algorithm, we can either implement larger systems (higher disparity ranges) or add enhanced adaptive matching schemes [15].

7 Conclusions

The use of pre-generated lookup tables which are then synthesized into the 'soft' circuitry of an FPGA ensures that correction and alignment of images does not add significant latency to the full system. The critical factor in the latency - the number of scan lines that need to be buffered before correction can begin - is mainly determined by the lens quality and camera alignment accuracy. The correction calculations add less than 10 pixel clocks to the latency, whereas poor camera alignment can add $\sim 5 \times 10^4$ pixel clocks of latency! The SDPS matching algorithm also adds at most two scan lines of latency, 2×10^3 pixel clocks. Therefore, with our system, good quality optics and careful mechanical design become the key determinant of

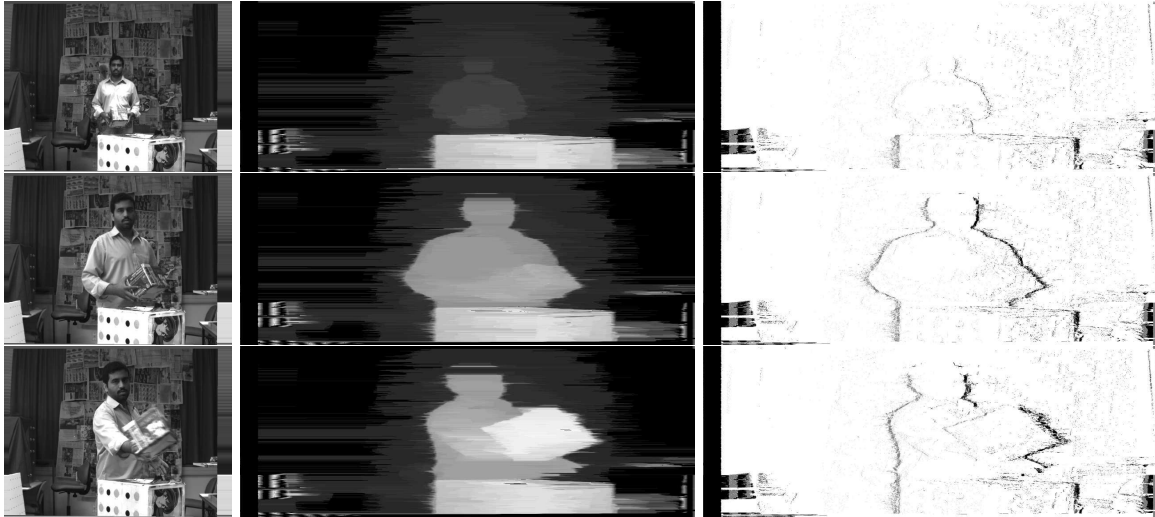
performance.

The use of dynamic programming techniques simplifies the correspondence section of our system - permitting larger disparity ranges (and thus higher depth accuracy) to be obtained in smaller FPGAs.

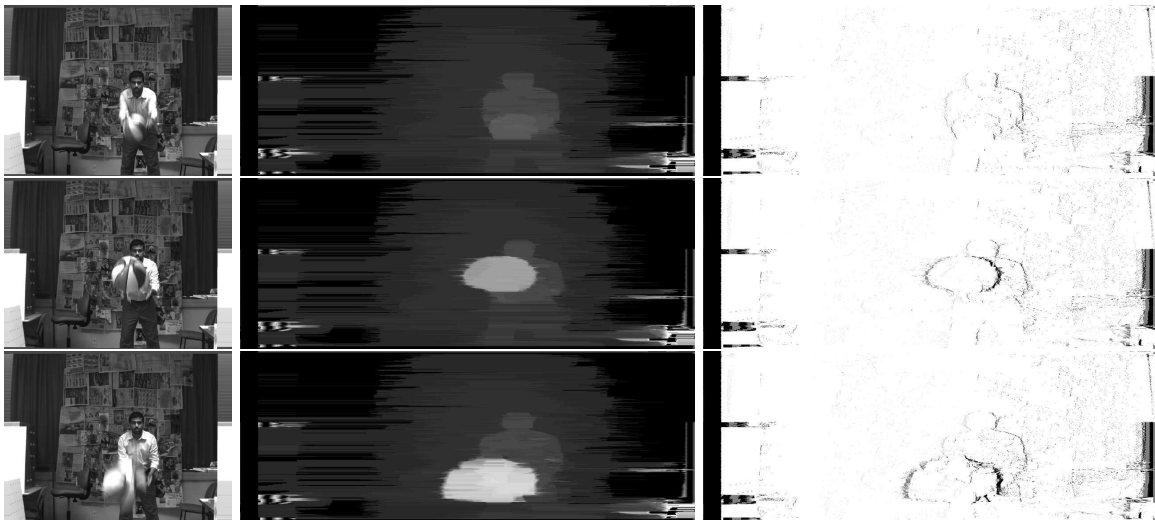
The overall design of our final system - with small FPGAs associated with each camera to remove distortion and align images - also presents a good match of required functions to technology. The small (and cost effective) FPGAs rectify images independently and can be programmed with the lookup tables that are appropriate to lenses and optical configuration needed for any specific application. The larger FPGA is used for matching, where a high degree of parallelism - readily attainable by replicating the small disparity calculators needed by the SDPS algorithm - is needed to attain real-time performance. Finally, almost all of the cycles of the host processor are freed of simple repetitive calculations needed in rectification and correspondence, and are available for the interpretation of the raw images, depth maps and occlusion maps. These calculations are often rule-based and global (needing large random access memories) and ideally suited to a state-of-the-art PC's general purpose processor.

References

- [1] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.
- [2] K. T. Gribbon, C. T. Johnson, and D. G. Bailey, "A real-time fpga implementation of a barrel distortion correction algorithm with bilinear interpolation," in *Proc Image and Vision Computing New Zealand*, 2003, pp. 408–413.
- [3] K. Jawed, J. Morris, G. Gimel'farb, T. Khan, and P. Medrano-Garcia, "High resolution real time stereophotogrammetry," in *CAIP, 2009*, 2009, p. submitted.



Box: Our model moves forward and places a box.



Ball: Figure at the rear (gently) tosses a basketball towards us.

Figure 9: Snapshots from video sequences captured in our laboratory: from left to right - left image; disparity map ($\Delta = 40$, white represents larger d - closer, black smaller d - farther); occlusion map (grey - ML, right occlusion, white - B, binocular, black - MR, left occlusion). Note that, in the Cyclopæan view, images have twice as many pixels per scan line. The full videos and additional sequences may be seen here:

<http://www.cs.auckland.ac.nz/~jmor159/HRPG/RT/index.html>

- [4] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.
- [5] D. C. Brown, "Close-range camera calibration," *Photogrammetric engineering*, vol. 37, no. 8, pp. 855-866, 1971.
- [6] H. Akeila and J. Morris, "High resolution stereo in real time," in *International Workshop on Robot Vision*, 2008, pp. 72-84. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78157-8_6
- [7] K. Jawed, J. Morris, and G. Gimel'farb, "Towards an intelligent vision processor," in *Proc Image and Vision Computing New Zealand, 2008*, K. Irie, Ed., 2008, pp. 1 - 8.
- [8] D. Scharstein and R. Szeliski, *Stereo Vision Research Page*. <http://cat.middlebury.edu/stereo/data.html>, 2005. [Online]. Available: <http://cat.middlebury.edu/stereo/data.html>
- [9] S. Birchfield and C. Tomasi, "Multiway cut for stereo and motion with slanted surfaces," *ICCV '99*, pp. 489-495, September 1999.
- [10] G. Gimel'farb, "Intensity-based computer binocular stereo vision: signal models and

- algorithms,” *Int J Imaging Systems and Technology*, vol. 3, pp. 189–200, 1991.
- [11] P.-W. Han and Y.-Y. Yang, “A new stereo matching hardware with merged odd-even PE architecture,” in *Proc IEEE Tencon*. IEEE Tencon, 1999.
- [12] S. Park and H. Jeong, “Real-time stereo vision FPGA chip with low error rate,” in *MUE*. IEEE Computer Society, 2007, pp. 751–756. [Online]. Available: <http://doi.ieeecomputersociety.org/10.1109/MUE.2007.180>
- [13] J. Yi, J. Kim, L. Li, J. Morris, G. Lee, and P. Leclercq, “Real-time three dimensional vision,” in *Proceedings of the 9th Asia-Pacific Computer Systems Architecture Conference*, ser. LNCS, J. Xue and P. Yew, Eds., no. 3189. Springer-Verlag, Sep. 2004, pp. 309–320.
- [14] G. L. Gimel’farb, “Probabilistic regularisation and symmetry in binocular dynamic programming stereo,” *Pattern Recognition Letters*, vol. 23, no. 4, pp. 431–442, 2002.
- [15] J. Morris and G. Gimel’farb, “Real-time stereo image matching system,” May 2, 2008, NZ Patent Application 567986.
- [16] J. Morris, K. Jawed, and G. Gimel’farb, “Fast hardware for stereo correspondence,” in *ACIVS’2009*, 2009, p. submitted.