Compressed Look-Up-Table based Real-Time Rectification Hardware

Abdulkadir Akin, Ipek Baz, Luis Manuel Gaemperle, Alexandre Schmid, and Yusuf Leblebici
School of Engineering (STI), Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland, firstname.surname@epfl.ch

***Abstract*—Stereo image rectification is a pre-processing step of disparity estimation intended to remove image distortions and to enable stereo matching along an epipolar line. A real-time disparity estimation system needs to perform real-time rectification which requires solving the models of lens distortions, image translations and rotations. Look-up-table based rectification algorithms allow image rectification without demanding high complexity operations. However, they require an external memory to store large size look-up-tables. In this work, we present an intermediate solution that compresses the rectification information to fit the look-up-table into the on-chip memory of a Virtex-5 FPGA. The low-complexity de-compression process requires a negligible amount of hardware resources for its real-time implementation. The proposed image rectification hardware consumes 0.28% of the DFF and 0.32% of the LUT resources of the Virtex-5 XCUVP-110T FPGA, it can process 347 frames per second for a 1024×768 pixels image resolution, and it does not need the availability of an external memory.**

Keywords—Stereo Matching; Image Rectification; Compression; Real-Time; Hardware Implementation; FPGA

I. INTRODUCTION

Disparity estimation (DE) is an algorithmic step that is applied in a variety of applications such as autonomous navigation, robot and driving systems, 3D geographic information systems, object detection and tracking, medical imaging, computer games, 3D television, stereoscopic video compression, and disparity-based rendering.

The stereo matching process compares the pixels in the left and right images and provides the disparity value corresponding to each pixel. If the cameras could be aligned perfectly parallel and if the lenses were without distortion, the matching pixels would be located in the same row of the right and left images. However, providing the perfect set-up is almost impossible. Lens distortion and camera misalignments should be modeled and removed by internal and external stereo camera calibration and image rectification processes [1].

Image rectification is one of the most essential pre-processing parts of DE. Nevertheless, many real-time stereo-matching hardware implementations [2-4] prove their DE efficiency using already calibrated and rectified benchmarks of the Middlebury evaluation set [5], while some do not provide detailed information related to the rectification of the original input images [6].

In a system that processes the disparity estimation in real-time, image rectification should also be performed in real-time. The rectification hardware implementation presented in [7] solves the complex equations that model distortion, and consumes a significant amount of hardware resources.

A look-up-table based approach is a straightforward approach that enables saving hardware resources [8-10]. In [8-10], the mappings between original image pixel coordinates and rectified image pixel coordinates are pre-computed and the pre-computations are used as look-up-tables. Due to the significant amount of generated data, these tables are stored in an external memory such as a DDR or SRAM [8-9]. Using an external memory for the image rectification process may cause an additional cost for the disparity estimation hardware system or impose additional external memory bandwidth limitations on the system. In [10], look-up-tables are encoded to consume 1.3 MB data for 1280×720 size stereo images with a low-complexity compression scheme. This amount of data requires at least 295 Block RAMs (BRAM) without considering pixel buffers, thus it can only be supported by largest Virtex-5 FPGAs or recent high-end FPGAs.

In this paper, we propose a novel compressed look-up-table based image rectification hardware, which requires a negligible amount of hardware resources for low complexity de-compression algorithm, and does not require using an external memory. The presented hardware requires 45 BRAMs for rectifying 1024×768 size stereo images, which enables its implementation into low-cost FPGAs.

II. TYPICAL LOOK-UP-TABLE BASED SOLUTION

Look-up-table based rectification methods can be distinguished by two different image warping flows: forward mapping and inverse mapping. Forward mapping computes the rectified target pixel locations considering the given pixel locations in the original image. Inverse mapping computes the original source pixel locations considering the given pixel locations in the rectified image. The mapping requires separate tables for X and Y coordinates, and for the right and left images. Therefore, four tables are required. The formulations for forward and inverse mappings are presented in equations (1) and (2), respectively. In these equations, *ForwT* is the forward mapping table, *InvT* is the inverse mapping table, *Ori* represents the original image taken from the camera, *Rec* represents the rectified image. Y_{Rec}, X_{Rec}, Y_{ori} and X_{ori} represent the Y and X coordinates.

978-1-4799-0524-9/13/$31.00 ©2013 IEEE

$$Forward: (Y_{Rec}, X_{Rec}) = \left(ForwT_y(Y_{Ori}, X_{Ori}), ForwT_x(Y_{Ori}, X_{Ori})\right)$$
$$Rec_{(y,x)} = linear\_interpolation(nearest\ neighbours\ of\ Rec_{(y,x)}) \quad (1)$$

$$Inverse: (Y_{Ori}, X_{Ori}) = \left(InvT_y(Y_{Rec}, X_{Rec}), InvT_x(Y_{Rec}, X_{Rec})\right)$$
$$Rec_{(y,x)} = linear\_interpolation(nearest\ neighbours\ of\ Ori_{(Y_{Ori}, X_{Ori})}) \quad (2)$$

A typical rectification process utilizes fractional pixel precision which requires the linear interpolation of four pixels. The linear interpolation schemes for forward and inverse mappings are represented in Figure 1 and Figure 2, respectively. The linear interpolation process for forward mapping is more complex than the linear interpolation process of inverse mapping, since it requires additional computation and an intermediate memory consumption to find the closest target pixels in the rectified image. The look-up-table based rectification hardware architectures presented in [8-10] use the inverse mapping due to its simplicity.

The size of the look-up-table depends on the size of the rectified image and the fractional precision. For example, for the rectification of 1024×768 resolution stereo images with 6 bits fractional precision, only the rectification map requires approximately 6 MB of space in a memory. This amount of data is excessive to fit into the on-chip memory of a mid-range FPGA. Therefore, dumping look-up-tables into an external memory is preferred in the hardware implementations of [8-9].

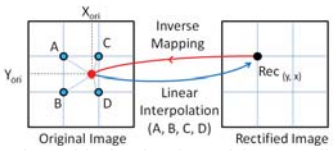


Fig. 1. Inverse mapping with fractional precision coordinates. Corners indicate integer pixel coordinates.

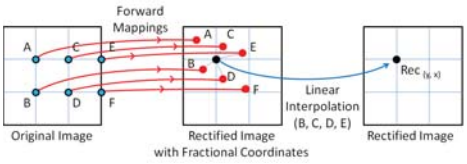


Fig. 2. Forward mapping with fractional precision coordinates.

III. Proposed Compression Algorithm for Rectification Mappings

In contrast to the selection of the hardware implementations of [8-10], a forward mapping based rectification scheme is selected for the proposed compressed look-up-table based rectification (CLUT-R) algorithm. In CLUT-R, fractional precision is ignored. Therefore, CLUT-R causes performance loss in the disparity estimation. This performance loss is evaluated and its negligible distortion is analyzed in section V. Ignoring fractional precision allows an efficient compression scheme.

The compression scheme is presented in the flow graph in Figure 3. The proposed compressed rectification algorithm produces four compressed tables. The compression scheme requires eight steps. The details of steps 1-2 can be found in [1]. The details of steps 3-8 are detailed in this section.

In the third step, integer coordinate precision forward mapping is extracted from the fractional precision inverse mapping. The extraction scheme is demonstrated in Figure 4. The example original and rectified pictures have a size of 4×5 pixels. First, inverse mapping is applied to find the fractional source pixel locations of all pixels in the rectified image. Due to the 3D rotation, some of the pixels in the rectified image cannot be related to their source pixels in the 4×5 original image, as shown in Figure 4(a). The nearest integer coordinates of all fractional source coordinates are computed, and they are targeted onto the integer pixel coordinates in the rectified image, as presented in Figure 4(b). Thus one-to-one mapping is provided in the third step.

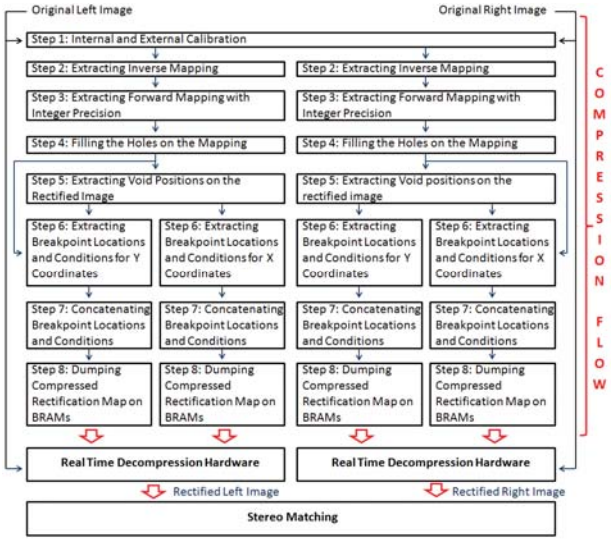


Fig. 3. The flow chart for the proposed compressed look-up-table based stereo image rectification process.

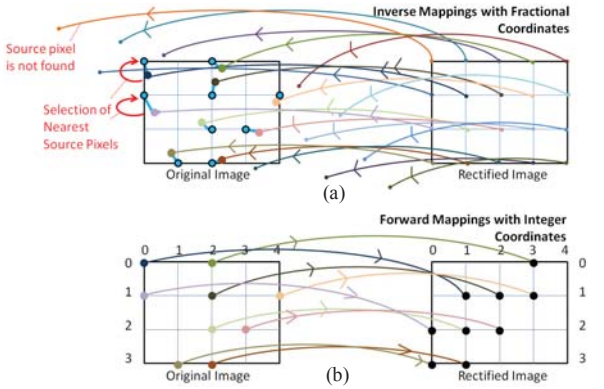


Fig. 4. The third step of the compression flow (a) selection of nearest source pixels from fractional inverse mapping (b) extraction of forward mapping with integer coordinates.

	0	1	2	3	4
0	1	NT	0	NT	NT
1	2	NT	1	NT	1
2	NT	NT	2	2	NT
3	NT	3	3	NT	NT

(a)

	0	1	2	3	4
0	1	NT	3	NT	NT
1	0	NT	2	NT	3
2	NT	NT	1	2	NT
3	NT	0	1	NT	NT

(b)

Fig. 5. Integer coordinate precision forward mapping look-up-tables after the third step. Regular orders are shown with red ellipses (a) mapping of Y coordinates (b) mapping of X coordinates.

0	0, 1	1, NT	2, 0	3, NT	
1	0, 2	1, NT	2, 1	3, NT	4, 1
2	0, NT	2, 2	4, NT		
3	0, NT	1, 3	3, NT		

(a)

0	0, 1	0, NT	0, 3	0, NT	0, NT
1	1, 0	3, 0	1, 2	2, 2	1, 3
2	2, NT		3, 1	3, NT	2, NT

(b)

Fig. 6. The coded regular orders after the third step (a) coded mapping of Y coordinates (b) coded mapping of X coordinates.

	0	1	2	3	4
0	1	1	0	0	0
1	2	2	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3

(a)

	0	1	2	3	4
0	1	0	3	2	3
1	0	0	2	2	3
2	0	0	1	2	3
3	0	0	1	2	3

(b)

Fig. 7. Look-up-tables after filling the NT pixels using the fourth step (a) mapping of Y coordinates (b) mapping of X coordinates.

0	0, 1	2, 0
1	0, 2	2, 1
2	0, 2	
3	0, 3	

(a)

0	1	2	3	4
0, 1	0, 0	0, 3	0, 2	0, 3
1, 0		1, 2		
		2, 1		

(b)

Fig. 8. Coded regular orders after filling the NT pixels using the fourth step (a) coded mapping of Y coordinates (b) coded mapping of X coordinates.

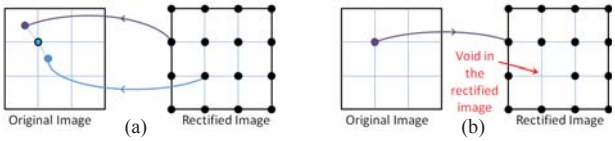


Fig. 9. Visualization of the reason for the voids on the rectified image (a) inverse mappings with fractional coordinates (b) forward mapping with integer coordinate.

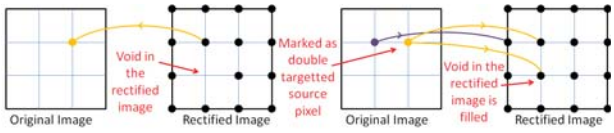


Fig. 10. Filling the voids on the rectified image in the fifth step (a) finding the source location of a pixel at one row above the void (b) marking the source pixel as double targeted pixel.

0	1, 0	2, -1	5, 0	← 3 brakpoints in a row
1	2, 0	2, -1	5, 0	
2	2, 0	5, 0		
3	3, 0	5, 0		← 2 brakpoints in a row

(a)

0	1	2	3	4
1, 0	0, 0	3, 0	2, 0	3, 0
1, -1	4, 0	1, -1	4, 0	4, 0
4, 0		2, -1		
		4, 0		

3 brakpoints in a column ↑ ↑ 2 brakpoints in a column

(b)

Fig. 11. Coding the behavior of breakpoints at the sixth step (a) coded mapping of Y coordinates (b) coded mapping of X coordinates.

The integer pixel precision forward mapping extracted for the example picture in Figure 4 yields the look-up-tables of X and Y coordinates shown in Figure 5. The pixels that are not targeted to any location are identified with NT. *Ori(2,2)* and *Ori(2,3)* are adjacent pixels, and both of them target "row no 2" of the rectified image; *Ori(2,2)* and *Ori(3,2)* are adjacent pixels and both target "column no 1" of the rectified image. This regular order is more apparent with higher resolution images. According to our experiments with a 1024×768 image, repetition of a single target coordinate up to 220 times is observed in the integer precision forward mapping table of X coordinates.

The method governing compressed rectification is similar to the run-length encoding technique. In the proposed coding scheme, instead of coding the run-length of the regular order, the locations where the regular order changes are encoded. These locations are called breakpoints. Moreover, the proposed scheme includes additional specific techniques to compress the integer precision forward mapping efficiently.

The regular order of the mapping of Y coordinates is encoded following a row-by-row scheme, and the regular order of mapping of X coordinates is encoded following a column-by-column scheme. The resulting look-up-tables after encoding Figure 5(a) and Figure 5(b) are presented in Figure 6(a) and Figure 6(b). In Figure 6(a), the elements of the compressed table are represented as *(column number, new value in row)*. In Figure 6(b), the elements of the compressed table are represented as *(row number, new value in column)*.

The high number of NT pixels dramatically increases the number of breakpoints. This issue becomes more pronounced for high resolution images. Therefore, the fourth step of the compression algorithm fills the NT pixel locations to keep the regular order. In order to fill the NT pixel locations, the same order is repeated vertically and horizontally for Y locations and X locations, respectively. After the fourth step, Figure 5 is transformed into Figure 7, and Figure 6 into Figure 8.

After the first two steps, two or more source fractional coordinates can have the same pixel coordinate in the original image as their nearest neighbor, as presented in Figure 9 (a). However, after step three and four, every integer pixel coordinate of the original image is targeted to a single coordinate in the rectified image. Consequently, some pixels in the rectified image may be void, as presented in Figure 9 (b). The fifth step is applied to fill these voids. As shown in Figure 10, the pixels on the original image which target the pixel coordinates that are located on the row above these voids are marked. Marked pixels are used to fill the voids as source pixels which have double targets.

The sixth step of the algorithm extracts the breakpoint locations and analyzes the behavior of the breakpoints. As shown in Figure 8, the difference between the new and previous target locations equals plus or minus one, which can be encoded consuming less data than encoding the exact integer coordinates. An example of coding the behavior of the cells in Figure 8 is presented in Figure 11 as *(location, behavior)*. The initialization coordinates are provided in the first column of the look-up-table for Y coordinates, and in the first row of the look-up-table for X coordinates. The next breakpoint values are identified with ± 1. Moreover, dummy breakpoints are inserted at the edges of the image to simplify the hardware implementation. Dummy insertions are represented by (5,0) and (4,0) in Figure 11 (a) and Figure 11 (b), respectively.

BreakPoint Conditions				Concatenation of 18-bits			
Initialization or Edge	Double Target	-1	+1	17^{th}	16^{th}	15^{th}	14^{th}-0^{th}
X	X	✓	X	0	1	0	Col No
X	X	X	✓	0	0	1	Col No
X	✓	X	X	1	0	0	Col No
X	✓	X	✓	1	0	1	Col No
X	✓	✓	X	1	1	0	Col No
✓	X	X	X	0	0	0	Col No
✓	✓	X	X	1	0	0	Col No

(a)

BreakPoint Conditions			Concatenation of 18-bits			
Initialization or Edge	-1	+1	17^{th}	16^{th}	15^{th}	14^{th}-0^{th}
X	✓	X	0	1	0	Row No
X	X	✓	0	0	1	Row No
✓	X	X	0	0	0	Row No

(b)

Fig. 12. Concatenation of the locations and behaviors at the seventh step (a) for the mapping of Y coordinates (b) for the mapping of X coordinates.

In the seventh step, the locations and behaviors of the breakpoints are concatenated and stored in a data array. Every BRAM in a Virtex-5 FPGA has 1024 addresses and it can be configured to store one array composed of 1024×36bits or two arrays composed of 1024×18bits. The BRAMs of the FPGAs are configured to store 18-bits in each address in the proposed concatenation scheme. As shown in Figure 12, 3-bits are used for coding the behaviors, and the remaining 15-bits encode the locations of the breakpoints. Therefore, the proposed concatenation scheme can easily be applied to an image that has a resolution lower than 32767×32767 pixels.

The number of the breakpoints in every row of the Y table and the number of the breakpoints in every column of the X table depend on the distortion of the lens, the resolution of the image sensor and the mechanical misalignment. The experimental setup used in this paper has 1024×768 resolution cameras. At most 21 breakpoints are observed in any given row of Y tables, and at most 17 breakpoints are observed in any given column of X tables. Data arrays are created for 24 possible breakpoint locations for Y tables and 20 possible breakpoint locations for X tables to support more challenging distortion conditions. Therefore, storing the X and Y tables for the right and left images requires 44 BRAMs which can even be supported by low cost FPGAs. The data arrays that are programmed into the BRAMs are converted into coefficient (COE) files using MATLAB.

In the eighth step, 44 BRAMs are instantiated as single port ROMs. The pre-computed compressed rectification maps are programmed into the BRAMs using the Xilinx ISE 12.4 and COE files.

IV. Real-Time De-Compression Hardware

The de-compression process is simpler than the off-line compression process in terms of computational complexity. The proposed rectification module can be used as a hardware accelerator taking place between the camera interface hardware and the on-chip memory controller, as shown in Figure 13. The rectification module is used for the left and right cameras separately. The rectification module processes source pixel values as $Ori_{(Yori,\ Xori)}$ and the respective source row and source column coordinates as Y_{ori} and X_{ori}. The rectification module computes the target row and target column coordinates as Y_{Rec} and X_{Rec}, and the 1-bit DT signal to identify double targeted locations. $Ori_{(Yori,\ Xori)}$ is delayed for 6 clock cycles and $Rec_{(Yrec,\ Xrec)}$ is given as an output. Due to the pipelined structure of the hardware, inputs can be consecutively received and outputs can be consecutively provided.

The top-level block diagram of the rectification module is presented in Figure 14. The rectification module involves 12 BRAMs to store the compressed table of Y coordinates and 10 BRAMs to store the compressed table of X coordinates. Half of 1 additional BRAM is used to store the last break point locations and the last target X coordinates of the row which is located above the row currently being processed.

The block diagram of the decompression hardware of Y coordinates is presented in Figure 15. The hardware resets itself every time X_{ori} is equal to zero which implies that the pixel of a new row is fetched from the camera. The target Y coordinate of the first incoming pixel in a new row is loaded from the ROM and written to the output register of Y_{Rec}. For every consecutive pixel, X_{ori} is compared to the coordinate of the next breakpoint which is loaded from the ROM. When a breakpoint is reached, the Y_{Rec} value is changed using a multiplexer depending on the coded behaviors of the breakpoints. Meanwhile, the hardware loads the coordinate of the next breakpoints to compare with the upcoming X_{ori}.

The block diagram of the decompression hardware of X coordinates is presented in Figure 16. Pixels are supplied by the camera row-by-row, whereas the X coordinates are compressed column-by-column. This situation causes one important difference between the de-compression hardware architectures of the X and Y tables. When the camera provides pixels of a new row, the de-compression hardware needs to keep record of the previous X_{Rec} coordinates and the last checked breakpoint address in the ROM for the respective column of the previous row. Two 1x1024 size data arrays are needed to store this information. These arrays are named *array_last_break_x* and *array_last_target_x* in Figure 14. These arrays are concatenated for respective column coordinates of the original image, and stored into one half of the 1 BRAM, which is named X_last_data_BRAM in Figure 16. The values in X_last_data_BRAM are replaced with the new ones when a breakpoint is reached for the respective X_{Ori}. The de-compression hardware of the Y coordinates does not comprise these arrays because Y coordinates are compressed row-by-row. Therefore, the last Y_{Rec} can be directly used for computing the next Y_{Rec} of the next pixel in the same row of the original image. The decompression hardware of X coordinates operates in a similar fashion as the decompression hardware of Y coordinates, with the exception of keeping record of the information about the previous row.

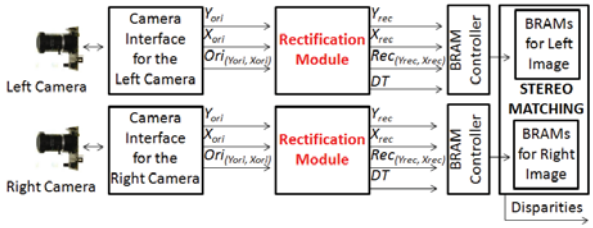


Fig. 13. Example utilization of the proposed rectification hardware.

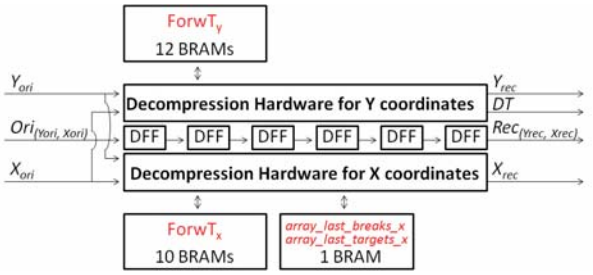


Fig. 14. Top-level block diagram of the proposed rectification hardware.

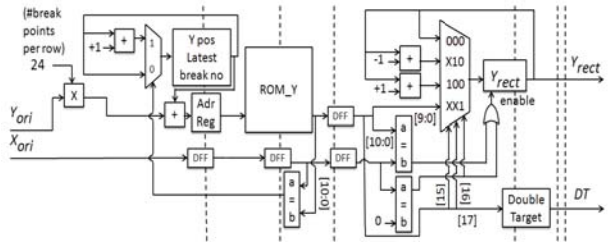


Fig. 15. Block diagram of the proposed rectification hardware for decompressing the table of Y coordinates. Pipeline stages are presented with dashed lines.

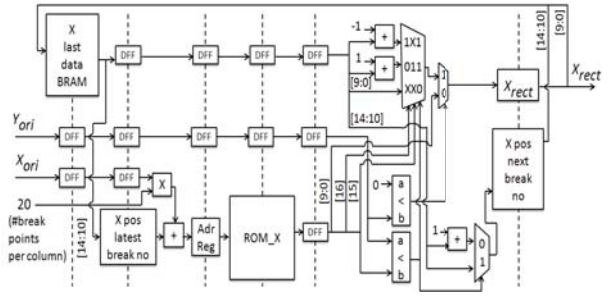


Fig. 16. Block diagram of the proposed rectification hardware for decompressing the table for X coordinates.

The proposed rectification hardware can be used in any stereo-matching system. The stereo matching process can be started when the required amount of rows is buffered in the BRAMs of the stereo matching hardware. Processed rows in these BRAMs can be overwritten by new rows during the stereo matching process.

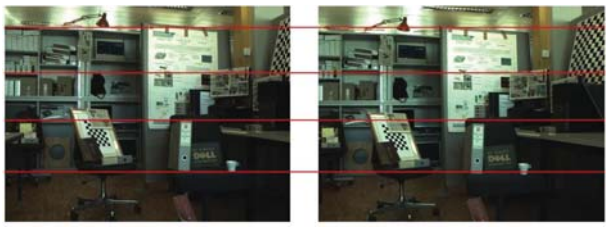
The hardware architectures presented in [9-10] require large pixel buffers due to the inverse mapping scheme. The proposed de-compression does not need large pixel buffers between the camera interface and the rectification modules. In contrast, the hardware requires these pixels buffers for the rectified image. However, typically DE hardware implementations already include BRAMs to buffer the pixels [2-4, 6]. Therefore, these buffers can be used for the proposed de-compression hardware. Thus, using the proposed rectification hardware on a complete DE system may not need additional large pixel buffers.

V. Implementation Results

The proposed rectification hardware is implemented using Verilog HDL, and verified using Modelsim 6.6c. The Verilog RTL models are mapped to a Virtex-5 XCUVP-110T FPGA comprising 69k Look-Up-Tables (LUT), 69k DFFs and 148 BRAMs. One rectification module consumes 0.32% of the LUTs, 0.28% of the DFF resources and 16% of the BRAM resources of the Virtex-5 FPGA. The proposed hardware operates at 273 MHz after place & route. Therefore, it can process up to 347 fps at a 1024×768 XGA video resolution. In addition, the proposed rectification hardware is merged with the DE hardware presented in [4]. The merged DE system is also verified using Modelsim 6.6c.

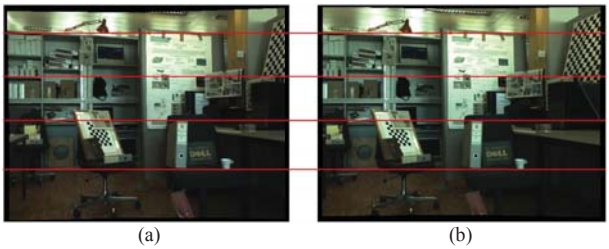
The proposed rectification hardware does not need the support of external memory if the cameras are synchronized. The cameras can be perfectly synchronized using one common I2C module for the initialization and driving the cameras with same clock source as explained in [11].

The proposed compression and decompression algorithms are evaluated using the pictures taken by the stereo camera system presented in [4]. The 1024×768 size original left and right pictures are shown in Figure 17. The original images are rectified using the Caltech rectification algorithm [1] and the proposed CLUT-R algorithm. The rectification results of the CLUT-R are presented in Figure 18. The breakpoint locations for the X and Y coordinates of the left image are presented in Figure 19.



(a) (b)

Fig. 17. Original images have distortions as observed near the lamp, bag, folder and cup; horizontal epipolar lines are demonstrated near the edge of these objects (a) left image (b) right image.



(a) (b)

Fig. 18. CLUT-R corrects distortions as observed near the lamp, bag, folder and cup (a) left image (b) right image.

(a) (b)

Fig. 19. Breakpoint locations of the left image (a) breakpoints of the targeted Y coordinates; coded row-by-row. (b) breakpoints of the targeted X coordinates; coded column-by-column.

(a) (b)

Fig. 20. DE results using the rectified images of CLUT-R (a) DE result of mini-census [2] (b) DE result of AWDE [4].

TABLE I. PSNR (dB) WITH THE RECTIFIED IMAGES PRODUCED BY [1]

	Comparison with Rectified Left Image [1]	Comparison with Rectified Right Image [1]
Original Image	15.69	16.08
Proposed (CLUT-R)	42.67	41.87

TABLE II. PSNR (dB) COMPARISON OF THE DISPARITY ESTIMATION RESULTS USING DIFFERENT DISPARITY ESTIMATION ALGORITHMS

	DE using [1] vs. DE using CLUT-R		DE using [1] vs. DE using Original Images	
	DR=120	DR=255	DR=120	DR=255
Mini-Census [2]	29.98	32.49	12.01	11.70
Georgulas [3]	28.72	32.31	12.39	13.44
AWDE [4]	29.95	32.87	12.93	13.65
Greisen [6]	26.30	27.94	11.20	10.79

TABLE III. HARDWARE RESOURCE COMPARISON OF THE RECTIFICATION HARDWARE IMPLEMENTATIONS

	Device	Resolution	LUT	DFF	Memory (KB)	E.M.
[7]	Virtex-4	752×480	3418	5932	0	✔
[8]	Virtex-E	640×512	2459	2075	99	✔
[9]	Spartan-2	640×480	≈2396	≈2396	16	✔
[10]	Virtex-5	1280×720	NA	NA	1300	X
CLUT-R	Virtex-5	1024×768	227	197	104	X
2×(CLUT-R + BRAM Contr.)	Virtex-5	1024×768	784	427	203	X

The PSNR between the rectification results of CLUT-R and Caltech rectification algorithm are evaluated in Table I. The PSNR of the left image is 42.67 dB, and the PSNR of the right image is 41.87 dB. Generally, a PSNR larger than 30 dB is considered acceptable to the human eye. Therefore, CLUT-R provides very high quality rectification results. The PSNR between the original images and the rectification results of Caltech are also provided in Table I for comparison.

The performance loss of CLUT-R is also evaluated for different DE algorithms. The DE algorithms that are implemented on real-time hardware are used for the evaluation [2-4, 6]. The DE results obtained using the images that are rectified by Caltech rectification algorithm are assumed as the respective ground truths of the DE algorithms. These ground truths are compared with the DE results of the respective algorithms using the images that are rectified by CLUT-R. The PSNR results are provided in Table II. 120 and 255 are applied as a disparity range (DR) and the respective DRs are used as peak signals for PSNR calculations. CLUT-R provides 32.87 dB and 27.94 dB PSNR for the AWDE algorithm [4] and Greisen et al. [6], for a 255 DR, respectively. Therefore, the proposed CLUT-R algorithm has an insignificant effect on the quality of the DE, and it can be used in different DE systems. The PSNR between the DE results using the original images and the DE results using the rectified images of Caltech are also provided in Table II for the comparison. In Figure 20, the DE results of the AWDE [4] and the mini-census [2] algorithms by using the rectified pictures of the CLUT-R are presented.

The hardware implementation of the CLUT-R is compared with the stereo image rectification hardware implementations in Table III. The hardware architecture of [7] requires a significant amount of hardware resources to support complex operations for solving the lens distortion models. Hardware architectures of look-up-table based implementations [8] and [9] require a significant amount of resources to implement external memory (E.M.) controller. Hence, combining CLUT-R with BRAM controller consumes less LUT and DFF resources than [7-9]. The DFF and LUT consumption of [10] is not available (NA). Nevertheless, the capacity of CLUT-R to fit the look-up-tables into the on-chip memory of the Virtex-5 FPGA is approximately six times more efficient than [10], as a benefit of its efficient compression scheme.

VI. CONCLUSION

In this paper, a novel compressed look-up-table based image rectification hardware is presented. The proposed method is based on off-line compression of the rectification information to be able to fit the tables into the on-chip memory of a Virtex-5 FPGA. The presented de-compression hardware consumes a negligible amount of hardware resources, and it does not require using any external memory to store the look-up-tables. The proposed hardware is advantageous if using external memory is considered as an additional cost, or if the disparity estimation system has external memory bandwidth limitations. The proposed rectification hardware would be even more profitable if it is adapted for high resolution multiple camera disparity estimation systems.

REFERENCES

[1] J. Y. Bouguet, "Camera Calibration Toolbox for Matlab," http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[2] N.C. Chang, T. H. Tsai, B. H. Hsu, Y. C. Chen, and T. S. Chang, "Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 20, no. 6, pp. 792-805, Jun. 2010.

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

[4] A. Akin, I. Baz, B. Atakan, I. Boybat, A. Schmid, and Y. Leblebici, "A Hardware-Oriented Dynamically Adaptive Disparity Estimation Algorithm and its Real-Time Hardware," *GLSVLSI Conference,* Paris, France, May 2013.

[5] D. Scharstein and R. Szeliski, "A Taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*,vol. 47, nos. 1–3, pp. 7–42, Apr. 2002.

[6] P. Greisen, S. Heinzle, M. Gross, and A. P. Burg, "An FPGA-based Processing Pipeline for High-Definition Stereo Video," *EURASIP Journal on Image and Video Processing*, pp. 18-25, Nov. 2011.

[7] H. Son, K. Bae, S. Ok, Y. Lee, and B. Moon, "A rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System," *Springer Journal on Communication and Networking*, pp. 147-155, 2012.

[8] C. Vancea, and S. Nedevschi, "LUT-based Image Rectification Module Implemented in FPGA," *IEEE International Conf. on ICCP*, Cluj, Romania, Sept. 2007.

[9] K. Gribbon, C. Johnston, and D. Bailey, "A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation," *Image and Vision Computing New Zealand*, pp. 408-413, 2003.

[10] D. H. Park, H. S. Ko, J. G. Kim, and J. D. Cho, "Real Time Rectification Using Differentially Encoded Lookup Table," *International ACM Conf. on ICUIMC*, 2011.

[11] A. Akin, O. Cogal, K. Seyid, H. Afshari, A. Schmid, and Y. Leblebici, "Hemispherical Multiple Camera System for High Resolution Omni-Directional Light Field Imaging," *IEEE Journal on JETCAS*, June 2013.