

Enhanced Compressed Look-up-Table Based Real-Time Rectification Hardware

Abdulkadir Akin, Luis Manuel Gaemperle, Halima Najibi, Alexandre Schmid, Yusuf Leblebici

▶ **To cite this version:**

Abdulkadir Akin, Luis Manuel Gaemperle, Halima Najibi, Alexandre Schmid, Yusuf Leblebici. Enhanced Compressed Look-up-Table Based Real-Time Rectification Hardware. 21th IFIP/IEEE International Conference on Very Large Scale Integration - System on a Chip (VLSI-SoC), Oct 2013, Istanbul, Turkey. pp.227-248, 10.1007/978-3-319-23799-2_11 . hal-01380308

HAL Id: hal-01380308

https://inria.hal.science/hal-01380308

Submitted on 12 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Enhanced Compressed Look-Up-Table Based Real-Time Rectification Hardware

Abdulkadir Akin, Luis Manuel Gaemperle, Halima Najibi, Alexandre Schmid, and Yusuf Leblebici

Ecole Polytechnique Fédérale de Lausanne (EPFL), School of Engineering (STI), Lausanne, Switzerland

{abdulkadir.akin, luis.gaemperle, halima.najibi, alexandre.schmid, yusuf.leblebici}@epfl.ch

Abstract. Real-time disparity estimation requires real-time rectification which involves solving the models of lens distortions, image translations and rotations. Low complexity look-up-table based rectification algorithms usually require an external memory to store large look-up-tables. In this chapter, we present an implementation of the look-up-table based approach which compresses the rectification information to fit the look-up-table into the on-chip memory of a Virtex-5 FPGA. First, a very low complexity compressed look-up-table based rectification algorithm (CLUTR) and its real-time hardware are presented. The implemented CLUTR hardware rectifies stereo images with moderate lens distortion and camera misalignment. Moreover, an enhanced version of the compressed look-up-table based rectification algorithm (E-CLUTR) and its novel real-time hardware are presented. E-CLUTR solves more extreme camera alignment and distortion issues than CLUTR while maintaining the low complexity architecture.

Keywords: Stereo Matching, Image Rectification, Compression, Real-Time, Hardware Implementation, FPGA

1 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 the lenses were perfect, without any distortion, the matching pixels would be located in the same row of the right and left images. However, providing a perfect set-up is virtually 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 solution that consumes a low amount of hardware resources in an FPGA or ASIC [8-10]. In [8-10], the mappings between original image pixel coordinates and rectified image pixel coordinates are precomputed and then 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 external storage for the image rectification process may increase the cost of the disparity estimation hardware system or impose additional external memory bandwidth limitations on the system. In [10], the 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 the largest Virtex-5 FPGAs or other recent high-end FPGAs.

In this chapter, we present a novel compressed look-up-table based image rectification (CLUTR) algorithm and its real-time hardware. A preliminary description of CLUTR has been presented in [11]. In this chapter, the CLUTR algorithm and its real-time hardware implementation are explained with further details. In addition, an enhanced version of the CLUTR (E-CLUTR) algorithm and its real-time hardware implementation are presented. The real-time hardware of E-CLUTR maintains the low complexity architecture of CLUTR, while it is able to rectify images under excessive mechanical misalignment of the cameras and lens distortions. Moreover, the Caltech rectification algorithm [1] which does not benefit from look-up-tables is implemented in hardware, and its hardware resource consumption results are presented to improve the hardware comparison and to evidence the efficiency of CLUTR and E-CLUTR much fairly.

This chapter is structured as follows. A typical look-up-table based rectification process is introduced in Section 2. The compression scheme that is used by the CLUTR algorithm is presented in Section 3. The proposed hardware implementation of the decompression process of CLUTR is presented in Section 4. The analysis of the challenges of CLUTR to rectify images under extreme conditions is presented in Section 5. The compression scheme of E-CLUTR is presented in Section 6. The hardware implementation of the decompression process of E-CLUTR is presented in Section 7. The implementation results of the CLUTR and E-CLUTR algorithms and their hardware resource comparisons are presented in Section 8. Section 9 concludes the chapter.

2 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 based on the given pixel locations in the original image. Inverse mapping computes the original source pixel locations based on 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 needed. 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.

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

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

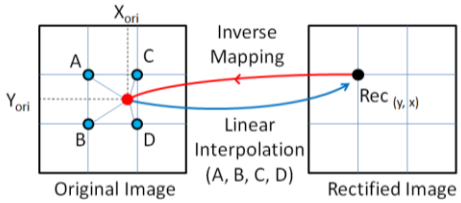


Fig. 1. Inverse mapping with fractional precision coordinates. Corners indicate integer pixel coordinates.

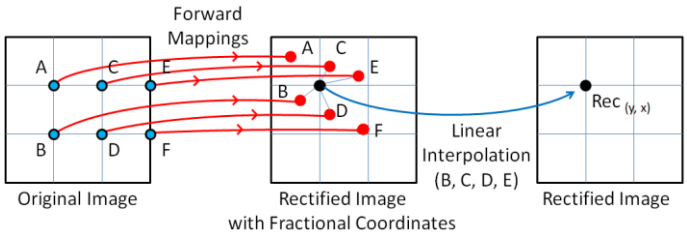


Fig. 2. Forward mapping with fractional precision coordinates

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 computations 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 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, the rectification map alone 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].

3 Proposed Compression Scheme of the CLUTR Algorithm

In contrast to the selection of the hardware implementations of [8-10], a forward mapping based rectification scheme is selected for the proposed CLUTR algorithm. In CLUTR, fractional precision is ignored. Ignoring fractional precision allows an efficient compression scheme. The negligible distortion in the rectified images originating from this simplification is analyzed in Section 8.

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.

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 Y coordinate mapping is encoded following a row-by-row scheme, and the regular order of the X coordinate mapping 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 and X locations, respectively. After the fourth step, Figure 5 is transformed into Figure 7, and Figure 6 into Figure 8.

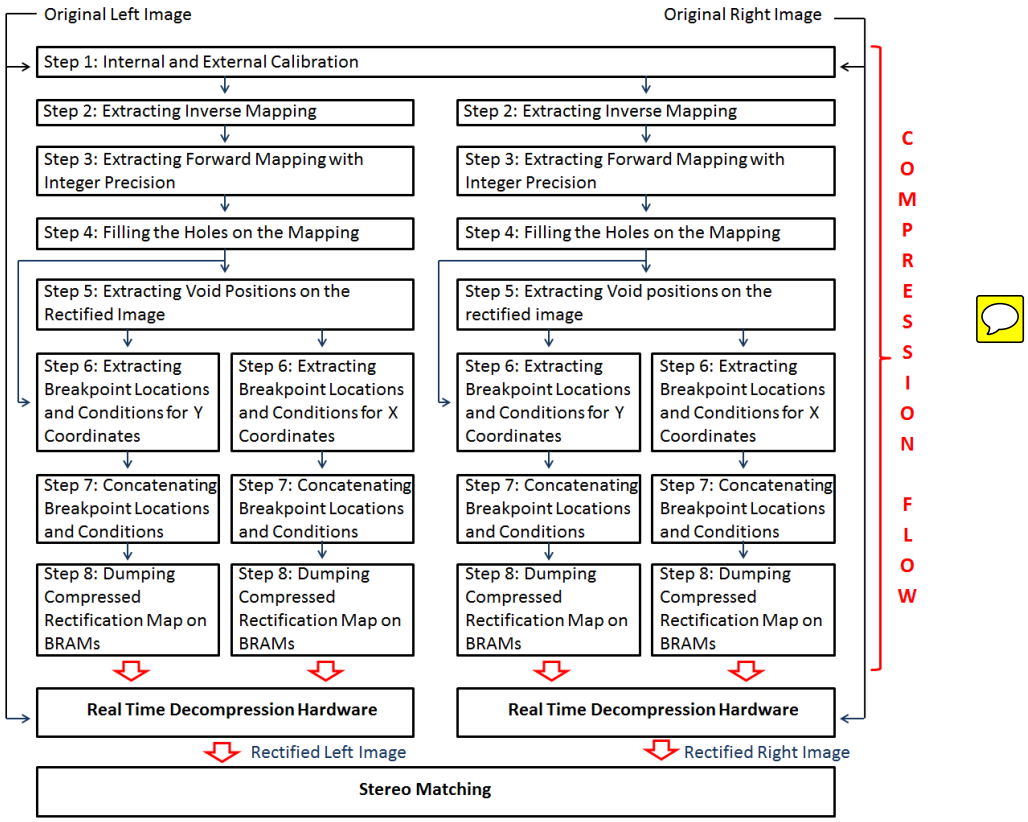
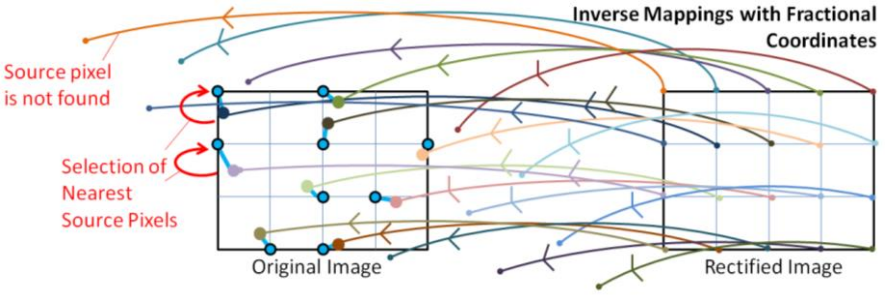
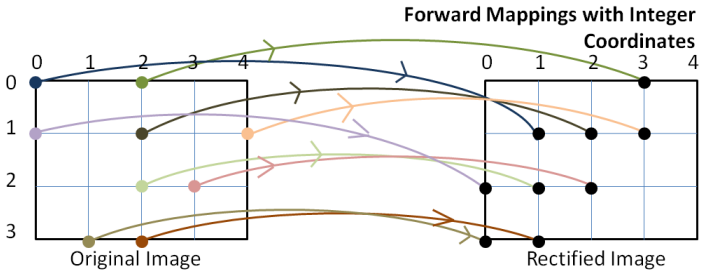


Fig. 3. Flow chart for the proposed compressed look-up-table based stereo image rectification process



(a)



(b)

Fig. 4. 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	1	2	3	4
0, 1	0, NT	0, 3	0, NT	0, NT
1, 0	3, 0	1, 2	2, 2	1, 3
2, NT		2, 1	3, NT	2, NT

(b)

Fig. 6. 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

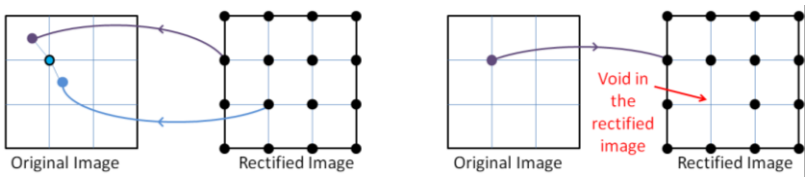
0	0, 1	2, 0
1	0, 2	2, 1
2	0, 2	
3	0, 3	

(a)

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

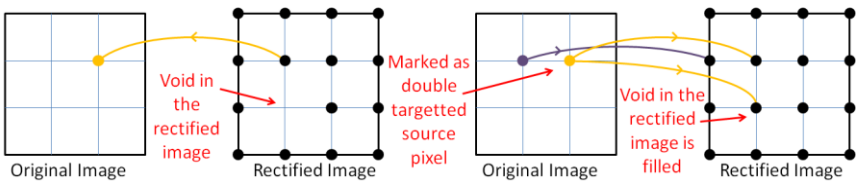
(b)

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



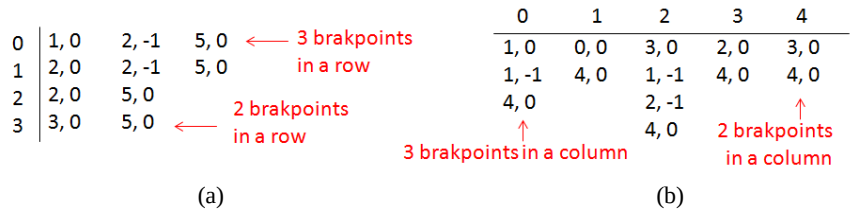
(a) (b)

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



(a) (b)

Fig. 10. Filling the voids in 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



(a) (b)

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

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). These pixels will remain as void, i.e. black pixels, in the rectified image if they are not filled. 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 (DT). Thus, DT pixels are used to concurrently target two vertically neighboring pixels on the rectified image.

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.

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 be applied to an image that has a resolution lower than 32767×32767 pixels. In Fig. 12 (a), DT and changing the last targeted row by ±1 are independent breakpoint conditions of Y coordinates, which can be applied to source pixels, concurrently or separately. Therefore, the "X" symbol in the -1 and +1 columns of Fig. 12 implies keeping the last targeted row coordinate.

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

The number of breakpoints in every row of the Y table and the number of 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 chapter consists of 1024×768 resolution cameras. In the experiments, cameras are aligned in parallel configuration without using any sensitive mechanical placement tool. 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 of CLUTR 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 38 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, 38 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.

4 Real-Time Decompression Hardware of CLUTR

The decompression 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 (768×24×18)/(1024×36) = 9 BRAMs to store the compressed table of Y coordinates and (1024×20×18)/(1024×36) = 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 first 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 decompression hardware architectures of the X and Y tables. When the camera provides pixels of a new row, the decompression 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 1×1024 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 decompression 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.

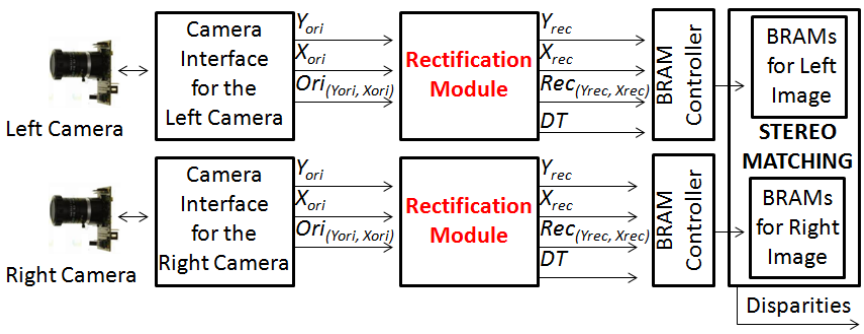


Fig. 13. Example utilization of the proposed rectification hardware

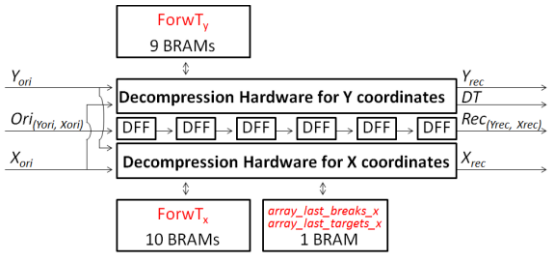


Fig. 14. Top-level block diagram of the proposed rectification hardware of CLUTR

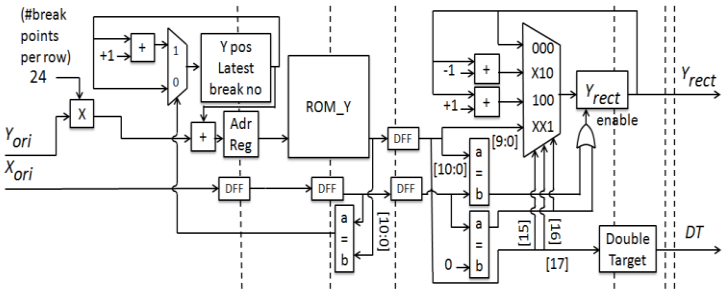


Fig. 15. Block diagram of the proposed rectification hardware for decompressing the table of Y coordinates. Pipeline stages are presented with dashed lines.

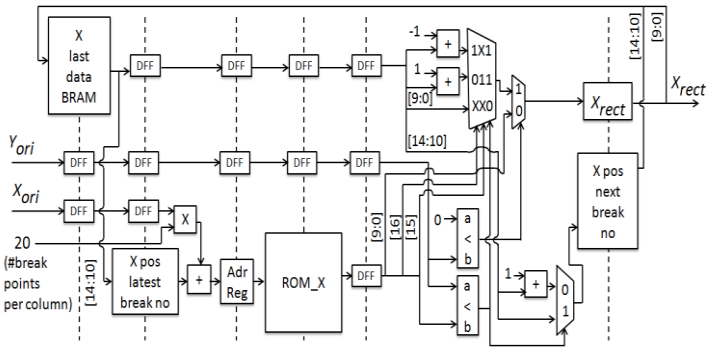


Fig. 16. Block diagram of the proposed rectification hardware for decompressing the table for X coordinates

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 decompression 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 decompression hardware. Thus, using the proposed rectification hardware on a complete DE system may not need additional large pixel buffers.

5 Limitations of the CLUTR

In an ideal case, i.e. where the cameras are perfectly parallel and lenses do not have distortion, two breakpoints are required for every row of the look-up-table for Y coordinates and two breakpoints are required for every column of the look-up-table for X coordinates. One of these two breakpoints is needed to define the initial breakpoint location to target coordinate 0, and the other one is needed to define the final target location as the horizontal or vertical size of the image. In the classical case of a real-time working environment, the mechanical set-up of the stereo-matching system should be carefully designed to be close to an ideal case. Still the main goal of the rectification consists of solving lens distortions and sensitive mechanical misalignments.

According to tests applied to the CLUTR algorithm and hardware, the pixel location difference of two consecutive breakpoints typically reaches more than 15 pixels and the number of breakpoints is smaller than the pre-defined breakpoint capacity of the ROMs of CLUTR. Therefore, CLUTR successfully rectifies the images when the lens distortion and the mechanical misalignments are not excessive. However, unusual conditions bring limitations on the CLUTR hardware.

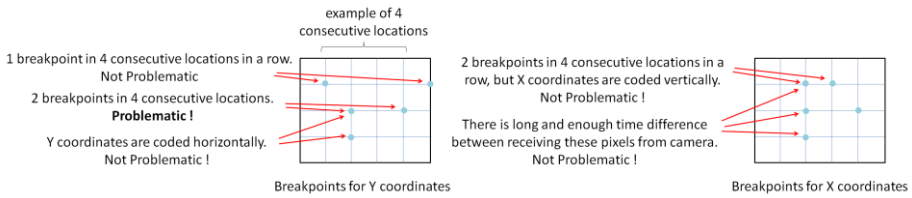
Two important limitations of the CLUTR hardware must be considered to maintain its suitability to rectify challenging situations. The first limitation relates to the capacity of the ROMs to store a sufficient number of breakpoints. The second limitation relates to the frequency of the breakpoints.

The limitation caused by the number of breakpoints is mainly due to the mechanical misalignment of the cameras. In order to identify the limit of the breakpoint storage capacity of ROMs, two cameras are manually rotated around 3 degrees around opposite directions of all rotational axis. This test can be considered as an excessive misalignment of a carefully designed mechanical setup of the stereo-matching system. Using the compression scheme of CLUTR, 43 breakpoints are needed in the look-up-table of X coordinates for one column, and 69 breakpoints are needed in the look-up-table of Y coordinates for one row. The pre-defined breakpoint capacity of CLUTR does not support this condition. Overcoming this first limitation is straightforward to achieve by increasing the size of ROMs to store more breakpoints.

CLUTR supports rectification if the two breakpoints of the Y coordinates have at least 4 pixel position difference. When a breakpoint location is reached, the hardware needs to read the next breakpoint from ROM. The address computation and reading the next breakpoint from the ROM consume 4 clock cycles. The camera continues to send pixels and the camera controller increases X_{Ori} during the address computation and reading breakpoints from the ROM. Therefore, if there are multiple breakpoints in 4 consecutive pixels, CLUTR is not able to apply a breakpoint condition to those pixels. Hence, the limits of the CLUTR hardware to successfully rectify stereo images is exceeded if breakpoints are frequent, i.e. if two breakpoints of Y coordinates in a row have less than 4 pixel position difference. Since the breakpoints of X coordinates are coded column by

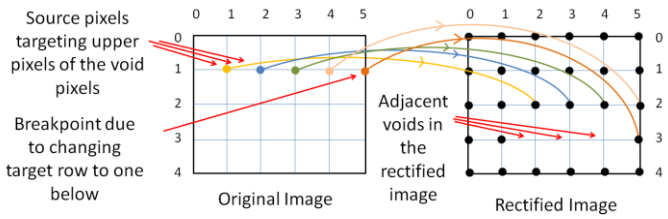
column but the camera sends pixels row by row, the time to process consecutive breakpoints in same column and consecutive rows is sufficiently long. Therefore, frequent breakpoints in the same column of the look-up-table of X coordinates does not cause a limitation. The frequency limitation of CLUTR is visually explained in Fig. 17.

The main reason for the occurrence of frequent breakpoints is the high number of adjacent void pixels, which is caused by excessive camera misalignment or lens distortion. An example of one such challenging condition is presented in Fig. 18. As shown in Fig 18 (b), 4 out of 5 consecutive pixels are marked as breakpoints. 3 out of these 4 breakpoints are DT breakpoints which are coded in the look-up-table of Y coordinates, and the other one is located at *Ori(1,5)* which requires changing the target row number from 2 to 3. This challenging example case exceeds the limits of CLUTR, since CLUTR is not able to apply a breakpoint condition if there are multiple breakpoints in 4 consecutive pixels.

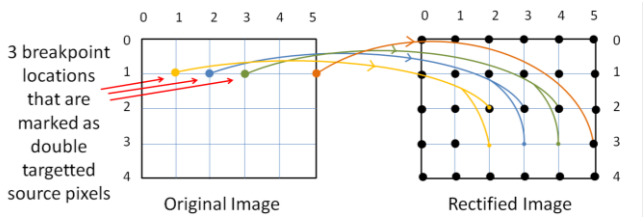


(a) (b)

Fig. 17. Visualization of the breakpoint frequency capacity of the X and Y coordinate mappings (a) breakpoints for the mapping of Y coordinates (b) breakpoints for the mapping of X coordinates



(a)



(b)

Fig. 18. Visualization of the reason for the frequent breakpoints (a) finding the source locations of three pixels that are targeting one row above of the three consecutive voids (b) four breakpoints in consecutive five locations

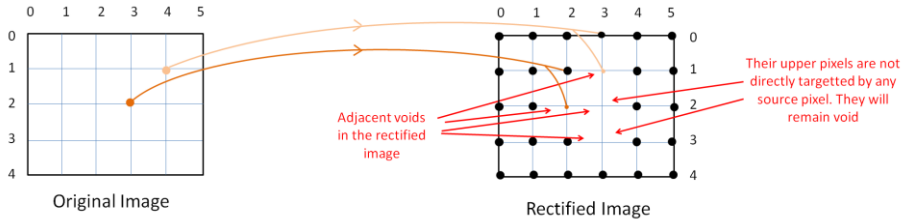


Fig. 19. Visualization of the reason for the voids which can not be filled by CLUTR

Adjacent void pixels may occur not only horizontally but also vertically. Vertically adjacent void pixels may cause void pixels, which cannot be filled by the DT feature of CLUTR. As visualized in Fig. 19, DT pixels can fill the voids located one row below the targeted pixel on the rectified image. If there are two voids which are vertically adjacent, the void below can not be filled by CLUTR since the pixel above is not targeted directly by any source pixel.

Another limitation of CLUTR is related to the usage of ROMs for the hardware implementation which increases the off-line processing duration. Using ROMs is suitable to demonstrate the efficiency of compressed look-up-table based rectification. However, after each adjustment of the camera settings and alignments, creating new compressed tables of the hardware requires re-synthesis and place & route of the implementation. Thus it takes a long time to initialize CLUTR hardware.

6 The compression scheme of the E-CLUTR algorithm

The E-CLUTR algorithm and its hardware implementation are designed to overcome the limitations of CLUTR while maintaining the low complexity decompression scheme. The limitations of CLUTR are mainly solved by improving the design of the decompression hardware. Moreover, algorithmic enhancements are applied to further improve the efficiency of the compression scheme to handle challenging lens distortions and mechanical misalignments.

Algorithmic enhancements are explained in this section. The flow chart of the compression scheme of E-CLUTR is identical to the flow chart presented in Fig 3. The steps 5, 6 and 7 that are shown in Fig. 3 are enhanced in E-CLUTR. The algorithmic enhancement for the compression scheme is proposed to reduce the frequency of DT breakpoints. In order to decrease the amount of consecutive breakpoints in a row, the condition type of breakpoints for filling the voids are improved. As explained in Section 4 in step 5, DT breakpoints are used to fill the voids that are located one pixel below the targeted pixel. To avoid any confusion, the DT condition of CLUTR is renamed as below-DT (B-DT) in E-CLUTR. In addition to B-DT, below-backward-DT (BB-DT), below-forward-DT (BF-DT), upper-DT (U-DT), upper-backward-DT (UB-DT) and upper-forward-DT (UF-DT) breakpoint conditions are defined in E-CLUTR. Using extra DT conditions, the source pixel can be targeted not only to one pixel below the target, but additional options are provided to fill any of 6 possible neighbors of the targeted pixel of the rectified image. These additional options are visualized in Fig. 20.

As presented in Fig. 21, the frequency of breakpoints in the same row is reduced compared to Fig. 18, by using BF-DT, BB-DT and U-DT breakpoint conditions. As presented in Fig. 19, vertically adjacent void pixels are problematic for CLUTR. However, these voids can be filled by multiple DT options of E-CLUTR as presented in Fig. 22.

In step 6, the breakpoints are coded considering the new DT breakpoint conditions. In the challenging example, there should be support of 2 consecutive breakpoints at least in 3 consecutive pixel coordinates, as presented for the breakpoints at *Ori(1,4)* and *Ori(1,5)*. Therefore, the algorithmic enhancement requires the support of at least 2 breakpoints for 3 consecutive locations as an additional constraint. The hardware based enhancement to provide this support is explained in section 7.

At the edges of the rectified image, black pixels occur. These stem from the 3-D rotation of the original image and the mapping of the rectified image to the original resolution. Consequently, the effective resolution slightly decreases in the rectified image [1]. Due to this fact, changing row and column coordinates stay in the range of ±1. However, this range may not be guaranteed for all possible extreme conditions. Thus, a generic solution should cover all possible extreme situations. In order to cover the cases that create a situation beyond the challenging camera misalignment tests, ±2 row and ±2 column coordinate change options are included as a breakpoint condition in step 6 of E-CLUTR.

The concatenation scheme presented in Fig. 12 of CLUTR is modified for E-CLUTR as presented in Fig. 23, Fig. 24 and Fig. 25 using improved breakpoint conditions. The breakpoint conditions for DT conditions and row changing can be applied concurrently or separately to the source pixel. Consequently, 7×5=35 different conditions occur for the concatenation of the conditions for the breakpoints of Y coordinates. The brief concatenation scheme of the Y coordinates is presented in Fig. 23. The concatenation scheme for DT codes and row changing are separately presented in Fig. 24 (a) and Fig. 24 (b). The concatenation scheme of the X coordinates is presented in Fig. 25.

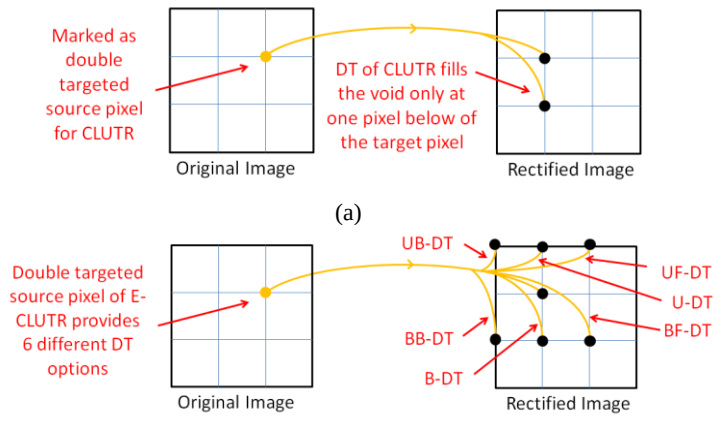
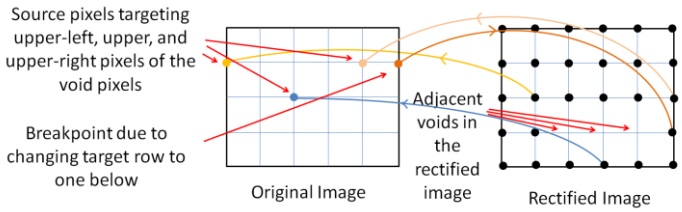
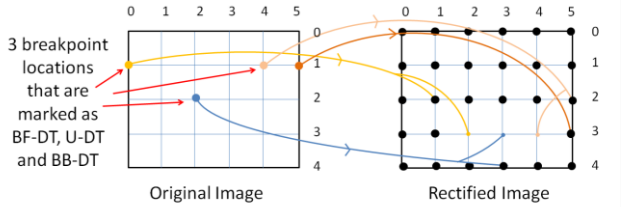


Fig. 20. Filling the voids on the rectified image in the fifth step (a) DT option of CLUTR (b) DT options of E-CLUTR



(a)



(b)

Fig. 21. Reducing the frequency of breakpoints using multiple DT options of E-CLUTR (a) finding alternative source locations for void pixels of rectified image (b) reduced frequency of breakpoints for the same row of the look-up-table of Y coordinates

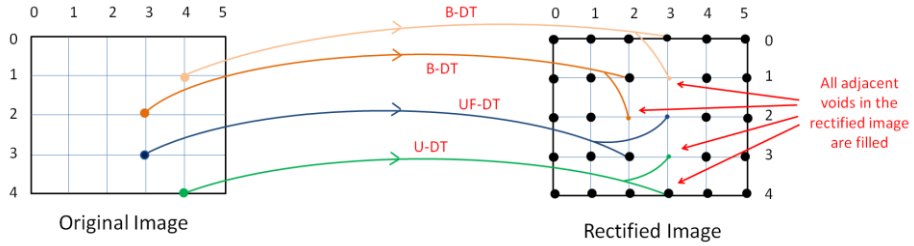


Fig. 22. Vertically adjacent void pixels can be filled by E-CLUTR using multiple DT options.

BreakPoint Conditions					Concatenation of 18-bits						
Initialization or Edge	Double Target		Changing Row		17^{th}	16^{th}	15^{th}	14^{th}	13^{th}	12^{th}	11^{th}-0^{th}
	Up/Below	Forward/Backward	-/+	1/2							
					DT Conditions			Conditions for Changing Last Targeted Row			Col No

Fig. 23. Brief representation for the concatenation of the locations and behaviors at the seventh step of E-CLUTR for the mapping of Y coordinates

BreakPoint Conditions		Concatenation of 18-bits		
Double Target				
Up/ Below	Forward/ Backward	17th	16th	15th
U	F	0	0	1
U	Non	0	1	0
U	B	0	1	1
B	F	1	0	0
B	Non	1	0	1
B	B	1	1	0
Non	Non	0	0	0

(a)

BreakPoint Conditions		Concatenation of 18-bits		
Changing Row				
-/+	1/2	14th	13th	12th
+	1	1	0	0
+	2	1	0	1
-	1	1	1	0
-	2	1	1	1
Non	Non	0	0	0

(b)

Fig. 24. Concatenation of the locations and behaviors at the seventh step for E-CLUTR for the mapping of Y coordinates (a) Concatenation scheme for DT options (b) Concatenation scheme for the breakpoint conditions for changing last targeted row

BreakPoint Conditions			Concatenation of 18-bits		
Initialization or Edge	Changing Column		17th	16th	15th-0th
	+/-	1/2			
X	+	1	0	0	Row No
X	+	2	0	1	Row No
X	-	1	1	0	Row No
X	-	2	1	1	Row No
✓	X	X	0	0	Row No

Fig. 25. Concatenation of the locations and behaviors at the seventh step of E-CLUTR for the mapping of X coordinates

7 Real-Time Decompression Hardware of E-CLUTR

The top-level block diagram of the E-CLUTR module is presented in Figure 26. Data arrays of E-CLUTR are created for 80 possible breakpoint locations for Y tables and 50 possible breakpoint locations for X tables to support very challenging distortion conditions. The rectification module involves (768×80×18)/(1024×36) = 30 BRAMs to store the compressed table of Y coordinates and (1024×50×18)/(1024×36) = 25 BRAMs to store the compressed table of X coordinates. As presented in Figure 26, the ROMs are converted to RAM to ease the initialization of the look-up-tables of E-CLUTR after changing the camera settings. The decompression hardware for the X coordinates is presented in Figure 27. The decompression hardware for Y coordinates is presented in Figure 28.

As presented in Figure 27, the decompression hardware of E-CLUTR pertaining to X coordinates is similar to the hardware used in CLUTR. The multiplexing stage is adapted to provide the ±2 target pixel column change feature for the X breakpoints.

The decompression hardware of E-CLUTR pertaining to Y coordinates is redesigned to support frequent breakpoints and the six different DT options. As presented in Figure 28, the E-CLUTR hardware reads the first six breakpoints from the RAM as soon as the camera starts to send a new row. The first breakpoint is used to initialize the target

row and the next five breakpoints are buffered in a local cache. Whenever a new breakpoint location is reached, the next breakpoint location is read from the RAM and the cache shifts the existing upcoming breakpoint locations. Using this local cache of the breakpoints, the original pixel coordinates can be compared to the pixel locations in the cache. Therefore, the breakpoint conditions can be applied to all passing pixels even if the breakpoints are frequent.

The multiplexing stage for the computation of the next target row is improved to provide the ±2 target pixel row change feature. Moreover, the hardware sends the 3-bit DT condition to the BRAM controller together with the pipelined source pixel and its target locations $Y_{Rec,}$and X_{Rec}, synchronously.

The BRAM controller that is shown in Fig. 13 writes the pipelined source pixels to the decompressed target row and target column coordinates. E-CLUTR hardware is verified by merging it with the DE hardware presented in [4], which buffers pixels of rows in its own, separate BRAMs. Y_{Rec} is used to select and enable the BRAM to write target pixel. X_{Rec} is used to determine the write address of the enabled BRAM of the DE hardware. If a DT condition exists, the same source pixel is written to two BRAMs concurrently by enabling two BRAMs that buffer two consecutive rows. If the DT condition is pointing into forward or backward positions, the address port of the BRAM that is targeted by the DT condition receives the computed target BRAM address ±1.

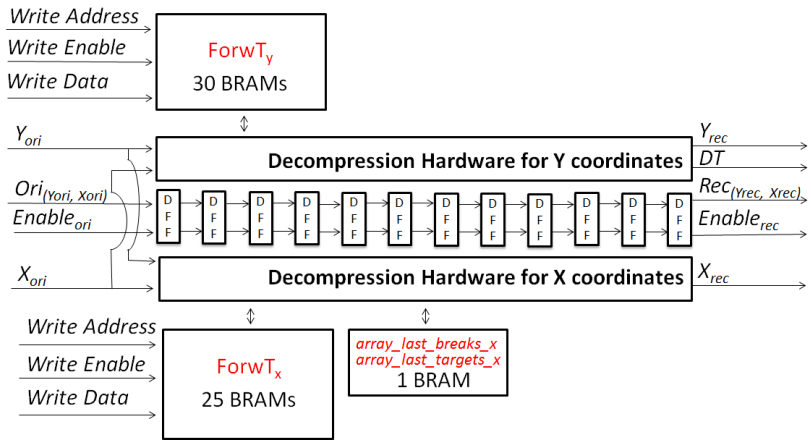


Fig. 26. Top-level block diagram of the proposed rectification hardware of E-CLUTR

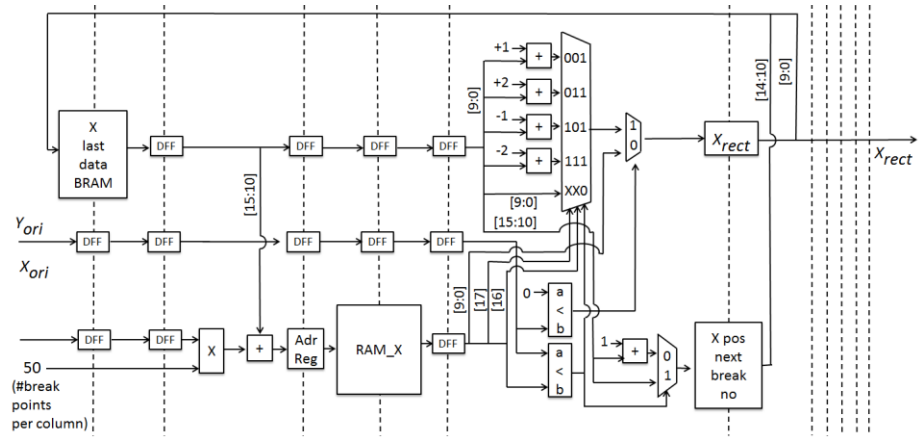


Fig. 27. Block diagram of the proposed rectification hardware for decompressing the table of X coordinates

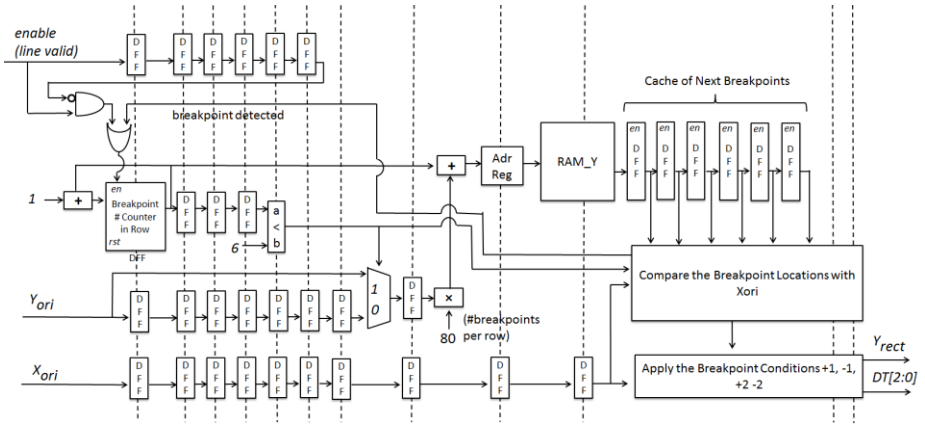


Fig. 28. Block diagram of the proposed rectification hardware for decompressing the table of Y coordinates. Pipeline stages are presented with dashed lines.

8 Implementation Results

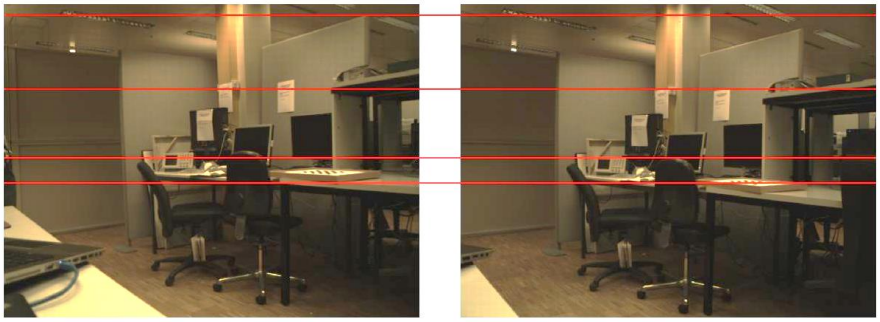
The proposed rectification hardware architectures of CLUTR and E-CLUTR are implemented using Verilog HDL, and verified using Modelsim 10.1d. The Verilog RTL models are mapped to a Virtex-5 XCUVP-110T FPGA comprising 69k Look-Up-Tables (LUT), 69k DFFs and 148 BRAMs. One rectification module of CLUTR consumes 0.32% of the LUTs, 0.28% of the DFF resources and 14% of the BRAM resources of the Virtex-5 FPGA. One rectification module of E-CLUTR consumes 0.63% of the LUTs, 0.51% of the DFF resources and 38% of the BRAM resources of the Virtex-5 FPGA. The proposed E-CLUTR hardware operates at 212 MHz after place & route.

Therefore, it can process up to 269 fps at a 1024×768 XGA video resolution. In addition, the proposed rectification hardware of CLUTR and E-CLUTR are merged with the DE hardware presented in [4]. The merged DE systems are also verified using Modelsim 10.1d.

The proposed rectification hardware of CLUTR and E-CLUTR do not need the support of external memory if the cameras are synchronized. The cameras can be perfectly synchronized by driving the cameras with same clock source and using one common I2C module for the initialization the cameras [12].

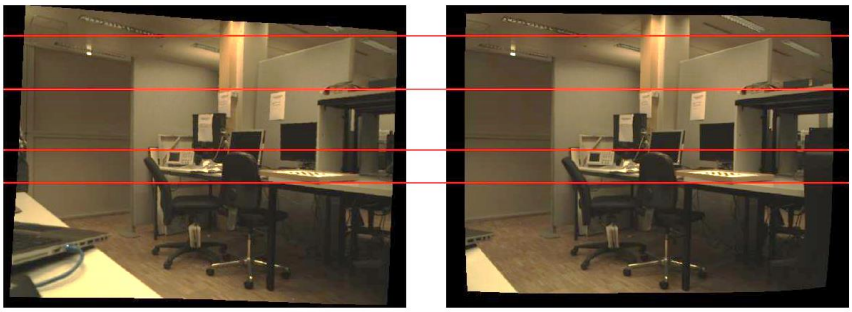
The proposed compression and decompression algorithms are evaluated using the pictures taken by the stereo camera system presented in [4]. If the cameras are not extremely misaligned, CLUTR and E-CLUTR provide identical visual and numerical results. The 1024×768 size original left and right pictures that are shown in Figure 29 are taken under a camera misalignment condition that exceed the limits of CLUTR implementation. The example pictures and the test results of CLUTR for typical camera misalignment are presented in [11]. The original images in Figure 29 are rectified using the Caltech rectification algorithm [1] and the proposed E-CLUTR algorithm. The rectification results of the E-CLUTR is presented in Figure 30. The extreme rotation of the rectified image can be visually observed in Figure 30. The breakpoint locations for the X and Y coordinates of the left image are presented in Figure 31.

The PSNR between the rectification results of E-CLUTR and Caltech rectification algorithm are evaluated in Table I. The PSNR of the left image is 43.10 dB, and the PSNR of the right image is 42.02 dB. Generally, a PSNR larger than 30 dB is considered acceptable to the human eye. Therefore, E-CLUTR 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.



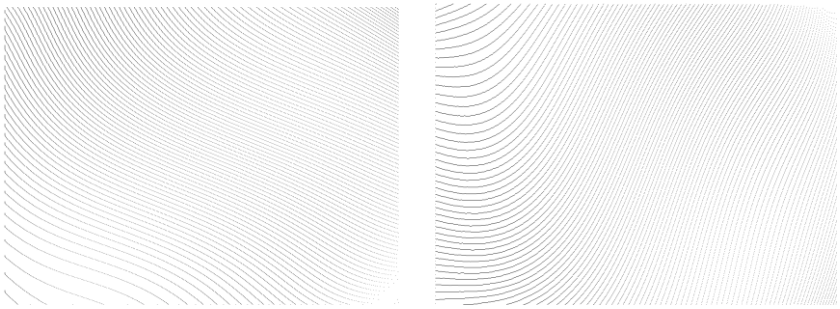
(a) (b)

Fig. 29. Original images have distortions as observed on the lines (a) left image (b) right image.



(a) (b)

Fig. 30. E-CLUTR corrects distortions as observed on the lines (a) left image (b) right image.



(a) (b)

Fig. 31. 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

Table 1. PSNR (dB) with the Rectified Images Produced By [1]

	Comparison with Rectified Left Image [1]	**Comparison with Rectified Right Image [1]**
Original Image	17.99	19.34
Proposed (E-CLUTR)	43.10	42.02

The hardware implementation of the E-CLUTR is compared with the stereo image rectification hardware implementations in Table II. 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 an external memory. Combining the E-CLUTR with BRAM controller consumes less LUT and DFF resources than [7-9] and it does not require an external memory. The DFF and LUT consumption of [10] is not available (NA). Nevertheless, the capacity of E-CLUTR to fit the look-up-tables into the on-chip memory of the Virtex-5 FPGA is approximately two times more efficient than [10], as a benefit of its efficient compression scheme. Moreover, high precision hardware of the Caltech rectification algorithm is implemented, and its hardware consumption results are presented in Table II for comparison. Hardware implementations of CLUTR and E-

CLUTR require much less hardware resource than the hardware implementation of Caltech rectification while providing almost same rectification results.

The hardware resource consumption of E-CLUTR is higher than CLUTR hardware. However, if the cameras are extremely misaligned, the limitations of CLUTR can be exceeded. In these extreme conditions, E-CLUTR still supports rectification. Whereas, using CLUTR hardware can be more profitable if the stereo cameras are carefully aligned.

Table 2. Hardware Resource Comparison of the Rectification Hardware Implementations

	Device	**Resolution**	**LUT**	**DFF**	**On-Chip Memory (KB)**	**External Memory**
[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
Caltech Hardware	Virtex-5	1024×768	24384	25346	0	✔
CLUTR	Virtex-5	1024×768	227	197	90	X
2×(CLUTR + BRAM Contr.)	Virtex-5	1024×768	784	427	176	X
E-CLUTR	Virtex-5	1024×768	434	350	252	X
2×(E-CLUTR + BRAM Contr.)	Virtex-5	1024×768	2278	956	500	X

9 Conclusion

In this chapter, two novel compressed look-up-table based image rectification algorithms and their hardware implementations are presented. The proposed CLUTR and E-CLUTR algorithms are based on off-line compression of the rectification information to fit the tables into the on-chip memory of a Virtex-5 FPGA. The presented decompression hardware implementations of CLUTR and E-CLUTR consume negligible amounts of hardware resources, and they do not require any external memory to store the look-up-tables. The proposed hardware implementations are 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 implementations would be even more profitable if they are adapted for high resolution multiple camera disparity estimation systems.

References

1. Bouguet, J. Y.: Camera Calibration Toolbox for Matlab, http://www.vision.caltech.edu/bouguetj/calib_doc/index.html (2010)
2. Chang, N.Y.C., Tsai, T.H., Hsu, B.H., Chen, Y.C., Chang, T.S.: Algorithm and Architecture of Disparity Estimation with Mini-Census Adaptive Support Weight. IEEE Transact. Circuits Syst. Video Technol. 20 (6), pp. 792–805 (2010)

3. Georgoulas, C., Andreadis, I.: A Real-time Occlusion Aware Hardware Structure for Disparity Map Computation. In: Image Analysis and Processing ICIAP 2009, vol. 5716, pp. 721–730. Springer, Berlin (2009)
4. Akin, A., Baz, I., Atakan, B., Boybat, I., Schmid, A., Leblebici, Y.: A Hardware-Oriented Dynamically Adaptive Disparity Estimation Algorithm and Its Real-Time Hardware. In: Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI, pp. 155-160. ACM, Paris (2013)
5. Scharstein, D., Szeliski, R.: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. Int. J. Comput. Vision, vol. 47(1–3), pp. 7–42 (2002)
6. Greisen, P., Heinzle, S., Gross, M., Burg, A. P.: An FPGA-based Processing Pipeline for High-Definition Stereo Video. EURASIP Journal on Image and Video Processing, vol. 1, pp. 18-25 (2011)
7. Son, H., Bae, K., Ok, S., Lee, Y., Moon, B.: A Rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System. Springer Journal on Communication and Networking, pp. 147-155 (2012)
8. Vancea, C., Nedevschi, S.: LUT-based Image Rectification Module Implemented in FPGA. In: IEEE International Conference on Intelligent Computer Communication and Processing, pp. 147–154 (2007)
9. Gribbon, K., Johnston, C., Bailey, D.: A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation. In: Proc. Image and Vision Computing, pp. 408-413. New Zealand (2003)
10. Park, D. H., Ko, H. S., Kim, J. G., Cho, J. D.: Real Time Rectification Using Differentially Encoded Lookup Table. In: Proc. International Conf. on UIMC (2011)
11. Akin, A., Baz, I., Gaemperle, L. M., Schmid, A., Leblebici, Y.: Compressed look-up-table based real-time rectification hardware. In: Proc. IFIP/IEEE 21st International Conference on Very Large Scale Integration, pp. 272-277. Turkey (2013)
12. Akin, A., Cogal, O., Seyid, K., Afshari, H., Schmid, A., Leblebici, Y.: Hemispherical Multiple Camera System for High Resolution Omni-Directional Light Field Imaging. IEEE Journal on JETCAS, vol. 3, pp. 137-144 (2013)