FPGA-BASED RECTIFICATION OF STEREO IMAGES

João G. P. Rodrigues[*]

Faculdade de Engenharia
Universidade do Porto
Email: nijoao@gmail.com

João Canas Ferreira[†]

INESC Porto, Faculdade de Engenharia
Universidade do Porto
Email: jcf@fe.up.pt

ABSTRACT

In order to obtain depth information about a scene in computer vision, one needs to process pairs of stereo images. The calculation of dense depth maps in real-time is computationally challenging as it requires searching for matches between objects in both images. The task is significantly simplified if the images are rectified, a process which horizontally aligns the objects in both images.

The process of stereo images rectification has several steps with different computational requirements. The steps include 2D searches for high fidelity matches, precise matrix calculations, and fast pixel coordinate transformations and interpolations.

In this project, the complete process is effectively implemented in a Spartan-3 FPGA, taking advantage of a MicroBlaze soft core for slow but precise calculations, and of fast dedicated hardware support for achieving the real-time requirements. The implemented system successfully performs real-time rectification on the images from two video cameras, with a resolution of 640×480 pixels and a frame rate of 25 fps, and is easily configured for videos with higher resolutions.

The experimental results show very good quality, with rectified images having a maximum vertical disparity of two pixels, thereby showing that stereo image rectification can be efficiently achieved in an low-resource FPGA (with 64 KB for program instructions and data).

Index Terms— FPGA, Stereo vision

1. INTRODUCTION

Depth information of objects in an image is essential in applications such as video security, military, cinematography, robotics and medical imaging. The process of recovering this information uses information from two images and triangulation operations to determine the depth of objects in the scene.

The triangulation between images from two cameras can be accomplished by finding a corresponding point or object viewed by one camera in the image obtained by the other. In general camera configurations finding these correspondences requires a search in two dimensions, but it becomes an one-dimensional search if the images are horizontally aligned. It has been proved that at least eight reliable matches between the images are necessary to estimate the fundamental matrix required to align the images horizontally, thereby reducing the subsequent search for correspondences to an area around the epipolar line [1, 2, 3].

Stereo image rectification is the process of transforming the images to obtain perfectly horizontal and aligned epipolar lines. For rectified pairs of images, objects have the same vertical position in both, so we just need to search along a horizontal line to find corresponding points, as is done when measuring the disparity between images in order to produce dense depth maps. Even when the rectification is not perfect, it is still very useful: the better the rectification, the smaller the search area for correspondences can be. Furthermore, the rectification of stereo video streams is sufficient for their visualization in a 3D display.

The process of rectification is normally divided in two phases: calculation of the required transformation matrices, and application of those transformations to the images. The calculation of the transformation matrices typically does not have real-time requirements, but must be very precise. However, their application to the video images is computationally heavy and has real-time constraints, making it difficult to implement in normal CPUs. However, this task can be parallelized and implemented in an FPGA.

The transformations needed to rectify the images are represented as two 3×3 matrices, one for each camera. Several authors, such as Fusiello [1] and Hartley [2], discuss the methods and mathematical background for the calculation of the matrix coefficients.

Some systems for image rectification have been previously proposed and implemented, like the MSVM-III of Jia [4], or the auto-rectification system based on the IC3D

[*]Currently with IT – Instituto de Telecomunicações. Support grant from the Masters Program in Electrical and Computer Engineering of the University of Porto.

[†]Work partially supported by research contract PTDC/EEA-ELC/69394/2006 from the Foundation for Science and Technology (FCT), Portugal.

processor of [5]. The MSVM-III implements rectification and depth map estimation in a system based on an FPGA Xilinx Virtex 2 and 3 640×480 CMOS sensors working at 30 frames per second (FPS). However, the system requires previous knowledge and input of the transformation matrices. The system of [5], based on a SIMD video processor with 320 processing elements, performs rectification and depth map estimation. The transformation matrices are calculated by the system, but are limited to translations, and do not correct small skews or rotations in the stereo kit. The system only supports video streams of 320×240 at 30 fps.

The restrictions of these systems make them unsuitable for many applications. For example, the MSVM-III requires the transformation matrix to be given, and the IC3D-based system performs rectification only through translations of the images, without considering rotation or scaling.

In our project we implemented both phases of rectification process in an FPGA-based system, using software running on a Microblaze soft core processor together with dedicated hardware. The system is able to output the rectified video streams from two uncalibrated cameras at a frame rate of 25 frames per second (fps). The image data is obtained from a stereo kit with two CMOS sensors of 640×480 pixels of resolution at the same frame rate.

The next section gives a brief overview of the proposed system organization. Section 3 describes the steps needed for estimating the matrices required for image rectification, and Section 4 explains how that information is used to rectify the images. Section 5 gives details of our FPGA-based implementation. Section 6 presents the experimental results obtained with prototype implementation, and section 7 presents some concluding remarks about the proposed system.

2. GENERAL SYSTEM ORGANIZATION

The goal of the project is to implement a full solution for stereo image rectification on a single low-cost board equipped with a Spartan-3 FPGA [6]. The system is targeted at grayscale images (1 byte per pixel) with VGA resolution (640×480 pixels).

The system is able to compensate for deviations in every image parameter, except for lens distortion, and can therefore be employed in setups with general camera locations, as long as there is no relative movement between them. As it is based on epipolar geometry, the precision of the algorithm is improved if the objects in view are not coplanar and have high horizontal disparity, thus being relatively near the camera. This requirement is important in order to allow the acquisition of relevant spatial information from the images, thereby improving the accuracy in calculations of the transformation matrices. An example of a set of images to which this method should not be applied is satellite photography (e.g. Google Earth). In this case, the objects (e.g. houses) are practically coplanar, belonging to the plane corresponding to the earth surface. In these situations, where almost no 3D information exists, a simpler method of rectification with fewer parameters should be used, like the one proposed by Jia [4].

The operation of our system is divided in two distinct phases, each with different computational and time requirements. The goal of the first phase is to determine the transformation matrices to be applied to each camera. For this purpose, the system starts by determining some tens of correspondences between points on both images. These correspondences are then refined iteratively (using criteria based on epipolar geometry), until a specific confidence threshold is met. The transformation matrices are estimated from the correspondences using a singular value decomposition (SVD) algorithm [7]. These calculations require high numerical precision to minimize errors, but have no real-time requirements. Therefore they can be implemented in software running on a soft core processor.

The second phase consists in applying the calculated matrices to the streams of videos from the cameras. This consists in a multiplication of every pixel coordinate ($[0-639, 0-479, 1]$) with each 3×3 transformation matrix, in order to obtain the coordinates of the pixels (from the incoming images) that must be used to calculate (by interpolation) the correct pixel value of the outgoing images. The result of the interpolations are the rectified images, which, in our prototype system, are then sent to the display. These calculations require many multiplications and divisions to be applied in real-time to the video stream, and thus need to be parallelized and implemented by dedicated hardware in the FPGA.

A more detailed explanation of the implementation aspects is presented in section 5.

3. DETERMINATION OF IMAGE TRANSFORMATION MATRICES

This section explains how we addressed the determination of the image transformation matrices. We first describe the approach adopted to address the task of finding good-quality correspondences between the two images (the correspondence problem). Then we describe the method used to determine the image transformation matrices from the correspondence information.

3.1. The Correspondence Problem

The correspondence problem consists in finding, for some points in one image, the corresponding points in the other image, that is, those points of the other image that belong to the same feature of the scene [8].

In the current situation, the limited instruction memory available on the FPGA (64 KB for instructions and data) severely restricts the implementation of the algorithm. In order to allow the system to fit in the 64 KB memory, some memory-hungry functions were implemented in dedicated hardware, as explained in Section 5.2. The softcore keeps almost no temporary information about the images, besides the list of candidates and possible correspondences.

The correspondence problem was simplified by splitting the evaluation into some small weighting functions. The various steps of the correspondence search implemented in the current system are described next. They are applied to a pair of images taken simultaneously from the cameras.

[Step 1.] One image is selected as the reference image. If the cameras have different characteristics, the image chosen should be the one with better quality.

[Step 2.] Every 5×5 pixel block from the reference image is analyzed with a non-linear algorithm.

First, a simple detector discards blocks containing strings or edges. This detector searches the block for straight lines (in any direction) made of pixels with similar luminance. The reason to skip these block features is that they usually repeat themselves in the neighborhood, creating a pattern and leading to many possible matches in the other image.

Then, the "quality" (Q) of each block is calculated as the maximum between the number of pixels that are darker than the central one, and the number of pixels that are lighter ($0 \leq Q \leq 24$). This method is similar in nature to the census transform presented in [9].

Pixels with a luminance value similar to the central pixel (with a difference in value of less than 20) are considered as having the same brightness, and thus ignored when calculating Q. This fact is important to suppress small Gaussian noise that is always present in images. Blocks with a $Q < 6$ or $Q > 22$ are not considered further, because they are very likely caused by image noise or memory access errors.

[Step 3.] The image is divided in zones of 80×60 pixels and only the candidates with higher Q from each zone are selected. This division is important to obtain correspondence matches dispersed throughout the entire image, improving the precision of the transformation matrix. This effect is further explained and motivated in Section 3.3.

[Step 4.] For each candidate 5×5 block, a search for a match is performed in the other image. The search area is defined as a rectangular block around the same coordinates, with small horizontal offset, and is iteratively reduced around the epipolar line. The initial relative coordinates of the search area are $\{[-50, 150], [-60, 60]\}$, but can be tuned for different stereoscopy kits. The negative value of 50 in horizontal disparity is due to the fact that in our kit the cameras are not parallel, but are pointing slightly to the middle, causing negative disparity in objects very far from the cameras.

The similarity of the candidate blocks between both images is calculated based on the following simple weighting functions. The 10 most similar (lower result) candidates for each block are saved in memory.

- A linear comparator: the sum of the absolute differences (SAD) between both blocks. This function is sensitive to luminosity differences, but susceptible to noise.

- A non-parametric comparator: based on a modified census transform [9], but made ternary to allow for noise suppression. This method is similar to the quality metric used before. Each pixel of a block is compared to the central pixel in terms of luminance, and a ternary number is saved, indicating if it's brighter, similar (threshold of 10) or darker. The result is compared with the other block's sequence of ternary values, and the "differences" are added. The difference is calculated as 0 if both values are equal, as 1 if one is similar and the other is not, and 2 if one is darker and the other is brighter. This method detects structural differences in the blocks, even thought they might be similar in luminance and passed the previous test with a good score. This measure can also be very useful in detecting similar blocks when the perceived luminosity of an object is different between images, as may happen due to reflections and other artifacts. A weight of 20 for this measure is used, since the result range is $[0 - 49]$, compared to $[0 - (5 \times 255)]$ from the previous method.

- The distance to the epipolar line estimated in the previous iteration. We assume the cameras are placed horizontally side-by-side, so we use a horizontal line in the first iteration. Because of this we also assume the epipolar lines are almost horizontal, and so we use the difference in the vertical coordinates instead of the Euclidean distance. This measure has weight 2 in the calculations.

- To break ties, we use the quality Q of the blocks calculated previously. This factor has a weight of -1, since a higher quality improves the confidence on the blocks.

[Step 5.] The possible matches for each candidate are refined, but this time using a block size of 15×15 pixels. The similarity of the bigger block is calculated as the sum of the similarities of the 9 smaller 5×5 blocks, using the same algorithms as previously. For each candidate, the similarity of the two best matches is saved for the next step.

[Step 6.] The unicity of each candidate is calculated. This factor measures the confidence associated with a correspondence pair. For each candidate coordinate, it is calculated as the difference in similarities between the two best matches on the other image. This results in a high unicity when there is only one clear match.

This step eliminates errors in patterns and repetitive textures, giving more importance to unique characteristics.

[Step 7.] For each zone of the reference image, only the two correspondence pairs with higher unicity are added to the final list of pairs.

[Step 8.] The list of pairs is iteratively refined (be repeating from step 5), until the search area is significantly reduced around the epipolar geometry. In each iteration the height of the rectangle is halved, up to a minimum of 3 pixels, resulting in a minimum search area with a height of 7 pixels. The epipolar geometry is estimated using the 8-points algorithm (as described in the next sub-section).

The iterative process will stop with failure when the algorithm finds less than 8 pairs in the current search area or the estimated epipolar geometry is far from horizontal (we previously assumed horizontal placement of the cameras). In this case the cameras are reactivated for a few seconds, two more images are taken and the process is restarted from step 1.

After 10 iterations with more than 8 correspondences, these are considered very reliable and will be used to calculate the transformation matrices.

3.2. Transformation Matrix Calculations

The transformation required to rectify the images is given in the form of a 3 x 3 matrix, for each camera. The coordinates of at least eight correspondence pairs are needed in order to estimate the epipolar geometry and to calculate the transformation matrices.

In this project we need to determine, for each coordinate of the final rectified images, the coordinates of the pixels from the (unrectified) images that must be used to calculate (by interpolation) the new pixel value. For this we have to determine one matrix for each camera as given by equation 1.

$$H = D \cdot [C \cdot T \cdot G \cdot R]^{-1} \cdot N \tag{1}$$

The matrix **H** represents the complete transformation to be applied to the video stream. This formulation extends the H^{-1} found in [2], to better account for the stereoscopy needs: our matrix **T** preserves depth information and the common area of both images is maximized by the matrix **C**. In addition, the normalizing and denormalizing matrices are already included to improve precision.

N and **D** are the Normalizing and Denormalizing matrices. These matrices put the coordinates in the $[-1, 1]$ range, in order to improve the precision of the 8-points algorithm as described by Hartley [2]. They are determined by the size of the images. For our prototype, they are:

$$N = \begin{vmatrix} 1/320 & 0 & -1 \\ 0 & 1/320 & -0.75 \\ 0 & 0 & 1 \end{vmatrix}, \ D = \begin{vmatrix} 320 & 0 & 320 \\ 0 & 320 & 240 \\ 0 & 0 & 1 \end{vmatrix}$$

R and **G** are the matrices with the same name described by Hartley [2]. These matrices send the epipole of the image to the point at infinity in the horizontal axis, making the epipolar lines horizontal and parallel between them.

They are calculated using the 8-points algorithm and a C implementation of a homogeneous equation solver by SVD taken from [10]. Performing the SVD in the coordinates of the correspondence pairs produces the Fundamental Matrix, which describes the epipolar geometry of the images. Performing SVD on the fundamental matrix and its transpose produces the coordinates of the epipoles of both images. The matrices **R** and **G** have the form:

$$R = \begin{vmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix}, \ G = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{-1}{f} & 0 & 1 \end{vmatrix}$$

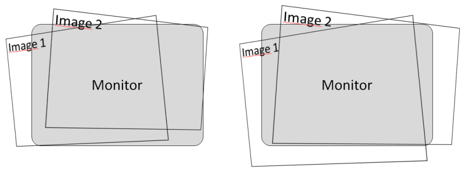

Fig. 1. The displayed area before (left) and after (right) application of matrix C.

where f is the distance of the epipole to the origin, and θ is the angle between the epipolar and horizontal lines.

T is a matrix of scaling and vertical translation that makes the epipolar lines coincide in both images. Hartley [2] performs this task by minimizing the square of the distance between corresponding points. In this project we decided to only minimize the distance between vertical coordinates, since the horizontal distance contains depth information that should not be altered. Matrix **T** has the form:

$$T = \begin{vmatrix} k & 0 & 0 \\ 0 & k & d \\ 0 & 0 & 1 \end{vmatrix}$$

For its calculation we need to apply the previous matrices to the original coordinates, and then find k and d, so that $Y \cdot k + d = Y'$, where Y and Y' are the vertical coordinates of the correspondence pairs , k is the scaling factor and d the vertical translation. This is the same as solving $Y \cdot k - Y' + d = 0$, which is a homogeneous system solvable by SVD. Horizontal translation should not be corrected because it represents the spatial information and therefore should be preserved. This matrix is only applied to the images of one camera, in this case the one from which the candidates (Y) were retrieved.

C is a matrix that should maximize the visibility of the common area between the images of both videos. It is very useful for stereoscopy, since only the common area can be analyzed. This matrix is the same for both cameras and consists only of a scaling (s) and translation factor in both axis(xt and yt). It has the form:

$$C = \begin{vmatrix} s & 0 & xt \\ 0 & s & yt \\ 0 & 0 & 1 \end{vmatrix}$$

The matrix C can be trivially calculated from the other matrices. In our implementation it can also be interactively set by the user. The effect of applying matrix C is a maximization of the common area of both images, as shown in figure 1.

3.3. Evaluation of the Matrix Calculation Procedure

The calculation of the matrices described in the previous subsection was simulated and shown to be very reliable.

For the simulations, a list of random coordinates (with variable size) was used to represent the correspondence

Number of pairs	9	25	100
Error range for:			
Almost coplanar image	50 - 57	7 - 11	1.7 - 2.3
Depth-rich image...	10 - 14	1.8 - 2.9	0.6 - 0.8
...with dispersed points	5 - 8	1.5 - 2.6	0.5 - 0.7

Table 1. Error in coordinates recovered by using **H** with rounded distorted coordinates

points. A second list, representing the second image, was created by randomly changing horizontal coordinates, thus introducing a variable horizontal disparity up to a predefined depth threshold. Then, different random distortion matrices were applied to the two coordinate lists. In all cases, the methods described in the last subsection could successfully and precisely retrieve the distortion matrices, given only the lists of distorted coordinates.

Another set of simulation was performed with the distorted coordinates rounded to the nearest integer. To avoid the rounding problem in real images, it is necessary to use mechanisms for feature detection with sub-pixel accuracy, which have high complexity and are computationally demanding. The effect of coordinate rounding was to introduce the errors in the recovered coordinates as reported in table 1. The errors were calculated by measuring the maximum Euclidean distance between the original coordinates before distortion and the coordinates after rounded distortion and rectification.

As can be seen from the table, increasing the size of the list and the maximum horizontal disparity allows the precision of the calculated matrices to be greatly improved. Furthermore, when the coordinates of the original list are spread throughout the image, very good results can be obtained with a few number of coordinates.

The simulation results were used to confirm that our system handled a sufficient number of correspondence points for ensuring the quality of the final result, and to motivate the inclusion of step 3 of the search procedure (cf. section 3.1).

4. REAL-TIME RECTIFICATION OF THE STEREO VIDEO STREAM

In this phase, the transformation matrices have already been calculated and saved in registers, and must be applied to the video streams in real-time. The implementation of this process consists of the following steps:

1. For each coordinate $[X, Y] \in [0-639;\ 0-479]$ multiply them by both transformation matrices. The results are the homogeneous coordinates of the points to interpolate from the images.

$$[X, Y, 1]^T \cdot H = [U, V, W]$$

2. Transform the homogeneous coordinates into Cartesian coordinates. This requires a division.

$$[X, Y] = [\frac{U}{W}, \frac{V}{W}] = [U \cdot \frac{1}{W}, V \cdot \frac{1}{W}]$$

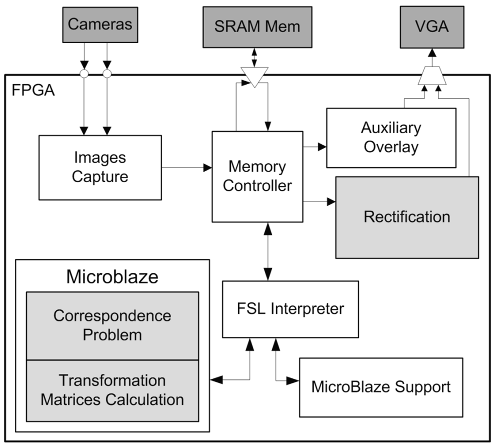

Fig. 2. Global overview of the system.

3. Read from memory the four pixels surrounding each calculated coordinate.
4. Perform bi-linear interpolation.

We chose to perform the bi-linear interpolation due to simplicity, but other methods could be used. As the CMOS sensors available for this project use a Bayer filter [11], the images are already low-pass-filtered, making the loss in quality caused by the interpolation almost negligible.

5. Send the rectified images to the monitor, memory, or other output device.

More information about the implementation of these steps is available in the next section.

5. FPGA IMPLEMENTATION

The methods and algorithms proposed were implemented in a Spartan-3 XC3S1500 FPGA, together with the support modules and controllers to communicate with external resources. The full system is shown in Figure 2, and has a system clock of 100 MHz for most of the modules. The auxiliary modules are represented in white, the external resources in dark gray, and the modules for the two main phases in light gray.

5.1. Auxiliary Modules

The cameras use CMOS sensors with embedded Bayer filter [11] and provide an 8-bit luminance channel. The systems supports a 25 fps image rate (640×480 pixels) with a pixel clock of 12.5 MHz. As the Y channel is obtained by a demosaicing filter, the black-and-white images have a lower real resolution. This fact makes image processing harder, but has been taken into consideration during the development. The cameras are fixed side-by-side, but the resulting images show some vertical disparity (cf. Fig. 4), due to some slack in the connectors and differences in the sensors themselves.

A module was implemented to handle the communications and synchronization with both cameras. This module outputs a common clock of 12,5 MHz and a reset signal to both cameras, synchronizes image reception and sends the data to the memory controller.

For display, image data is sent to an external VGA module that has having 3×512 KB FIFOs (one for each color) and an RGB output.

Both the Auxiliary Overlay module (during the first phase) and the Rectification module send images from the on-board SRAM to the VGA module. These two modules use a new way to evaluate pairs of images, based on traditional 3D display mode: they create a bi-color image, on which each source image is assigned a different color channel (red or blue). Without glasses, we easily compare the objects position on both images, and thus the precision of the rectification method. When looking at the images with colored 3D glasses it is possible to confirm the increase in visual quality after the rectification. In this black and white paper, it is not possible to see these advantages. In our prototype, the third color and FIFO (green) is only used during the first phase to display information about the correspondence problem and epipolar geometry estimation.

Each image occupies about 300 KB, so it does not fit in the internal FPGA block RAMS (BRAMs). To simplify the different image processing parts, the images received are saved in a 2 MB external SRAM (32 bit data bus). The memory controller generates a memory clock of 50 MHz, and arbiters the access to the external memory according to a fixed priority scheme. The scheme awards higher priority to the writing operation for data from the cameras; next, comes the reading of data to be sent to the display or rectification module, and finally come microprocessor accesses (for the non-real-time image processing of the first phase).

The writing and reading operations are performed 4 pixels at a time in order to optimize the 32 bit bus utilization. This allows us to make $(50\,MHz/12.5\,MHz)\times 4) = 16$ memory operations in the time necessary for the cameras to deliver 4 pixels each. Two of these operations are required to save the streams from both cameras, leaving 14 memory operations to be used by the rectification module as described in Section 5.3.

5.2. Processor and FSL Interpreter

The different modules are controlled by the MicroBlaze soft-core micro-processor (from Xilinx), through the fast simplex link (FSL) interface and our own FSL Interpreter module. The FSL Interpreter works at the system clock of 100 MHz, but the soft-core and FSL link are connected to a 50 MHz clock.

The FSL Interpreter has the ability to (de-)activate specific functions or modules, such as selecting operating mode and freezing the images in memory. It also implements many support functions for MicroBlaze operations, in order to reduce processing time and save instructions and data memory. Most of the image processing operations carried out by the processor are not performed, but consist in sending commands and analyzing the response from the Interpreter. This module implements most of the repetitive image processing functions, like each block's Q calculation, and the SAD and Census comparison of blocks. This approach greatly increases the performance in the software. For example, the MicroBlaze only needs to send one command with coordinates to get a similarity between two 5×5 pixel blocks.

For square root operations, we used a dedicated CORDIC core provided by the Xilinx Core Generator. It improves the performance of the SVD implementation significantly, because the SVD calculation requires many of these operations and they take very long to be performed in software. The CORDIC module uses the highest possible precision (48 input bits and 24 bits). The hardware implementation of the square root function also allowed us to save more than 10 KBytes of instructions memory.

The MicroBlaze microprocessor controls the execution of the calculations described in section 3. It is a 32 bit integer-only processor core, which runs at 50 MHz clock, and only occupies 1400 slices (about 10% of the FPGA). However, the FPGA only has 64 KB internal memory to be used for instructions and data. The procedures implemented on the MicroBlaze have no real-time restrictions, but we tried to keep the execution time within acceptable bound. The calculation of H^{-1} takes between 30 seconds and about 3 minutes to execute, depending on the number of iterations required in step 8 of the correspondence problem (section 3.1), which itself depends on the amount of depth-rich objects in the images analyzed.

5.3. Rectification Module

The rectification module must apply the calculated transformation matrices in real-time to both video streams. The memory operations are performed in groups of 4 pixels, and each pixel arrives to the memory at a pixel clock rate of 12.5 MHz. Therefore, each group of 4 pixels must be processed in $100\times4/12.5 = 32$ clock cycles. Since there are a lot of memory and mathematical operations, including a division, we split this module in 5 different concurrent state machines, all running at a speed of 100 MHz, as shown in Fig. 4. Each state machine implements one of the steps of the rectification procedure described in Section 4.

The synchronization between the different state machines is performed by shared registers, except for the first two. There must be an exact 32-cycle delay between them, because of the constraints imposed by the CORDIC divisor. Each of the other machines signals the next one when it is ready, and waits for a signal from the previous one, at which time the needed data registers are copied between them. There is not a rigid synchronization after state machine 3, because this state machine performs memory operations that can take a varying number of cycles, depending on the state of other modules

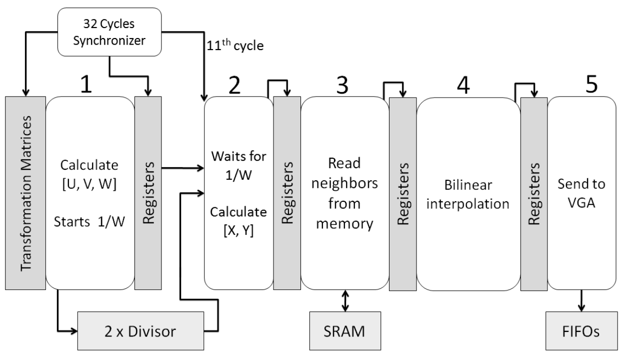

Fig. 3. Rectification module organization

(up to a certain maximum).

The first machine starts by calculating the transformed coordinates of the group of 4 pixels for both images, according to $[X, Y, 1]^T \cdot H = [U, V, W]$. This machine also starts the $1/W$ divisions on the two CORDIC divisor modules. Each CORDIC divisor has a 39 cycle latency and accepts new input values every 4 cycles.

Every 32 clock cycles the second machine reads the coordinates and waits for the result from the divisor, which arrives in the 11^{th} cycle, 39 cycles after the beginning of the division operation.

The remaining state machines perform the multiplications described in Section 4. Since there are only 14 memory access slots, we decided to always read 6 groups of pixels from each image, leaving 2 memory access slots unused. The 12 groups of four pixels are read from memory by state machine 3, which takes about 29 cycles to complete.

Due to the number of groups of pixels read from memory, there are some limitations on the maximum image distortions that cannot be compensated.The images cannot be rotated by more than 18 degrees, and the scaling factor should always be equal or less than 1. The first limitation is easily met by our setup, where the cameras are placed with almost no rotation between them. The limitation on scale is always met, since the matrix **C** will zoom in on the common area, which is always equal or smaller than the smaller image.

The last two machines are responsible for the interpolation and the forwarding of the results to the VGA module. As the interpolations require many sums and multiplications, we implemented them in a separate module with registered inputs and outputs and a slower clock of 50 MHz.

BRAMs	64 KB of 64 KB	100%
Slices	9,468 of 13,312	71%
LUTs	15,969 of 26,624	59%
Slice Flip Flops	10,788 of 26,624	40%
MULT18X18s	23 of 32	71%

Table 2. Resource utilization for Spartan-3 XC3S1500

Table 2 shows the FPGA resource utilization of the final system. Resource utilization by the memory controller and other auxiliary modules is almost negligible. All the BRAMs were used by the MicroBlaze core, which also used about 1400 slices. Most of the other resources were used in the rectification module, and a small number in the DIVISOR and CORDIC modules.

6. EXPERIMENTAL RESULTS

The FPGA used for implementation was a Xilinx XC3S1500, but the system is adaptable and easily ported to other FPGAs or cameras with different characteristics. The complete process was successfully implemented, meeting the time requirements by exploiting the hardware resources of the FPGA.

The proposed correspondence algorithm has been thoroughly tested in the development system. The cameras used have significant lens distortion and are Bayer filtered, but, even so, the algorithm was capable of detecting correspondences with good efficiency: this algorithm exhibited around 10% of false correspondence pairs after the first iteration, allowing it to successfully iterate and eliminate most of the false positives.

Although the cameras in our setup are visibly aligned, the original images are clearly unrectified, as we can see in image 4. The bi-color image display approach was used to visually evaluate the results. Results were also analyzed on a computer by freezing the video stream after rectification and downloading it from the board's external memory through a serial connection. In this way, object positions on both images could be easily measured, and thus the precision of the rectification method confirmed (cf. Figure 5).

The vertical disparity still present after rectification matches the precision expected from the simulations. When about 50 pairs of good correspondence points are used, the system rectified the videos with a maximum error of 1 pixel. In general, there were always between 20 and 60 pairs (with 1 or 2 bad correspondences) and the resulting error was less than 2 pixels, as seen in Fig. 5. In previous works [4, 5], the authors did not present the systems' precision for maximum vertical disparity of the rectified images.

The interpolation resulted in a visually-lossless rectified image when compared to the original images.

7. CONCLUSIONS

The proposed solution for the correspondence problem is reliable, even with such a small instruction memory.The new formulation of the transformation matrices is very useful for stereoscopy, preserving depth information and maximizing the common image area. Unlike previous implementations, and thanks to the combination of software and hardware solutions on the same FPGA platform, we were able to calculate

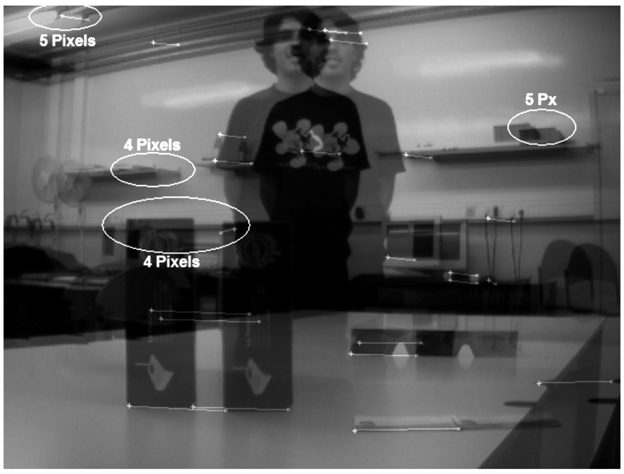

Fig. 4. Unrectified images (overlayed)

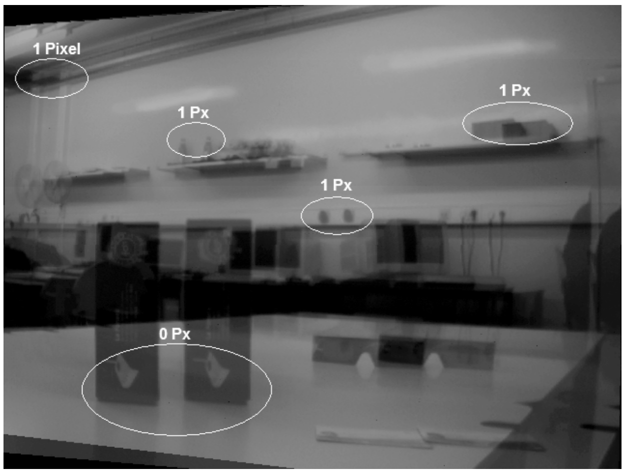

Fig. 5. Rectified images (overlayed).

the transformation matrices that rectify every non-radial image distortion.

However, the execution time of the first phase could be greatly improved with faster processors, like the PowerPC present in Virtex-4 FPGAs. The availability of a large on-chip data memory would allow a greater number of correspondence pairs to be considered and more advanced algorithms to be used, such as correspondences finding methods with sub-pixel accuracy.

The implemented rectification module is capable of rectifying the videos in real-time, with good precision even though a simple bi-linear interpolation is performed. The 2 pixels of maximum error is good enough to practically reduce the search area to a line, and thus very useful for stereoscopic applications.

We showed that a good and reliable stereo images rectification process can be implemented in a single FPGA, using only 64 KB of memory (not including the memory required to store one pair of images). A system like this could be used in low-cost, portable 3D-cameras that save rectified 3D video in real-time, or in advanced real-time 3D security systems. Although in these cases the movement of the cameras can change their relative positions, our system can periodically freeze 2 video frames in separate sections of memory and recalculate the rectification matrices, while still performing rectification on new frames using the old transformation values.

8. REFERENCES

[1] Andrea Fusiello, "Epipolar rectification," March 2000, `http://profs.sci.univr.it/~fusiello/rectif_cvol/rectif_cvol.html`.

[2] Richard I. Hartley, "Theory and practice of projective rectification," *Intl. J. Comp. Vision*, vol. 35, no. 2, pp. 115–127, Nov. 1999.

[3] Roger Mohr and Bill Triggs, "Projective geometry for image analysis: A tutorial given at ISPRS," July 1996, `http://lear.inrialpes.fr/people/triggs/pubs/isprs96/isprs96.html`.

[4] Yunde Jia, Xiaoxun Zhang, Mingxiang Li, and Luping An, "A miniature stereo vision machine (MSVM-III) for dense disparity mapping," in *17th Intl. Conf. Pattern Rec.*, 2004.

[5] Xinting Gao, R. Kleihorst, and B. Schueler, "Implementation of auto-rectification and depth estimation of stereo video in a real-time smart camera system," in *IEEE Conf. Comp. Vision Pattern Rec. Workshops*, 2008, pp. 1–7.

[6] Xilinx, *Spartan-3 Generation FPGA User Guide*, Dec. 2009.

[7] Dan Kalman, "A singularly valuable decomposition: The SVD of a matrix," *The College Math. Journal*, vol. 27, no. 1, pp. 2–23, 1996.

[8] D. Scharstein, R. Szeliski, and R. Zabih, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Proc. IEEE Work. Stereo and Multi-Baseline Vision*, 2001, pp. 131–140.

[9] Ramin Zabih and John Woodfill, "Non-parametric local transforms for computing visual correspondence," in *ECCV '94: Proc. 3rd Euro. Conf. Comp. Vision*, 1994, pp. 151–158.

[10] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery, *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[11] Sean McHugh, "Digital photography tutorials," Feb. 2009, `http://www.cambridgeincolour.com/tutorials.htm`.