Microprocessor and Hardware Laboratory
ECE Department
Technical University of Crete

Diploma Thesis

Design of a real time, 3D stereo vision algorithm and its FPGA-based implementation

by
Sotiris Thomas

Chania, April 2012

Contents

Acknowledgments....3
Chapter 1: Introduction....5
Abstract....5
Introduction....5
Requirements....9
Chapter 2: Related Work....11
Chapter 3: The Main Algorithm and its Enhancement....25
An overview of stereo algorithms....25
Algorithm matching costs....26
Parameter tuning....37
Chapter 4: Implementation....45
Design overview, parallelism and optimization....45
Block Diagrams....46
Notes on operation....50
Performance....50
Resources analysis....52
Chapter 5: Validation....57
References....60

Acknowledgments

I wish to thank, first and foremost, my supervisor Dr. Kyprianos Papadimitriou, who advised and guided me from the very start of my thesis, as well as my supervisor Professor Apostollos Dollas who pointed my efforts to the right direction. I also wish to thank Michalis Zervakis and Ioannis Papaefstathiou for accepting reviewing my work and participate in the examination committee. Also a thank you to my family and friends for their understanding and support.

Chapter 1
Introduction

Abstract

The ability to track objects and people in real time would greatly benefit many applications that interact with real environments. For example cars could brake automatically when detecting pedestrians or objects in harms way. Agricultural equipment could autonomously navigate fields avoiding obstacles. Security systems could track people moving through buildings or different areas. 3D Vision and its most effective implementation, stereo vision, could assist these applications. Stereo vision uses two cameras side by side to produce virtually instantaneous estimates of the distances to elements in a scene. These distances can provide a primary cue for identifying objects that stand out from the background and interpreting their shape, thus assisting object segmentation and identification.

In the present work we describe a 3D stereo vision design and its implementation that exploits effectively the resources of an FPGA. Our place-and-route design achieved a high processing rate for large resolutions, while the hardware prototype system was fully tested and validated over several data sets with medium resolutions.

Introduction

The purpose of all stereo vision algorithms is to construct an accurate depth map out of two or more images of the same scene, taken under a slightly different angle/position (Fig 1). In other words, they use the cue of stereopsis to calculate depth valued pixels. The resulting depth map is usually a grayscale image, where closer points are brighter.

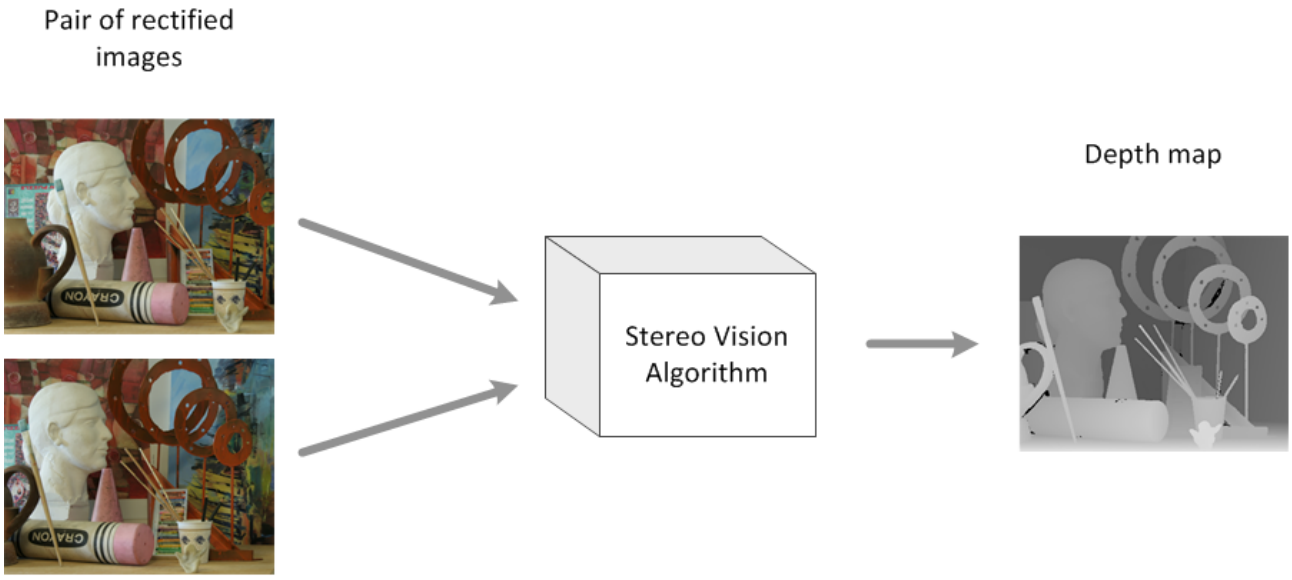


Fig 1: *Inputs-Output of a stereo vision algorithm. Middlebury 2005 Art dataset.*

In its most basic form, stereo vision uses two cameras. A point in space should correspond to one pixel in each of the two images from the two cameras. If we can find those pixel pairs that correspond to the same point in the scene, we can extract the distance of that point through triangulation (Fig 2). The difference in the position of the two corresponding pixels, which is called disparity, is directly connected to the distance of the point they correspond to. More specifically, larger disparities are connected to closer objects, while smaller disparities suggest points farther away from the cameras.

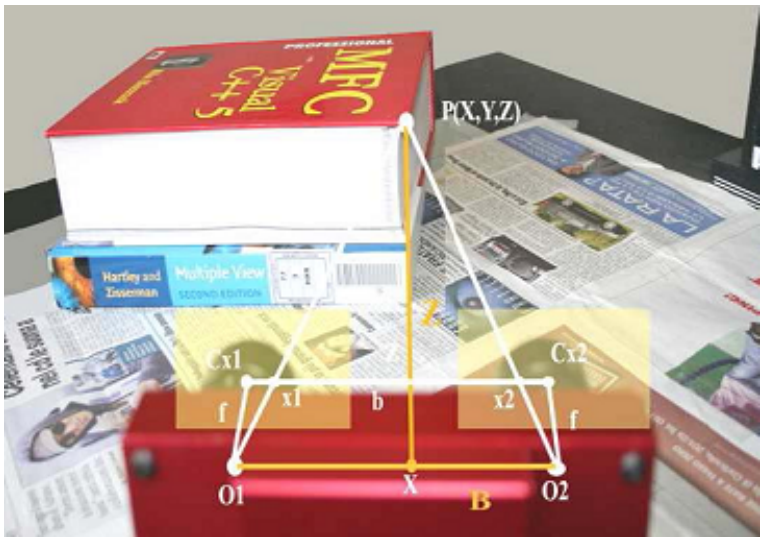


Fig 2: *Point P(X,Y,Z) corresponds to points x1 and x2 in the two cameras. Z is the unknown distance of the point, f the focal length of the cameras and b the baseline distance between the two cameras. Using these values and the coordinates of x1 and x2 in the two images we can extract Z through triangulation. Image courtesy of Stefano Mattoccia [16].*

Placing the two cameras one next to the other on the same plane, enforces corresponding pixels on the same line, called the epipolar line (the white line defined by Cx1 and Cx2 in Fig 2). In other words, disparity is reduced from a two dimensional displacement Δx, Δy to just Δx, a displacement on the same horizontal line. However, as exact camera alignment is not only difficult but also prone to possible future events, such as mechanical shocks sustained by the system (e.g when the system is used on a moving platform such as a car), lenses degradation etc, the epipolar line constraint must be dynamically enforced.

Rectification is the process of applying spatial transformations on the input images in order to bring them on the same plane, accounting for any imagers misalignment, difference in focal length and removing any camera induced distortions. This allows us to satisfy the epipolar line constraint, which, as discussed previously, ensures that the matching pixel belongs on the same horizontal line (scan-line) as the reference pixel and only in one specific direction relative to it. The disparity search space is thus reduced from 2D to 1D (see Fig 3).

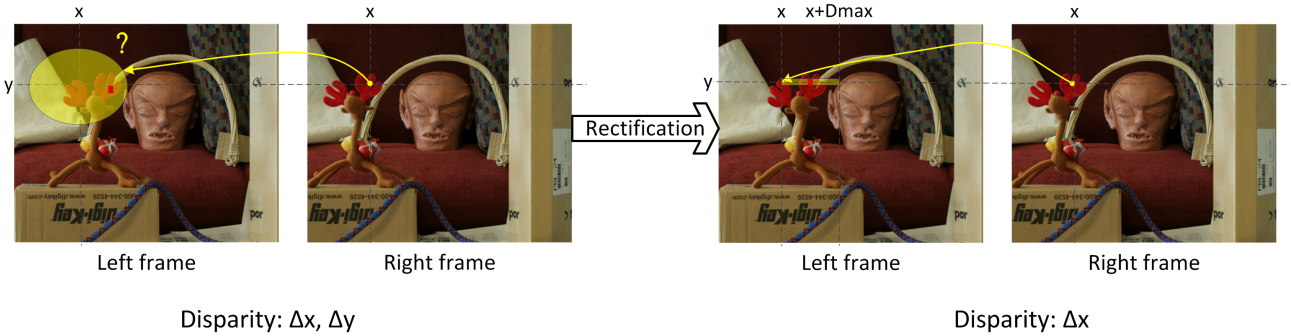


Fig 3: *Rectification reduces corresponding pixel search space from two dimensions to one, greatly simplifying the corespondence problem.*

The basic problem of finding pairs of pixels that correspond to the same point in two images, known as the correspondence problem, appears not only in stereo vision but also in other image processing topics such as optical flow calculation[1]. Although, one would expect that the reduction of the search space of corresponding pixels from two dimensions to one through rectification would make finding unique and accurate matches easy, this is not the case. In practice, many real-world "inconveniences" such as reflections, occlusions, texture-less areas, noise etc, along with the inherent similarity among nearby pixels, make this task difficult. Furthermore, real world conditions such as moving objects and poor weather pose another significant challenge especially for outdoor applications. Rain, snow and ice can alter the appearance of objects. People or objects can be stationary or moving in different directions and different speeds.

Many stereo vision algorithms have been developed over the years to tackle the correspondence problem. A taxonomy of dense stereo correspondence algorithms has been developed in Scharstein et al. [11]. The algorithm presented here produces dense depth maps, which unlike sparse depth maps, convey depth information on every pixel. It is

1 Optical flow calculation is the process of estimating the relative speed of an object by locating its position in consecutive frames of a video stream.

also important to note that the stereo vision algorithm that we discuss, is fed with rectified images. See Fig 4 for an overview of a stereo vision system.

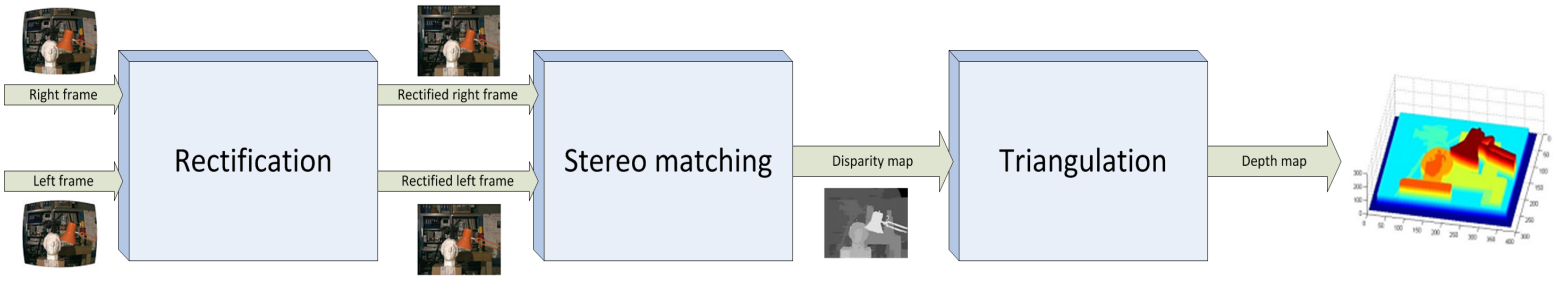


Fig 4: *General stereo vision system structure.*

The correspondence problem is made more difficult by the presence of occlusions, which prevent a pixel from appearing in both images (Fig 5). Occluded areas are areas that appear only in one of the two images and thus are effectively unmatchable by the correlation process. For this reason, any matches produced in occluded areas are statistically false. In order to match occluded areas, some form of propagation of confident disparities to uncertain areas must be performed. Usually occluded areas are presented close to object edges.

Reference image | *Occlusions marked in black*

Fig 5*: Reference image (Teddy) and occlusions map. Notice how occlusions appear on object borders and in areas out of the field of view of both images (right bar).*

Requirements

Real time applications depend on stereo censors to deliver adequate frame-rates of confident depth maps with low latencies. Desirable characteristics of a stereo vision algorithm are:

a) Robustness in different lighting conditions, where light intensity can be significantly different from one camera to the other.
b) Robustness in depth discontinuities. Object edges must be accurately and unambiguously defined.
c) Outliers correction, where algorithm false results must be flagged and handled.
d) Robustness in occlusions, where occluded areas must be detected and dealt with.
e) Robustness in low-texture areas.

The algorithm we present, in its bare-bones form, satisfies many of these constraints, while supporting easy modular addition of further enhancing features. The algorithm also allows for a large degree of parallelism, which, along with the need for bit counting used in one of its steps, makes it a good candidate for FPGA implementation.

Digital cameras usually provide the pixel values in a sequential manner, on the horizontal or vertical scan line. In order to increase their throughput, some cameras provide multiple scan lines simultaneously called taps. Many of them implement a standardized communication protocol, Camera Link. Camera Link uses a dedicated cable connection to transmit data as LVDS signals at rates 2.04 Gbit/s for the base configuration, up to 5.44 Gbit/s for the full configuration, at the maximum operating frequency of 85MHz and over a 10 meter cable. As comparison, a video stream of 640×480 of gray-scale frame pairs, requires:

```
2 x (640×480) pixels/frame x 8 bit/pixel x 30 frames/s =  147.456 Mbit/s
```

Chapter 2
Related work

[1] Computer Vision Algorithms on Reconfigurable Logic Arrays

This paper studies systolic hardware architectures for three machine vision algorithms, each one used in a different image processing task, on a Splash2 CCM (Custom Computing Machine) platform, with 16+1 FPGAs per board (may expand to more than one board). Each of these FPGAs in Splash2 constitutes a Processing Element (PE) with its own memory. The 17th is used to program the connections between the other PEs through a crossbar.

The first algorithm is employed for the important low level operation of 2d convolution. The basic idea is to convert the 2d mask into a 1d array and use the PEs to compute the partial sums. For example if the 1d mask array has k elements, then the k-th PE will compute the value of partial_sum(k-1) +pixel_value*mask[k] (where partial_sum(k-1) is already computed by its neighbor). The 1d mask array is extended with zeros and the PEs assigned to calculate those zero multiplications are simply programmed as shift registers. Comparing this implementation with others (von Neumann (C), MIMD and SIMD) it comes as no surprise that the FPGAs and Splash2 are much better (e.g on a 512×512 image with a 3×3 mask execution was completed in 13.89 ms). However no dynamic reconfiguration of the FPGAs occured during execution.

The second algorithm is about image segmentation (segmenting an image in "interesting" areas) and more precisely segmenting a document photo in text areas, image areas, etc. This algorithm has three stages: 1)application of 20, 7×7 filters on the initial image, 2)classifying the different image areas with a neural network and finally a small stage of 3)postprocessing to remove noise and place frames around the identified areas. Between the first and second stage Splash2 is reconfigured to accommodate a more efficient datapath. The first filtering stage uses the 2d convolution architecture described above. In the second stage -neural network-, each node-neuron of the network executes 2 basic tasks: It a) multiplies its inputs with weights and aggregates them (inner product) and b) passes the result to a non-linear function. Regarding the hardware mapping, these two jobs are assigned to 2 PEs respectively. The first PE computes the inner product of the input vector (size 20) with the weight vector. It carries out this computation through an accumulator. The second PE computes the non-linear function result by a Look-up Table and stores it in its memory. It is important to note that multiplications are also in general carried out by Look-up Tables because of their cost. When this process is completed for all

level-1 neurons, the host machine has the opportunity to access the intermediate results that are stored in the PEs memories. When the process finishes calculation on the final neuron level, the stage is complete. The crossbar is used to broadcast intermediate results to the rest of the neurons. Experimental data shows that this implementation reduced execution time from 250 seconds on a SPARCstation to just 3.8.

The third algorithm that was examined concerns fingerprint matching. The problem is formulated as a best match search: We are given a features database[1], including features sets from fingerprints of a population, and a features set of the wanted person. The hardware architecture is as follows: Each feature f of the wanted fingerprint is mapped to a PE. Initially a tolerance box for f is calculated and stored in a Lookup Table in the memory of the PE. This tolerance box is a set of features (x_i,y_i,θ_i) that are assumed to match with the wanted feature (x,y,θ), because the components x_i,y_i and θ_i are close to x,y,θ. During run-time, the database sends feature vectors for each fingerprint. Each vector is broadcasted in the system, with each vector element reaching each PE. If a PE matches the feature it received (finds it in its Lookup table), it drives a global OR bus to logical 1. The 17th FPGA, the one that controls the connections, listens to the Global OR bus and each time it detects a 1, it increments a counter. When all feature vectors for a specific fingerprint are transmitted from the database, the host machine reads this counter value (in other words it reads how many features matched). When this process is finished for all fingerprints in the database, the host has the number of feature matches for each fingerprint so it can easily extract the best one. This hardware implementation offers a 4 orders of magnitude improvement over the SPARCstation.

1 :Human fingerprints are characterized by ridges and valleys, which some times bifurcate or stop. This is what makes them unique in each person. A feature is defined as the set f(x,y,θ) where x,y is the coordinate of the bifurcation or ending and θ its angle.

[2] Low-Cost Stereo Vision on an FPGA

In this paper a low cost hardware architecture of stereo vision is discussed. Stereo vision uses two images taken in a slightly different angle to construct a depth map of a scene. An alternative to collect accurate depth data is to use a laser (LIDAR). However it is still very expensive for consumer applications.

The Census Transform algorithm was implemented in a Xilinx Spartan 3 FPGA. This algorithm finds the corresponding pixels pairs in two images and outputs their distance as a measure of depth. These are the steps of the algorithm in more detail: 1) For each pixel in the reference image (assume right), compute a bit vector of size 13×13=169, where each bit describes the relative difference in the intensities of the pixels in the 13×13 window with the central pixel. A bit value of 1 translates into a positive difference. 2) Repeat the first step for a range of pixels around the position of the reference pixel, in the left image (in the paper a range of 20 is used). 3) For each bit vector pair (20 pairs) compute the hamming distance. 4) Find the pair with the minimum hamming distance. The distance in the position of the two matching pixels (disparity) is inversely proportional to the true depth in that pixel.

The system is interfaced with a CMOS camera pair through two 8-bit buses and processes the incoming data on the fly, without the need of a frame buffer. The top module receives 2 pixels in every cycle, each one from each camera, and another 1-bit input which determines a new frame set. Internally, 2 blocks called xFormCensus compute the census transforms (the first one computes only the census transforms of the reference pixels whereas the second carries out the computation of all 20 non-reference pixels census transforms. A delay unit is used to synchronize the 2 blocks). A third block, named cmpLeftRight, receives the bit vectors, extracts the minimum Hamming distance and outputs the disparity of the 2 matching pixels, as a 5-bit value (maximum is 20).

The system reached 40 fps in 320×240 images, utilizing 57% of the FPGA resources and is oriented towards agricultural datasets (trees, bushes etc). However, lack of post-processing and the limited upper disparity of 20 come as disadvantages.

[3] An FPGA-Based Implementation of Spatio-Temporal Object Segmentation

As the title suggests, this paper discusses image segmentation on video data streams. The algorithm has 3 distinct stages:

1. Motion detection
2. Thresholding
3. Edge detection

The FPGA of choice is Virtex 2 Pro XC2VP20.

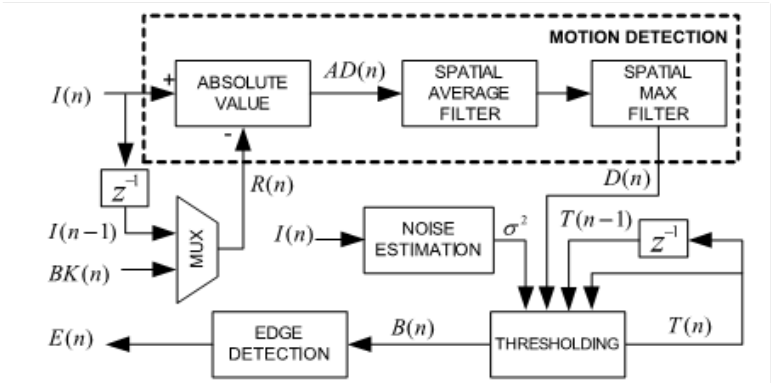


Fig. 1. Block Diagram of the Object Segmentation [9].

A series of operations is described according to the general algorithm architecture shown on the right, which result in an image with highlighted objects. The general hardware architecture is shown in Fig 2.

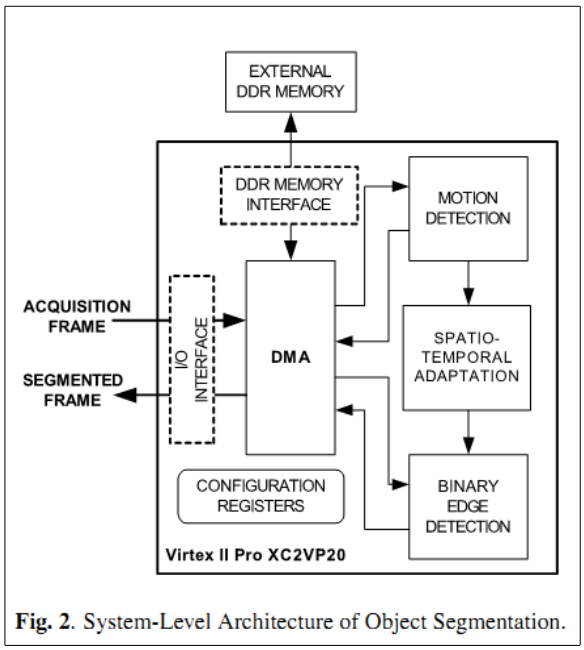


Fig. 2. System-Level Architecture of Object Segmentation.

Every data transfer passes through the DMA. Initially a video frame is sent to the DMA which consecutively broadcasts it on the external DDR memory and the Motion Detection module. At the same time, DMA also reads the previous frame I(n-1) or a background image BK(n) from the DDR memory and sends it to the Motion Detection module. Motion Detection module output, D(n), is routed back to the memory and is also send to the Spatio-temporal Tresholding module. Because the Spatio-temporal Thresholding module requires a full frame time period to produce a valid threshold, the frame that is currently being processed is buffered. While buffering the previous motion-detected frame is read from the memory and is sent to the last processing block for morphological edge detection. All these data transfers are handled seamlessly for the processing modules by the DMA. Each module implements an ad-hoc solution for the corresponding algorithmic component. The system achieved 7.5ms per 1024×1024 frame at 133MHz, performance characterized as real-time. It has a total throughput of 133Mpixel/s, using 60% of the FPGA area. Moreover it is very flexible as many of its parameters can be set at run-time. However, even though it is scalable, the DDR memory presents a potential bottleneck.

[4] Small Vision Systems: Hardware and Implementation

In this case, a stereo vision algorithm was implemented on a DSP platform called Small Vision Module (SVM). The algorithm uses SAD correlation on images filtered by a Laplacian of Gaussian (LoG, filtered image has zero crossings on edges). As LoG is an edge detection filter, it allows for robust lighting independent correlation, as edges remain the same no matter the lighting conditions. This algorithm was preferred over Census transform with hamming distances, as it does not require bit-counting hardware which is usually not available on microprocessors and DSPs.

To further enhance the resulting image, the algorithm incorporates (a) an interest operator (assigns confidence to textured areas) and b) a left/right check which helps remove false matches on depth discontinuities. SVM also offers a variable search disparity range (16,24 or 32 pixels) and automatic camera calibration and rectification of input images. Auto-calibration is required in order to satisfy the horizontal epipolar line constraint of the stereo vision algorithm and is done by maximizing the left/right matches and a disparity smoothness measure through horizontal and vertical offsets. This process takes a few seconds to complete as it performs a hierarchical search from coarse resolution to fine resolution at a few image points where the measures mentioned are highest. When vertical and horizontal offsets are calibrated, the same measures can be used to compensate for vergence and rotation distortions.

The updated version of SVM, SVM II, using a Texas Instruments DSP (TMS320C60x) running at 200 MHz, achieved a thirty-fold increase in performance in simulation over its predecessor, SVM, which run at 8fps with 160×120 images. Cameras with better SNR can increase matching accuracy. DSPs offer the best performance vs flexibility vs power consumption tradeoff whereas FPGAs offer the best performance and even microprocessors with SIMD instructions can achieve real-time performance and offer the biggest flexibility with the cons of high cost and power consumption.

[5] FPGA Based Hardware Implementation of Image Filter With Dynamic Reconfiguration Architecture

The system implements filtering and noise removal on an image by using a genetic algorithm. It also uses coarse-timescale reconfiguration as a means to adapt to the slow changing dynamic conditions (lighting conditions, noise etc) and requirements of the application (low latency vs high accuracy). Apart from the said slow changing speed of the variations, another reason for choosing coarse-timescale reconfiguration is the millisecond order of FPGA reconfiguration times.

As inputs, the filtered and the original images are given. The system stores these two images on a buffer and then computes the initial population by producing 16 250bit chromosomes in 16×25 cycles (25 cycles to generate 250 random bits from a 10bit random number generator). Each chromosome is used to filter the distorted input image, with a filter size of 3×3. The filtered image is compared to the original image and a fitness function is evaluated. Based on that fitness function value, the best chromosome is selected, which will be used to produce the new generation of chromosomes, through reproduction, crossover and mutation operations. In detail, the new chromosome population will consist of 15 new mutated children of the fittest chromosome along with the mutated children from the reproduction of other random chromosomes. The process continues until the quality constraints are met or the iteration limit is reached.

The platform used includes an FPGA 600K Spartan-IIE, a 4Mx16 data memory and a ROM 1Mx16, analog input with ADC, DAC for video output and RS232/PCI interfaces. According to conditions, a less complex hardware design can be swapped into the FPGA to achieve the same accuracy.

For Gaussian noise the system achieved better results than the Gaussian filter. Furthermore the FPGA implementation achieved a 400-fold acceleration compared to a C implementation.

[6] A Real-Time Large Disparity Range Stereo-System Using FPGAs

In this paper the stereo vision algorithm of Local Weighted Phase Correlation (LWPC) is implemented on an FPGA. This algorithm has the advantage of very large disparity search ranges, as far as 128 pixels. The key to achieve this is the use of two windows: One that defines where a pixel is in the current frame, based on the estimation of its position on the previous frame (uses the assumption that on 30 fps, the disparity of a pixel won't change dramatically between consecutive frames) (Primary Tracking Window, PTW) and another that does a random search outside PTW in order to deal with occasions where the disparity has changed a lot between frames (due to the speed of an object or the insertion of a new object in the scene) (Secondary Roving Window, SRW). The system also employes pre-processing (image rectification) and post-processing (left/right check) steps, as well as sub-pixel estimation through interpolation. The search for a new disparity is done in three orientations and scales, from which the most confident one is selected.

The block diagram of the system is shown below.

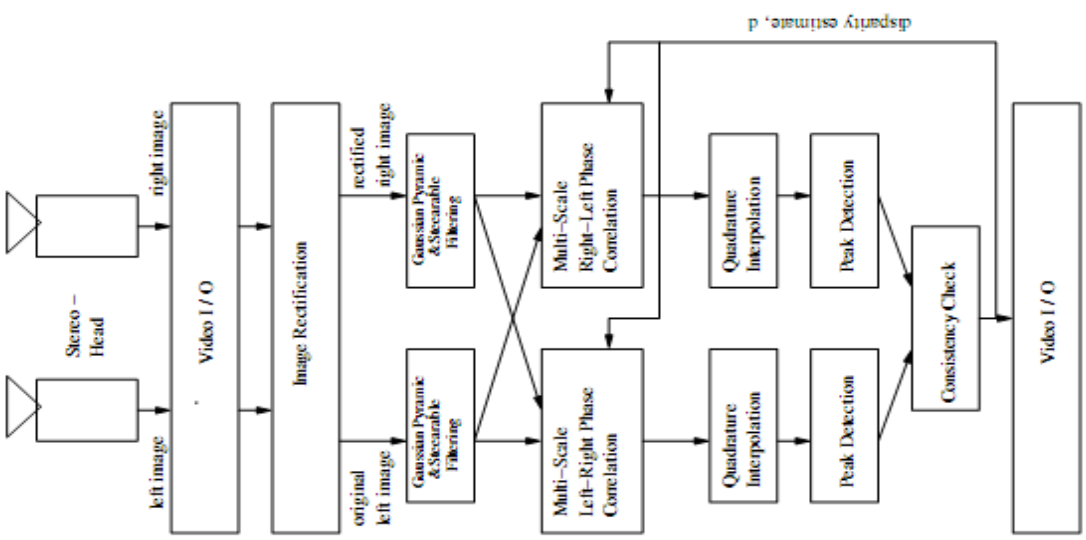


Fig. 2. High-level architecture of the stereo system

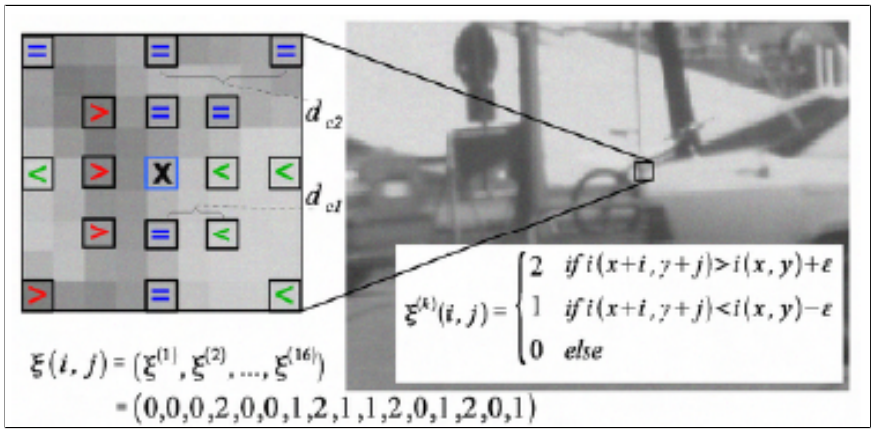
As the position estimation for each pixel is propagated from frame to frame, the algorithm has the risk of getting stuck to a local minimum. In such a case the STW helps to recover by replacing the PTW (the confidence for the STW will be greater than that of the PTW in such a case).

The platform of implementation was Transmogrifier-4, which contains 4 Altera Stratix S80 FPGAs, suitable video interfaces (eg NTSC and Firewire camera interfaces etc) and 2GB DDR RAM for each FPGA. Performance wise, the system is rated at 30fps in video streams of 640×480 pixels.

[7] High performance FPGA based Optical Flow calculation using the Census Transformation

The paper examines a variation of the census transform algorithm, adapted for computation of the Optical Flow from a single video stream. The Optical Flow image describes object motions, by means of motion vectors. In short, the algorithm finds the corresponding pixels in two subsequent frames of a video stream, and paints a vector from the starting position to the ending one. Given that information, a velocity vector can be extracted. The procedure is the following: For each pixel in frame t_k, a signature string is calculated using the modified census transform. The same thing is repeated for the next frame t_{k+1}. The pixel identities are then used to match each pixel in t_k with its corresponding one, in the subsequent frame t_{k+1}.

The modified census transform defines a bit string of 0, 1 and 2 (instead of the plain version, consisting purely of 0 and 1) for each pixel. The exact formula of this calculation is described in the image on the right. The intensity of the central pixel i(x,y) is compared to that of certain pixels in its neighborhood i(x+i,y+j). If i(x+i,y+j)>i(x,y)+ε, a two is written on the bit string. If i(x+i,y+j)<i(x,y)-ε, a one is written instead. Otherwise a zero is written. The number ε is given as a parameter to the algorithm. Moreover a sampling distance is also set (d_{c1}, d_{c2}) which denotes which pixels are compared to the central one. The correlation step demands that two bit strings are identical and unique in order to have a match (instead of the minimum hamming distance used in the classic census transform). It is also desirable to tune the parameters ε and d_c in such a way as to have many unique pixel signatures generated at each frame (in order to have potentially more matches) and at the same time have a small amount of incorrect matches. It is shown that adjusting these parameters to produce one effect involves tradeoffs to the other, so they must be carefully set.



An implementation based on software and one based on hardware were described and compared. The two solutions were not identical, as they were both optimized for their platform. Their most important common features are 1) that they both use two matrices, one for the frame t_k and the other for frame t_{k+1} and 2) that they both calculate the census signatures for all pixels in each of these frames:

In the software implementation (using a Core 2 Duo 1.86 Ghz) the matrices are indexed by the pixel signatures and each matrix cell contains the coordinates of the pixel (x,y) along with a counter denoting how many times that specific signature was generated. The algorithm produces the pixel signatures for the two subsequent frames and fills up two matrices. At the end, the pixels that match are the ones that have the same signature which was also generated just once (corresponding rows with both counters set to 1). Thus

in regard to the correlation step of this implementation, we can say that it uses a global matching scheme, meaning that a pixel in t_k can match to any pixel in t_{k+1}.

In the hardware implementation (using an XC2VP30 FPGA from Xilinx with two embedded PowerPC Processor cores) the matrices are indexed by the coordinates of the pixels (in order to take advantage of burst data transfers). Each matrix cell contains just the signature of the pixel. Again the algorithm fills up the two matrices and at the matching step, it searches in the second matrix just the area around the location of each pixel in the first matrix to find a match. In other words it uses a local matching scheme, so that a matching pixel in t_{k+1} is only found in the neighborhood of the reference pixel of frame t_k.

The hardware solution uses two modules, the Census Engine and the Matching Engine, to carry out the tasks of computing a signature given a window of pixels and matching two pixels from two subsequent frames, given a window of signature values. The system also contains a DDR SDRAM memory which is used to store intermediate results (such as the signature values of every pixel in each frame, computed by the Census Module). The PowerPC processor is used to paint the motion vectors at the end of the procedure. Everything is connected to each other through the Processor Local Bus.

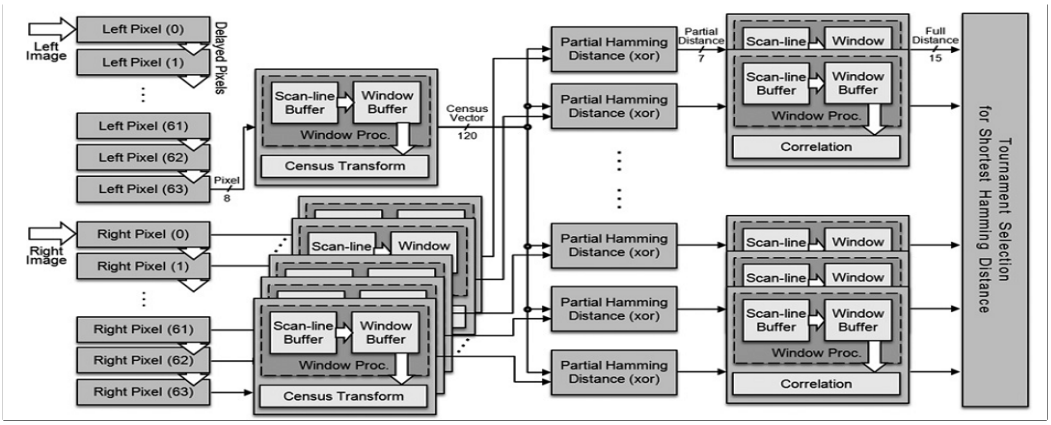
The hardware platform requires 22.17 ms to process each frame set, so approximately 45 fps is the estimated system's performance. The results also show a speedup of 1.8 in execution time from the software implementation, which takes 40.55 ms for each frame set. If the frequency is taken into account, a speed up factor of 16.15 is obtained from the HW version. The HW implementation consumes 9200 flip-flops, 13927 4-input LUTs and 50 BRAMs. Furthermore the FPGA platform offers far lower power consumption.

[8] FPGA Design and Implementation of a Real-Time Stereo Vision System

The paper presents a complete stereo vision solution, which is implemented on an FPGA. The system is synchronized with a pixel clock, which is the frequency at which new pixels are fed into it. The pixel clock is based on the frame rate and the resolution of the input images so the system is flexible in regards to these camera parameters.

The main stages of the system are image rectification, stereo local matching (using the census transform) and post-processing which enhances the quality of the result and is consisted of a uniqueness check, a left/right check, a sub-pixel estimation step and a step of spike removal.

Image rectification uses the matrices generated during the camera calibration, which is an off line procedure, to map each pixel coordinates of the original image to their counterparts in the rectified one, using reverse mapping. Subsequently, the stereo matching module undertakes the task of solving the pixel correspondence problem, using the local method of census. The module is separated in the census transform stage and the correlation stage. The census transform stage generates the census signature of each pixel on both the right and left images. It simultaneously constructs the bit string of a pixel in the reference image and all those in the disparity range on the other image. The window size of 11×11 sets the size of the bit strings to 120 bits so when computing the hamming distance, a maximum of 120 differences can be detected which translates into a 7bit number. The architecture of the module is better shown below:



The post processing module consists of the LR-check, the uniqueness test, the spike removal ad the sub-pixel estimation sub-modules. LR-check is used to remove occlusions, whereas the uniqueness test is used to determine whether the selected disparity is a unique minimum or non-unique minimum. If a disparity result passes these two checks, it is fed to the sub-pixel estimation and the spike removal phases.

The spike removal phase, assigns a label at each disparity pixel, depending on the label of its neighboring pixels and its disparity value. It then proceeds to eliminate all pixels of a label L, if they are fewer than a threshold parameter.

The system was implemented on a Xilinx Virtex-4 XC4VLX200-10 FPGA, utilizing 57% slices and 95% BRAMs. The amount of logic resources consumed by the census transform and correlation modules are linearly increased as the disparity range and the window size increase, while the other modules are affected less. A theoretical peak performance of 230 fps can be achieved with an average 17.24% of bad pixels per 640×480 frame.

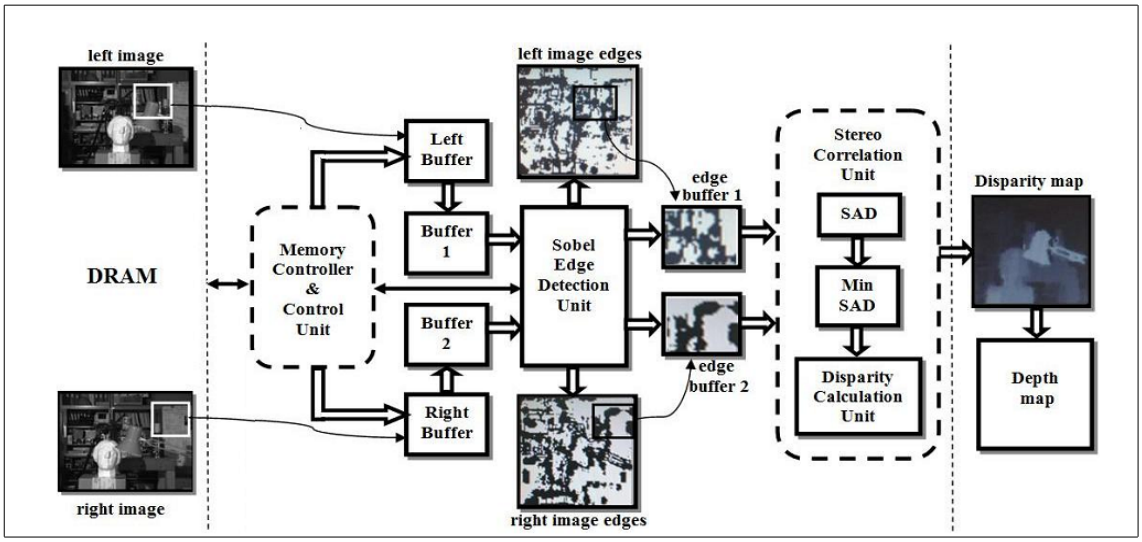
[10] Towards Hardware Stereoscopic 3D Reconstruction, A Real-Time FPGA Computation of the Disparity Map

In this paper a stereo vision local algorithm is explored and implemented on an FPGA platform. The system receives a rectified pair of frames and outputs a sparse 3D map of the scene. The process is divided into two steps: the correspondence step that involves finding pixels belong to the same point in space in the two frames and the reconstruction step which extracts the actual distance from the camera.

The correspondence step is the most computational intensive. The system uses a simple local SAD correlation method but applies it on edge detected images instead of the original dense pair. This allows for a large reduction on the data being processed and thus has a positive effect on the overall system performance, measured by frames per second. By applying a Sobel edge detection filter to both frames, the correspondence step has to be applied only on reduced 1 bit/pixel images, effectively cutting the data load to 1/8 of the initial. Aside from faster computation, this also drastically reduces memory requirements. However, doing pixel matching on edge detected images has a negative impact on quality.

The system can be parametrized in terms of correlation window size, disparity range and input image sizes, as the performance depends also on a combination of these parameters in addition to the operating frequency. Correlation window sizes has a negative effect on FPGA resources.

The FPGA of choice is a Xillinx Virtex2 Pro XC2VP30. For a disparity search range of 31, 320×240 images and a 9×9 correlation window, the system achieved 17,1 frames per second without the edge detection step and 37 frames per second with it. Below is shown the system architecture.



Comparison between Stereo Vision Implementations

	Algorithm Basic algorithm description	Features	Implementation	Resources	*Performance (fps)*	*Clock*[1]	Complexity 1-Most complex
Murphy, Lindquist, Rynning Cecil, Leavitt, Chang, Olin 2007	Non-parametric, local[2]: Census Transform, Hamming Distance	Basic algorithm	FPGA[3]: Xilinx Spartan 3 XC3S2000	• 57% of logic resources • 26/40 BRAMs	40 320×240 fps (limited by camera, theoretical 150+ fps) 8-bit gray-scale	26 MHz	4
Konolige 1997	Feature based, Laplacian of Gaussian, local: Absolute difference (SRI area correlation)	• Variable disparity search • Post-processing: Interest operator, left/right check x4 range interpolation • Automatic calibration	DSP: ADSP 2181	-	12 320×240 fps 8-bit gray-scale	33 MHz	3
Masrani, MacLean 2006	Phased based, Local: Local Weighted Phase Correlation, LWPC	• Can handle very large disparities • Pre-processing: Image Rectification • Post-processing: Left/right check x4 range interpolation	4-FPGAs Platform: Transmogrifier-4 4x Altera Stratix S80 FPGAs	Data based on the predecessor system[4] • ≈ 66644 4-input LUTs • ≈ 83026 flip-flops • ≈ 197/640 BRAMs	30 640×480 fps 8-bit gray-scale	-	1 $10x10^9$ 16×16 bit multiplications/second
S. Jin, J. Cho, X. D. Pham et al 2010	Non-parametric, local: Aggregated Census Transform, Hamming Distance	• Pre-processing: Image rectification • Post-processing: Left/right check Confidence check Sub-pixel estimation • Use of aggregated census bit string	FPGA: Xilinx Virtex-4 XC4VLX200-10	• 51.191 slices (57% logic resources) • 95% BRAMs	Theoretical 230 640×480 fps 8-bit gray-scale	12.2/24.5 MHz (for 30 and 60 fps accordingly)	2

1 Due to the parallel nature of the algorithms examined, performance depends more on the size of the FPGAs than their clock frequency.

2 In general, stereo vision algorithms are divided into local and global, depending on the range of their search for matches.

3 Census Transform algorithm is an excellent candidate for FPGA acceleration as it is highly parallel and demands bit-counting units, something that DSPs and µPs lack.

4 Data refer to the previous generation of this system (Transmogrifier-3A), as it is mentioned that there are no major changes.

A more comprehensive comparison

	Algorithm Basic algorithm description	Aggregation	Rectification	Post-processing	Features	Implementation	Parameters	Resources	Performance (fps)	*Frequency*
Our approach	AD-Census on original image	5×5	No	• Left/Right check • Scan-line belief propagation	• Disparity/window size/frame size agnostic design	Xillinx Virtex5 XC5VLX110T-1	Disparity Range: 0-63 Window Size: 9×9 Aggregation: 5×5	Slices: 82 % FlipFlops: 60 % LUTs: 54 %	650 640×480 fps 8-bit grayscale	201 MHz
Hadjitheophanous et al 2010	SAD on Sobel edge detector output	No	No	-	• Disparity/window size/frame size agnostic design	Xillinx Virtex2 XC2VP30 Pro	Disparity Range: 0-31 Window Size: 9×9 Aggregation: -	Slices: 80,2 % FlipFlops: 79,9 % LUTs: 61,1 %	75 320×240 fps 8-bit gray-scale	-
S. Jin, J. Cho, X. D. Pham et al 2010	Census on original image	15×15	Yes	• Left/Right check • Sub-pixel estimation • Spike removal	• Disparity/window size/frame size agnostic design	Xillinx Virtex4 XC4VLX200-10	Disparity Range: 0-63 Window Size: 11×11 Aggregation: 15×15	Slices: 57 % FlipFlops: 30 % LUTs: 34 %	230 640×480 fps 8-bit gray-scale	93.0907 MHz

S. Hadjitheophanous et al used a novel approach in order to reduce the computational load of a classic local algorithm solving the stereo correspondence problem. They first applied Sobel edge detection to the image pair and then performed the classic local algorithm steps using SAD as a matching cost on the edge image pair. Using edge images cuts the data load 35-55% on average, yielding significant gains in performance. This process however, has irreversible deteriorating effects on the final depth image and the authors report a qualitative 7% drop compared to the system running without the edge detector.

S. Jin, J. Cho, X. D. Pham et al implemented a complete stereo vision system. In its heart, there is a local stereo matching algorithm using the census transform as a matching cost. The coarse algorithmic flow of their solution is image rectification as a pre-processing step, followed by census correlation augmented by costs aggregation and finally the appliance of considerable post-processing, consisting of a left/right check, sub-pixel estimation and spike removal.

Our system follows closely the implementation of S. Jin, J. Cho, X. D. Pham et al. It uses a variation of census transform enhanced by aggregation to supply the bulk of the costs data, which is subsequently fed to post-processing, which includes a left/right consistency check and a basic scan-line belief propagation solution that propagates most confident disparities to inconsistent matches along the scan-line.

Reference	Mde/s	fps	Algorithm	Platform
Faugeras et al. [35]	7.489	3.6	Correlation	PeRLe-1
Kanade et al. [33]	38.4	30	SSAD	8 DSP
Woodfill and Von Herzen [34]	77.414	42	Census	16 FPGA
Kimura et al. [42]	38.4	20	SSAD	2 PCI boards
Miyajima and Maruyama [36]	491.52	20	SAD	FPGA
Woodfill et al. [37]	297.123	30	Census	ASIC
Forstmann et al. [20]	188.928	12.3	DP	CPU
Point Grey Research Inc. [40]	203.98	83	SAD	CPU
Yang et al. [45]	283.268	11.5	SAD	GPU
Gong and Yang [68]	42.11	23.8	DP	GPU
Wang et al. [48]	52.838	43	SAD	GPU
Yang et al. [27]	19.66	16	BP	GPU
Videre Design [41]	589.824	30	SAD	FPGA
Chang et al. [43]	88.473	50	SAD	DSP
Ernst and Hirschmueller [44]	165.15	4.2	SGM	GPU
Khaleghi et al. [46]	11.5	20	Census	DSP, MCU
Tombari et al. [67]	8.84	5	Efficient aggregation	CPU
Bradski and Kaehler [6]	117.97	66.67	SAD	CPU
Kosov et al. [73]	0.353	2.15	Variational methods	CPU
Zhang et al. [71]	100.859	57	Bitwise fast voting	GPU
Salmen et al. [72]	3.804	5	Optimized DP	CPU

Table 1: Comparison of several stereo vision implementations. Extract from [9]. Mde/s stands for million disparity evaluations per second.

In general the next statements are true for the implementation of all algorithms in one of the following technologies:

FPGAs

+ better performance due to parallelism

+ can be dynamically reconfigured to achieve an optimized task specific architecture

+ can implement any algorithm

+ low power consumption
- difficult programming and reprogramming
- difficult to adapt an existing algorithm to bigger instances of a problem (fixed resources)

DSPs

+ good balance between speed and flexibility

+ low power consumption
+ relatively easy programming (C and assembly)

- limited memory

µProcessors

+ most flexible
+ easiest programming
+ easy to scale the implementation to bigger instances of the problem
+ double precision fp
- need for SIMD instructions to achieve tolerable performance
- low performance
- high power consumption

Chapter 3

The Main Algorithm and its Enhancement

An overview of stereo algorithms

In general, stereo algorithms are divided in two categories, local and global. All stereo vision algorithms perform a subset of these steps:

a) Matching cost computation
b) Cost aggregation
c) Disparity computation/optimization
d) Disparity refinement

Local algorithms depend more on cost aggregation to provide quality results while global algorithms do most of their work in the disparity optimization step to perform a global minimization of a cost function that is defined over the whole image. In essence, local algorithms optimize disparity selection for each pixel independently from other pixels while global methods look to optimize disparity selection for many pixels at once. Local algorithms match pixels in the image pair corresponding to the same point on the scene, by doing for each pixel in the reference image an exhaustive search on a restricted search space in the non-reference image. As disparity optimization is done for each pixel in isolation, local methods allow for a large degree of parallel operations. Our algorithm falls under the local category and thus its high intrinsic parallelism makes it an ideal candidate for custom hardware implementation.

As we have already mentioned, rectification is the pre-processing step that reduces this search space to one dimension, so its size is fully definable by a max disparity parameter, D_{max}. We assume that this step has already been applied. In order to choose the best match, a matching cost is computed for each pixel combination. These matching costs depend only on local information surrounding the pixels in question. The x-axis distance of the two matched pixels (disparity) is directly connected with the actual distance of the object from the camera (to find the absolute distance, camera calibration parameters are required). In general it is a good idea to regard the matching cost as well as the strategy used to select the best match, as interchangeable and independent components of the general algorithmic structure.

Algorithm matching costs

Matching costs can be window-based or pixel-based. Pixel-based costs depend only on the pixel values in question, whereas window-based costs define windows around the pixels and thus also use neighboring pixel values. Several shape and sizes for cost windows have been proposed in the literature (see 16 for an overview), however simple square windows of size W x W are still being widely used. Window-based costs make comparisons between small image blocks and algorithms based on them effectively match spots of the image pair. The obvious strategy to select the best match using those costs is a Winner-Take-All scheme (WTA), where for each reference pixel, the pixel with the lowest cost out of a range of D_{max} candidates, wins. An example of a matching cost is the popular Sum of Absolute Differences (SAD), which can be formally described with the expression:

$$SAD = \sum_{i=n} \sum_{j=m} |I_r(x+i, y+j) - I_l(x+i+d, y+j)|$$

SAD is a window extension of the simple AD measure:

$AD = |I_r(x, y) - I_l(x+d, y)|$, where I_r is the reference image.

SAD is a well-known cost in the field of local stereo vision algorithms and it is widely used. However it has some drawbacks which prevent it from being embedded as a cost in state of the art algorithms. For example it tends to blur object borders. SAD performs W^2 comparisons per pixel evaluation whereas AD, performs only 1.

Another example of a matching cost, is census transform [13], robust to depth discontinuities and different light conditions. As a first step, it transforms the input image and then uses this transformation to produce a match in the classic WTA cost minimization manner. Census transform first calculates a bit-string for each pixel and then uses it to produce a match. It has a cost of $W^2 - 1$ comparisons per pixel.

A pure census transform based stereo algorithm contains two logical tasks. The first task concerns the creation of the bit-string. The second task uses that bit-string to find the best match in a search range of pixels in the non-reference image. For each pixel of the reference frame (assume right):

1. Compute the census bit string for the pixel. The census bit string is defined as a vector of bits, of size $W^2 - 1$, where each bit declares if the intensity of the pixel in the respective position of the window is greater/equal or less from that of the central pixel. For example assume we have the following window:

   127 129 130
   127 125 128
   100 102 103

   The bit-string will be: {1,1,1,1,1,0,0,0}. Similarly we produce the census bit strings of the candidate pixels in the left frame.

2. Compute the hamming distance between the reference and the D_{max} non-reference candidate bit-strings. This hamming distance is referred to as the matching cost. Using a WTA strategy we select the pixel with the minimum hamming distance as our match. For example assume {1,1,1,1,1,0,0,0} the reference bit string and {1,0,1,0,1,0,1,0}, {0,1,1,1,1,0,0,0}, {1,1,0,0,0,0,0,0}, {0,1,1,1,1,0,0,1} the candidate bit strings. The bit string {0,1,1,1,1,0,0,0} has the minimum hamming distance. The depth information is extracted from the x position shift (disparity) between the two matched pixels.

Census transform belongs in the category of non-parametric costs[1] and exhibits resilience to lighting conditions because of its sole dependence on the local image structure. Moreover it outperforms the also non-parametric rank cost (13), as with the bit-string it also encodes the spatial distribution of light. Census can be easily extended to color images but it was shown in [14] to perform only marginally better. However, by ignoring the pixel intensities completely, it looses an important chance of producing more diverse matching costs and thus, less false matches.

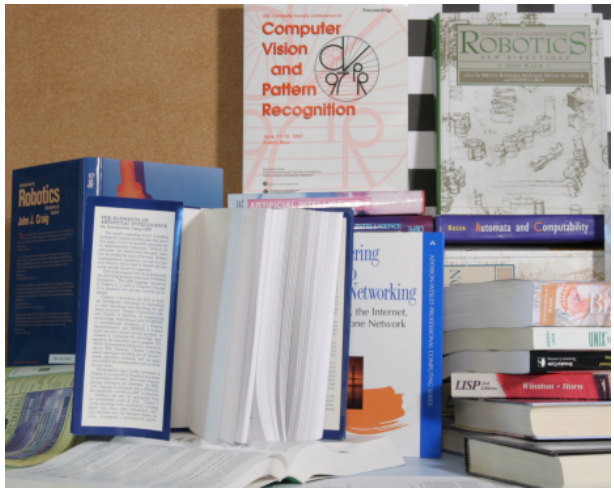
In order to add light intensity information to the census cost, we have decided to combine it with AD and SAD into two new matching costs respectively. SAD-Census and AD-Census are simply the sum of normalized SAD and AD respectively, with census. As we will see, the new costs are more powerful than their parts alone. We have avoided a weighted sum, as it would insert scene-dependent weighting parameters which would require fine tuning.

In order to assess the quality of our system and provide results that can be easily evaluated by the research community we applied our algorithm on well known datasets from the Middleburry database[2]. We selected Art, Books, Dolls, Laundry, Moebius and Reindeer which are shown in Fig 6. All datasets contain rich depth information. As we will see in the best/worst case graphs, some datasets are more difficult to process than others. For example Laundry has a lot of texture-less areas, whereas Dolls has very rich textures.

1 Non-parametric costs are costs that extract information about the pixel, based only on comparisons between neighboring pixels.

2 http://vision.middlebury.edu/stereo/eval/

Art



Books

Dolls

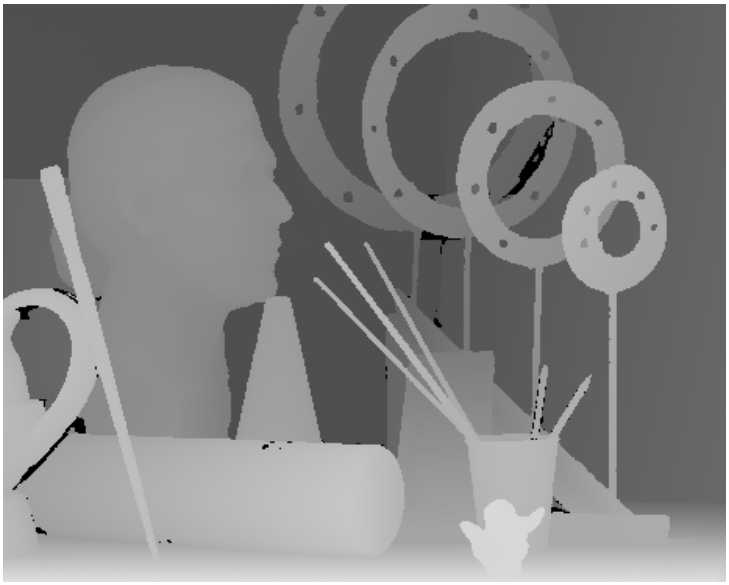
Laundry

Moebius

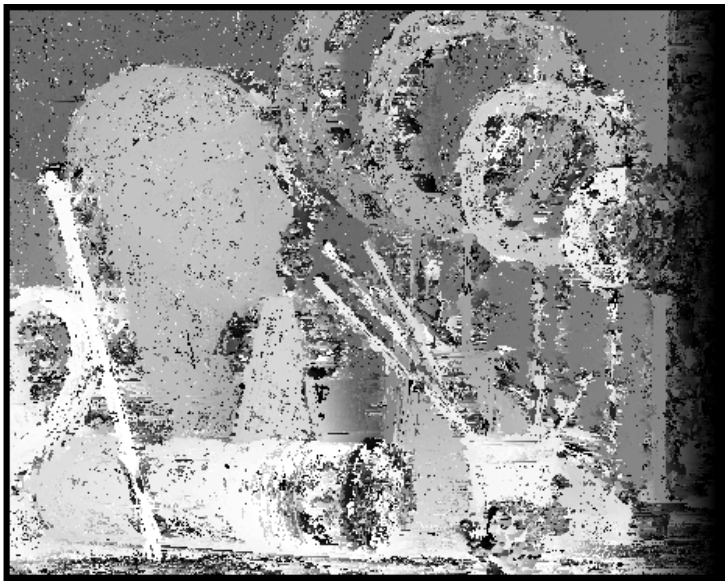
Reindeer

Fig 6: *Datasets used in all our quality comparisons.*

All results presented, are produced from a software MATLAB implementation running on a Core 2 Duo E6400 2.1 Ghz processor.



Ground truth



Census 9×9

AD-Census 9×9

SAD-Census 9×9

Fig 7: *Visual comparison of pure census vs SAD-Census vs AD-Census.*

Above (Fig 7), we compare visually Census, SAD-Census and AD-Census on the same scene. SAD-Census has less false-positives than either of them. More thorough comparisons are presented in Fig 8 and Fig 9.

We have chosen the percentage of good matches as our quality metric. Results where calculated for all 6 test image pairs and we present best/worst case and mean value comparisons. The quality metric was calculated on regions where our algorithm works optimally: a small frame of (W-1)/2 where we cannot fill in the census windows, as well as D_{max} pixels from the right side, where we must reduce the disparity search range, are ignored. SAD-Census may appear better by this metric but due to its low-pass filter nature, it has a tendency to blur edges. Performance was best with Census and worst with SAD-Census, as was expected (Fig 9a).

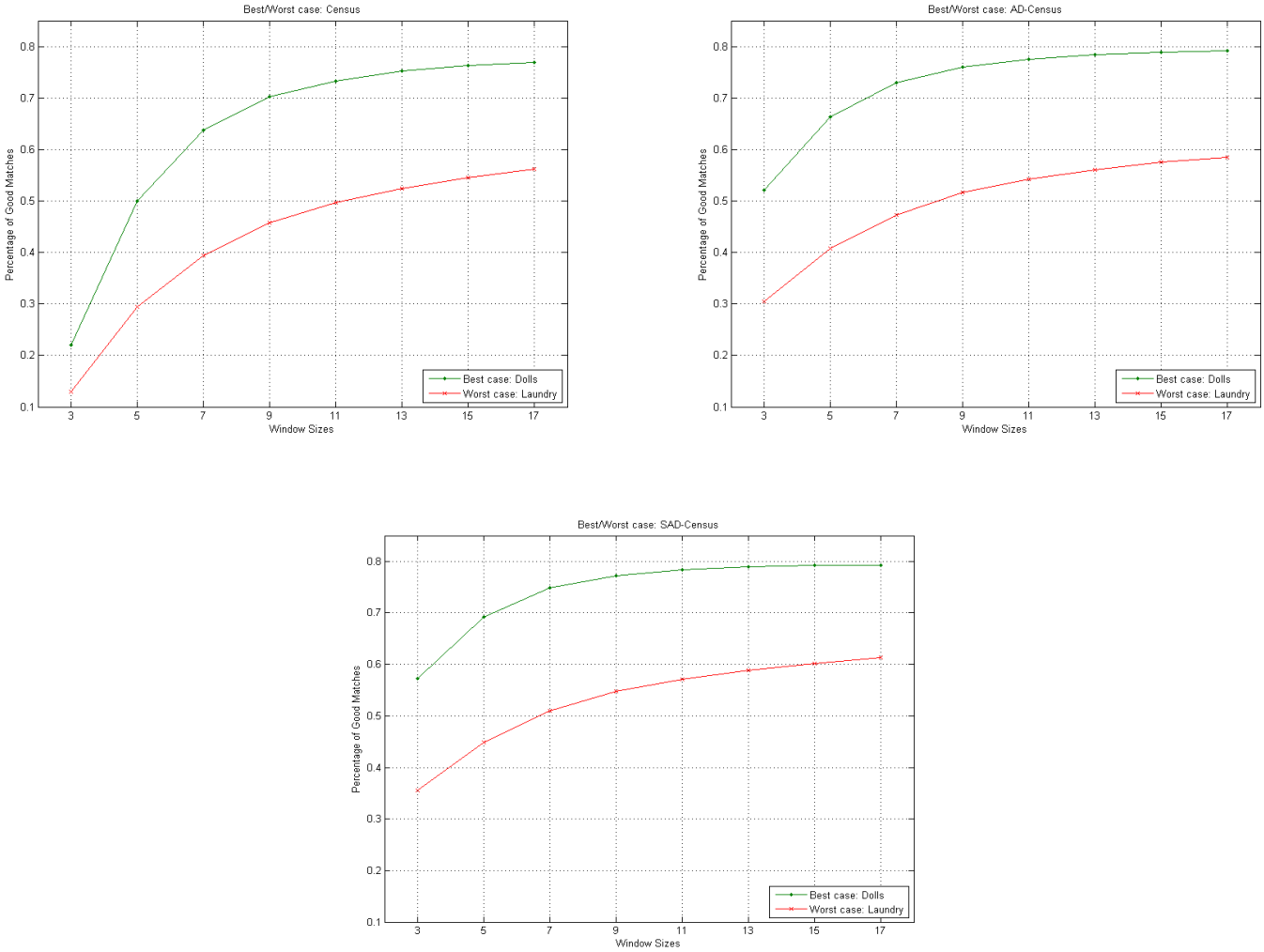
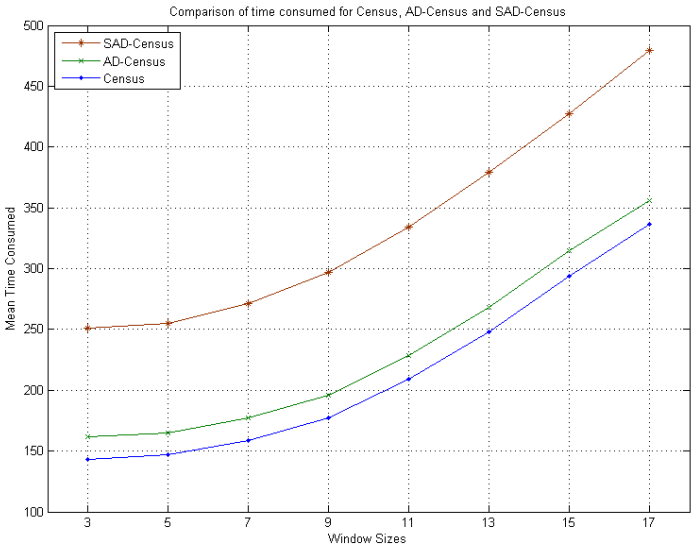
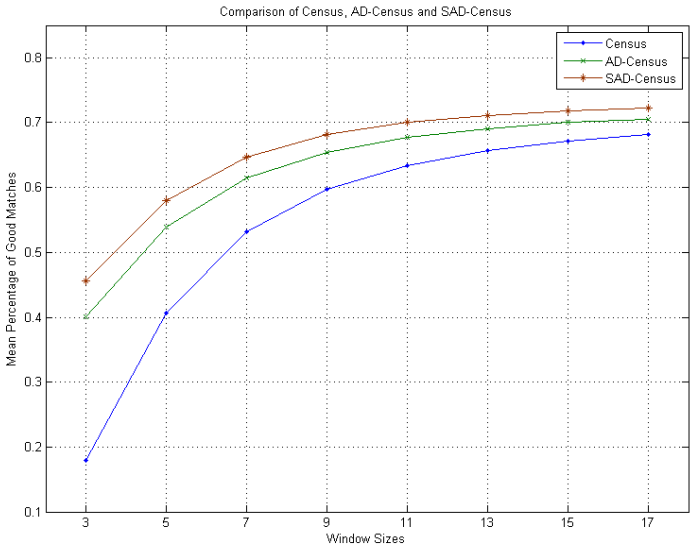


Fig 8: *Best/Worst case quality comparison using the percentage of good matches metric, for Census, AD-Census and SAD-Census. Window size ranges from 3×3 to 17×17. The Dolls dataset gives the best results in all three costs and Laundry dataset gives the worst. The reason for this is easily understood by observing the datasets: Dolls provides much greater variation, whereas Laundry has many uniform areas.*



a) Time taken (seconds)



b) Percentage of good matches

Fig 9: *Quality comparison of Census, AD-Census and SAD-Census using mean values over all the datasets. SAD-Census is better by the percentage of good matches metric. Window size ranges from 3×3 to 17×17. Performance refers to a single-threaded unoptimized implementation and is shown to demonstrate how window size increases algorithmic complexity.*

The new matching costs improve the results significantly as can be seen in Fig 9b, especially for small window sizes. However, we can still employ some simple processes for further gains.

The window aggregation of the matching costs at each disparity level is equivalent to applying a box filter on the (x,y) dimensions of the DSI[1] image ([15]). It smooths out the costs, based on the assumption that neighboring pixels have similar disparities (box filter weights each window pixel the same), and allows for a much less noisy disparity map (Fig 10). However, as this assumption is false on disparity discontinuities, it has a negative impact on object borders, producing an edge thickening effect. Box filter also implicitly assumes frontal-parallel surfaces which is often violated in practice with slanted surfaces. Use of edge preserving smoothing filters such as bilateral or guided help alleviate these problems ([15]) as they place more weight on window pixels that are similar in color to the central one and thus are more probable to belong to the same depth. Such filters, however, are computationally expensive. Even with its downsides, box filtering yields an impressive 16% improvement, augmenting our 65% mean percentage of good matches to 77%, when transitioning from W=9 W_a=1 (no aggregation) to W=9, W_a=5. By using a simple box filter, we have to compromise with the fact that the size of the aggregation window W_a is a tradeoff between border accuracy and better matches elsewhere.

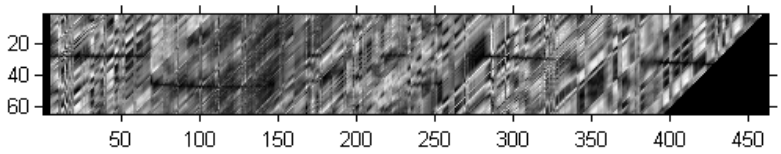
Following on our discussion about edge aware filters, we created a cost specifically designed to guide the aggregation process. The cost is defined as follows:

$$C_p = \begin{matrix} 1 & when \ |p_c - p_i| < \tau \\ 0 & otherwise \end{matrix} \quad with \ i \in W_a$$

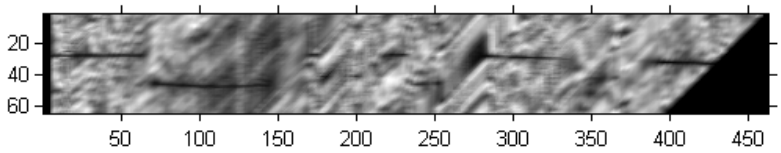
where W_a is the aggregation window, p_c the central value of this window and τ a threshold given as a parameter. Simply put, the resulting matrix shows the similarity, in terms of intensity, of the neighboring pixels to the central pixel in a 1/0 format. This matrix can act as a mask, selecting which values get aggregated in the window. The logic behind this cost is the simple but effective assumption that pixels with similar intensities probably also lie on the same disparity. More edge aware smoothing can be seen in Fig 10.

1 DSI (or Disparity Search Image), is a 3D volume of size WxHxD_{max} containing the matching costs for all D_{max} candidates of any pixel (x,y) in the WxH frame.

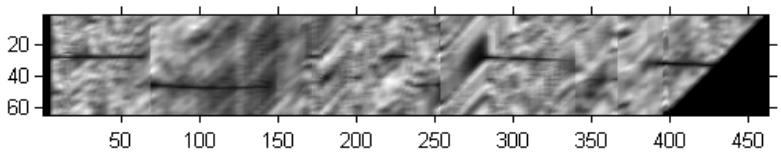
Reference image



Census without aggregation



Census with 5×5 aggregation



Census with 5×5 guided aggregation

Fig 10: *DSI(x,50,d) slices for aggregated and not aggregated versions. Matches appear as black lines along the horizontal axis. Aggregation smooths out many erroneous matches and produces a cleaner DSI. The effects of guided aggregation are best seen on disparity discontinuities.*

Optimization in regards to τ across all datasets led to a value of 60 (given that intensity values are in the range [0-255]). We also have to take into account that a very small τ value could lead to weak aggregation which results in an increase in errors. Guided aggregation led to a 0,29% improvement on the average percentage of good matches for W=9 and W_a=3, 0,35% for W=9 and W_a=5 and 0,49% for W=9 and W_a=9.

Another improvement, is the use of sparse instead of dense windows, as was shown in Humenberger et al. [9]. A way to produce such windows is to reduce the sampling rate from 1 (every pixel is sampled) to 0,5 (one every two pixels is sampled), both in x and y axis. Fig 19 shows the positive effect of this process on the quality of the results. It is important to note that equivalent window sizes were used for our quality comparisons with sparse and dense windows: the windows used the same number of pixels.

In order to get rid of false matches due to occlusions, we can perform a Left/Right consistency check (LRC), which allows only disparities that are validated by reversing the reference image. In other words, LRC performs the correlation step with the opposite

image as reference and checks if the new matches found are in accordance with the previous ones (condition $|Disp_{RL}(x,y)-Disp_{LR}(Disp_{RL}(x,y),y)| <$ LRC Threshold must be true for some small positive value of LRC Threshold). Being a post-processing step, it is independent of the matching cost used.

We have implemented a simple scan-line belief propagation solution that fills in the occluded pixels detected by LRC. Fig 14 shows the flowchart of our belief propagation algorithm. It works by replacing occluded pixels with the most confident local disparity along the processing direction on the scan-line. It has the advantages of simplicity, low memory footprint and online calculation as it doesn't use windows, processing pixels in a sequential manner instead, in the order given by the LRC step. This procedure fits well with LRC and doesn't disrupt our workflow at all. Furthermore it has good results as can be shown in Fig 24.

Using the mean intensity of the window instead of the central pixel for the comparisons in the census window gives a possible increase in noise robustness as we no longer depend on a single pixel which can end up being a statistical outlier. Testing however this method revealed that it led to strong edge fattening which ended up hurting the overall quality. Furthermore we experimented with a median solution which performed better than mean, but in the presence of aggregation was outmatched by the simple central pixel method, as can be seen in the comparison Fig 12.

We have also tried a multilevel census solution, where instead of a simple 1/0 comparison between the central pixel and its window, we have defined multiple levels of relation (Fig 13). Hamming distance was redefined as the absolute distance between the new census vectors. This tweak produced small quality improvements for its relatively high cost.

We also explored the possibility of improvement by weighting differently the Census or AD part of the AD-Census matching cost. Fig 11 shows that results were best for the default 50-50 balance.

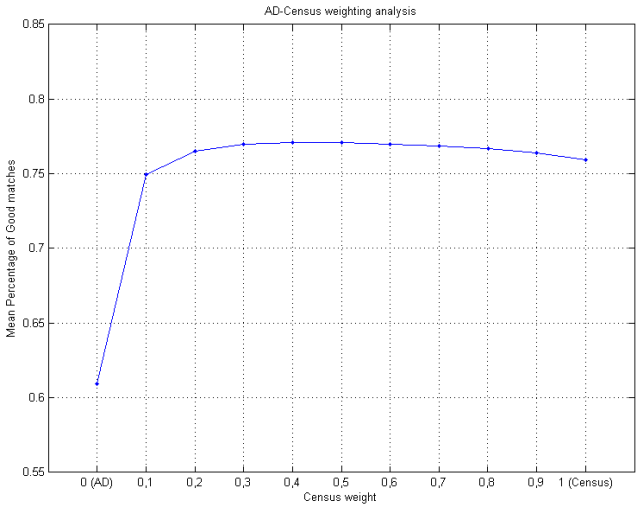
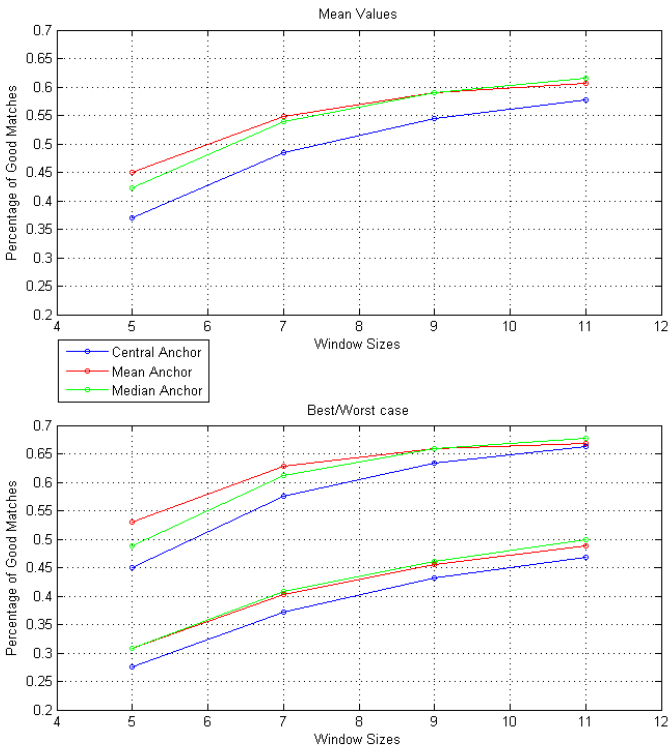
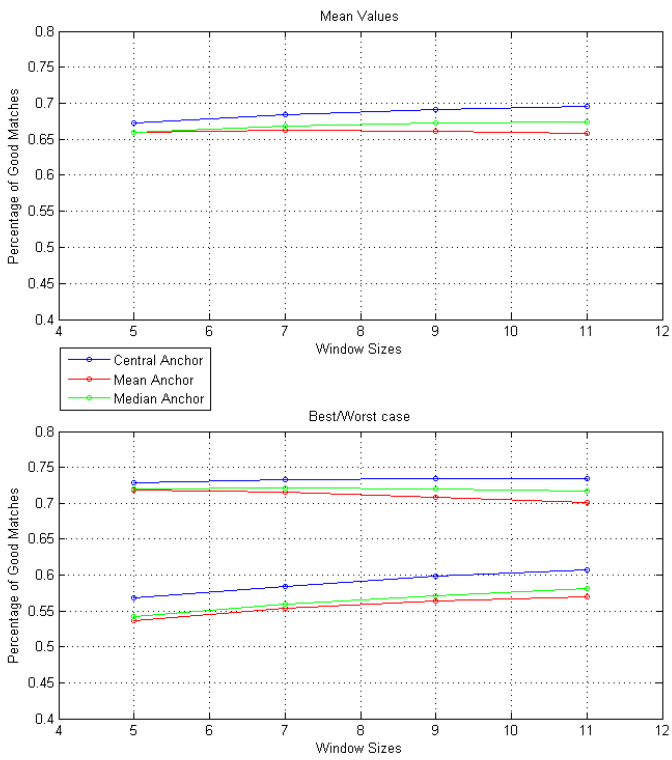


Fig 11: *AD-Census analysis for different weigthing values. X-axis shows the weight of Census,* $c \in [0,1]$ *. AD weight is defined as 1-c.*



Without aggregation



With aggregation (W_a=5)

Fig 12: *Quality comparison of different census methods with and without aggregation.*

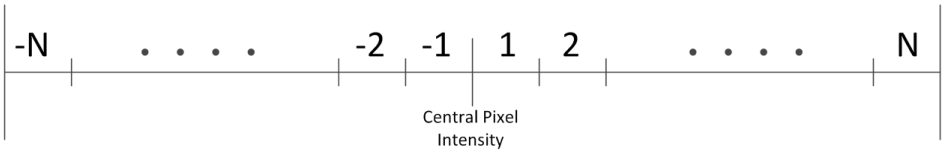


Fig 13: *An example of 2*N census levels.*

A way to quantify the confidence of a match, is to compare the minimum cost with the runner up. If their difference is small, there is a good chance that the match is a false positive. A formula to calculate the confidence of a match is the following:

$$C = min(255, 1024 * ((min_2 - min_1)/MaxCost)) \quad [9]$$

Another simple formula is the runner-up costs difference:

$$C = min_1 - min_2$$

In both cases, a threshold can be applied to cut off uncertain matches detected.

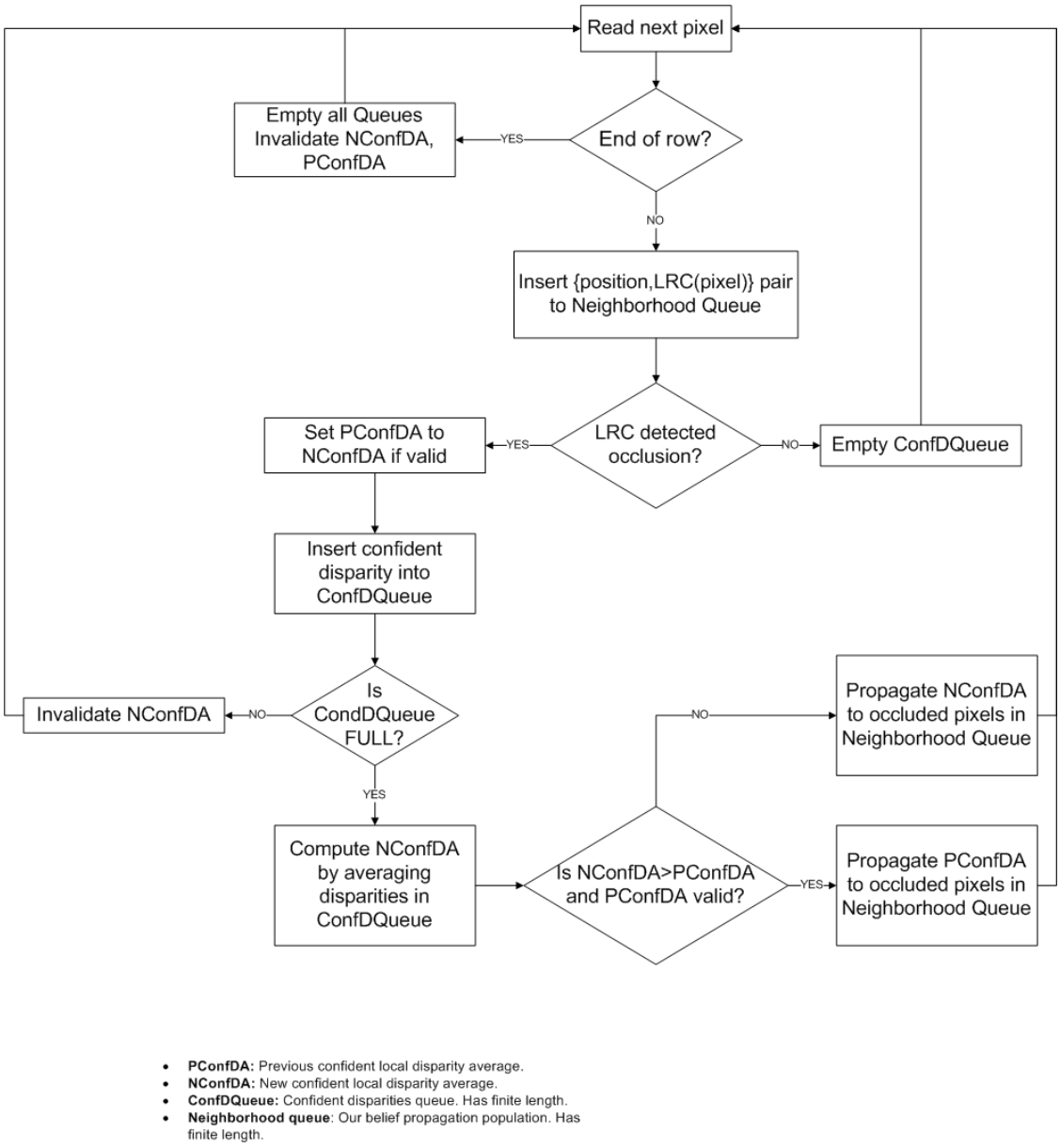


Fig 14: *Flowchart of our belief propagation algorithm. If we encounter foreground pixels when moving on the background (NConfDA>PConfDA), we propagate the background confident disparity (PconfDA). The opposite happens when we encounter background pixels whilst on the foreground (PconfDA>=NConfDA).*

Local methods are especially vulnerable to areas with a lack of texture, producing a lot of false matches. Local variance is a way to find these areas and take them into account when computing the confidence of a match.

Objects far away from the cameras, tend to produce disparities that are fractions of full disparity levels. Sub-pixel accuracy reduces the quantization errors by approximating fraction disparities through interpolation.

Our implementation will focus on aggregating costs and sparse windows as all those other processes can be added in a modular way to the algorithm.

Parameter tuning

The main matching procedure of our algorithm has three parameters that require tuning: The maximum disparity search range (D_{max}), the window size (W) and the aggregation window size (W_a). We also have to decide whether to use a sparse window solution or not. LRC depends on a threshold value to decide whether a pixel is occluded or not. Our belief propagation demands two additional parameters, neighborhood queue size and confident disparities queue size. Finally we need to select one of the three matching costs presented above.

As can be shown in Fig 16 and Fig 9, quality doesn't scale linearly with W. A compromise was settled on W = 9.

The aggregation window size improves the percentage of good matches but large windows suffer from inaccuracy at object borders, as shown in Fig 15. As can be seen in Fig 16, aggregation window size can be regarded equivalent to window size in terms of effect on our quality metric. We have chosen, however, to set a reasonable value of 5×5 so that we won't sustain serious degradation in object borders quality. A visual comparison between Census, AD-Census and SAD-Census with W = 9 and W_a = 5 can be seen in Fig 20.

Fig 19 shows the effect of different window sizes on the percentage of good matches for dense and sparse windows, for no aggregation and with a 5×5 aggregation. In both cases the biggest difference in quality exists for smaller window sizes. When aggregation was applied, the difference in quality is negligible and thus not justified cost-wise for the window size we are considering (9×9), so we have decided to use normal (dense) windows.

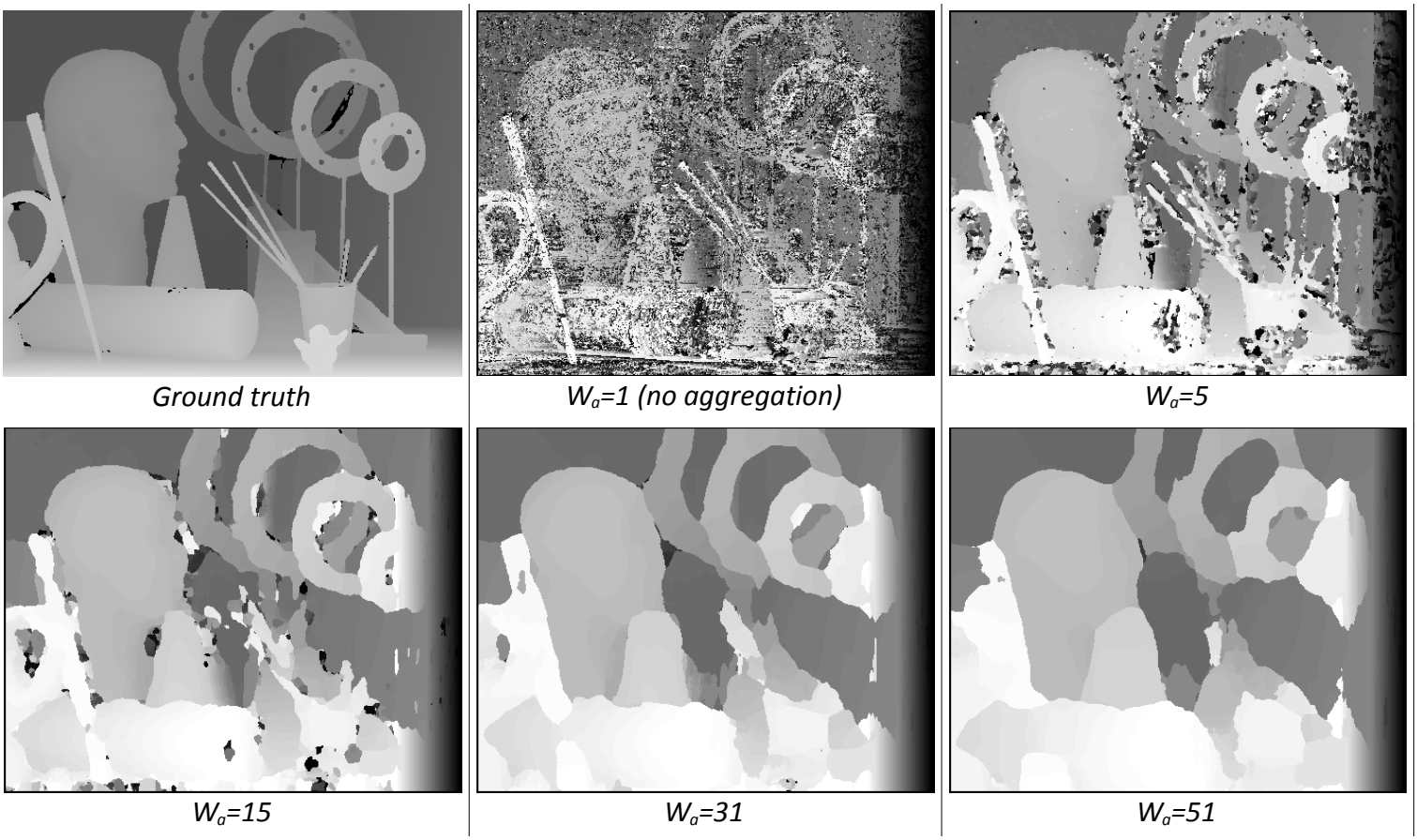


Fig 15: *The blur effect of increasing aggregation window sizes with a 5×5 census.*

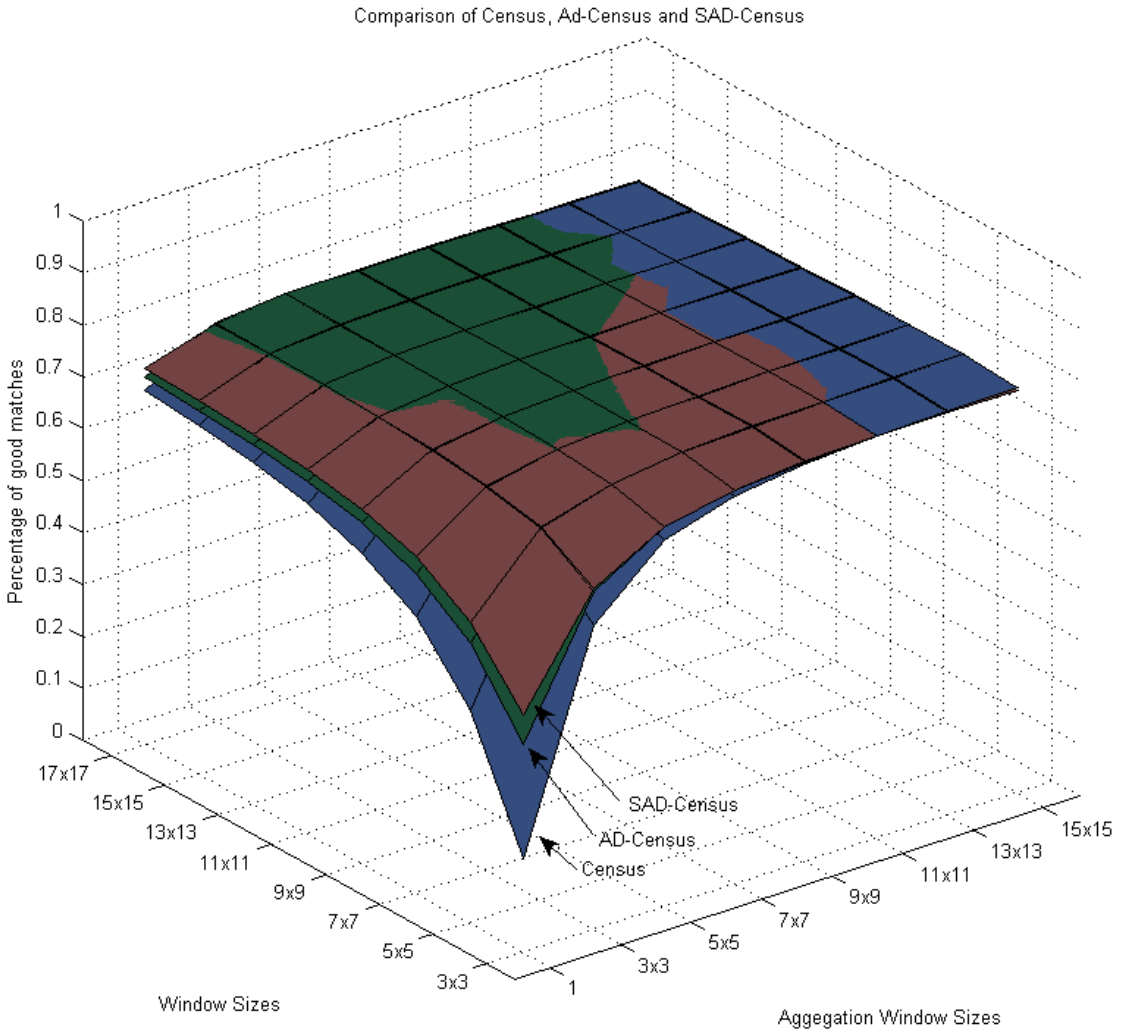


***Fig 16**: Quality comparison of Census, AD-Census and SAD-Census. When no aggregation is applied (W_a=1), the latter is better, but has significant computational cost, as it performs approximately 2xW^2 operations. Notice how AD-Census outperforms Census and SAD-Census in the region of* $W \in [7,17] \wedge W_a \in [3,13]$

Camera spacing (baseline, b) as well as camera focal length f play an important role on the range of depth values that can be calculated. More spaced out cameras allows us to detect points farther in expense of closer areas. More specifically, we can calculate minimum and maximum detectable depth by the formulas:

$Depth_{min}=\frac{b*f}{D_{max}*DP_{hor}}$ and $Depth_{max}=\frac{b*f}{D_{min}*DP_{hor}}$, where DP_{hor} is the pixel horizontal dot pitch, the horizontal distance between two neighboring pixels and depends also on the resolution configuration of the camera (active pixel size). For example a camera such as Aptina MT9D112 (1600×1200@15fps), with an example b=63mm, f=3.79mm, Dp_{hor}≈0.0022mm (for its maximum resolution of 1600×1200) and $D\in[0,63]$ gives $Depth_{min}$=1.723m and $Depth_{max}$=108.532m.

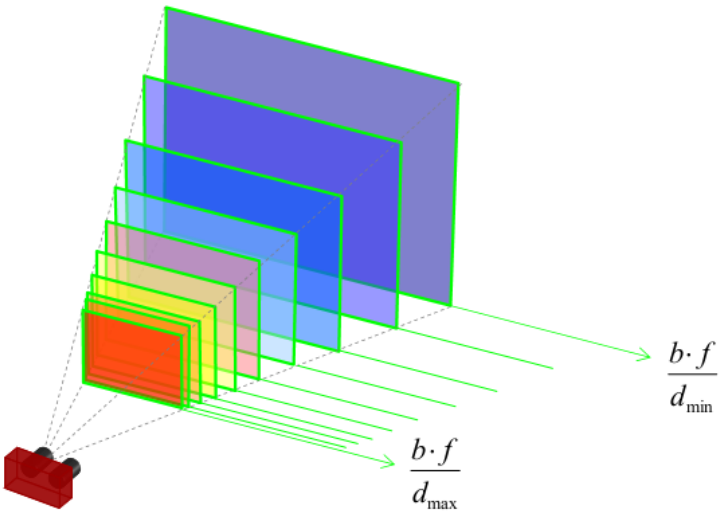
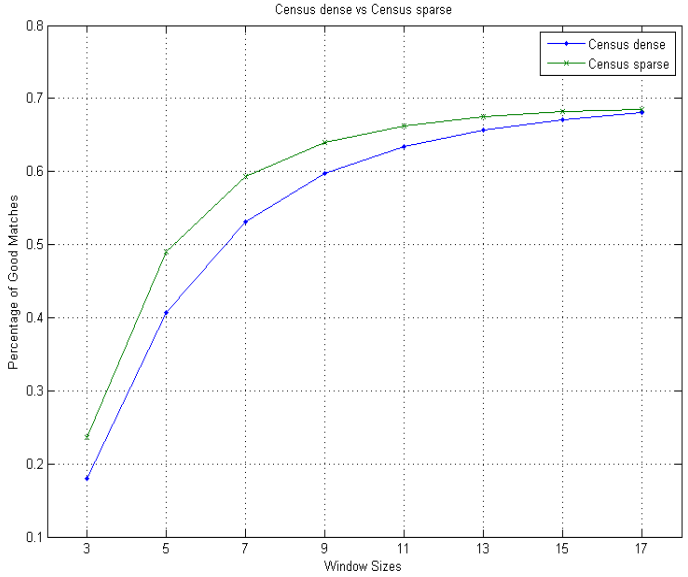


Fig 17: *Discrete disparity space, illustrating the stereo system's range capabilities. Image courtesy of Stefano Mattoccia.*

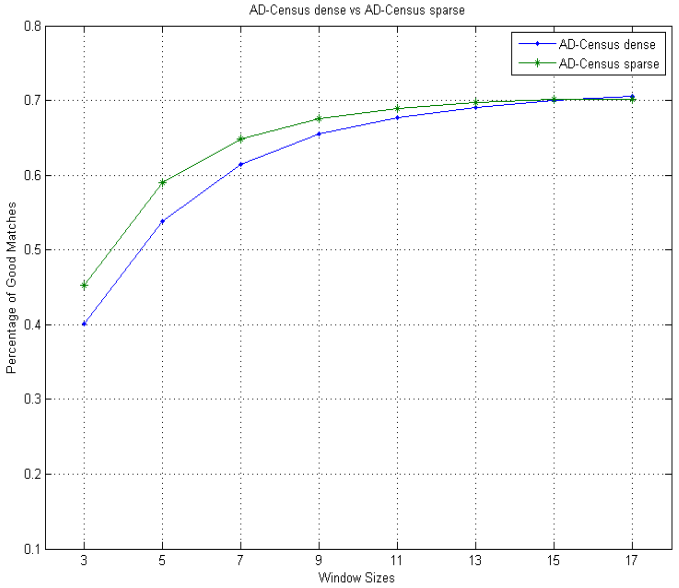
All parameters are influenced by the frame size (resolution) and especially D_{max}, which depends heavily on the frame width. This happens because features in the image become more spaced out with bigger frames. Fig 18 shows the approximate value that D_{max} should be set to, in order for the results to not degrade in quality. Inter-camera distance also plays an important role on D_{max}, as more spaced cameras magnify all disparities and thus require larger D_{max} if we want to keep our minimum range detectability intact. Frame size also enlarges texture-less regions where census performs poorly and would require window enlargement to compensate.

Resolutions	100×83	384×320	640×533	1024×853	1600×1333
D_{max}	10	40	64	128	160

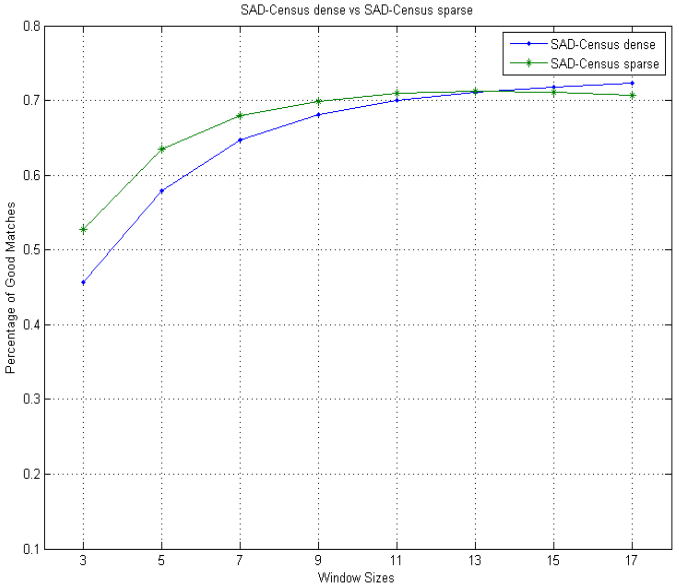
Fig 18: *A rule of thumb would be to set D_{max} to approximately 1/10 of the frame width.*



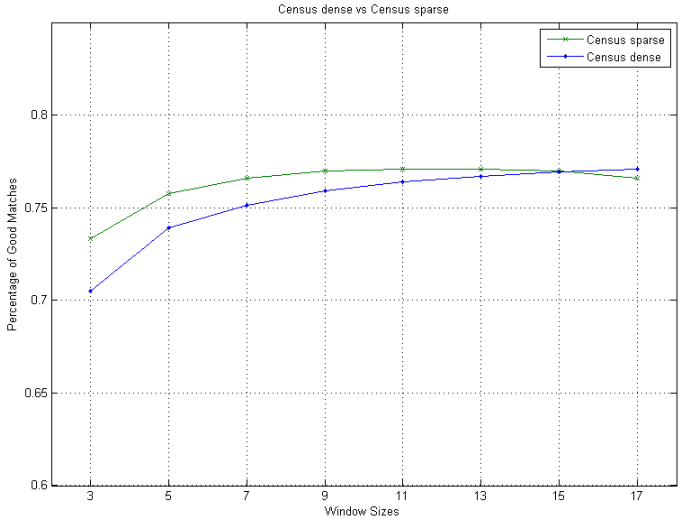
Census, no aggregation



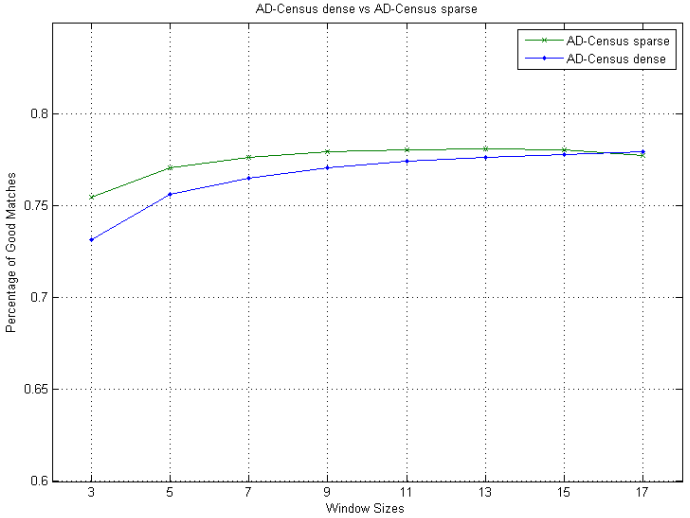
AD-Census, no aggregation



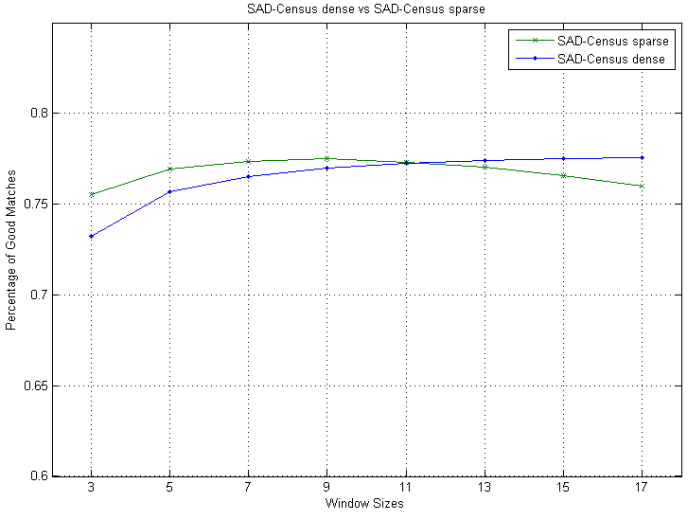
SAD-Census, no aggregation



Census, 5×5 aggregation

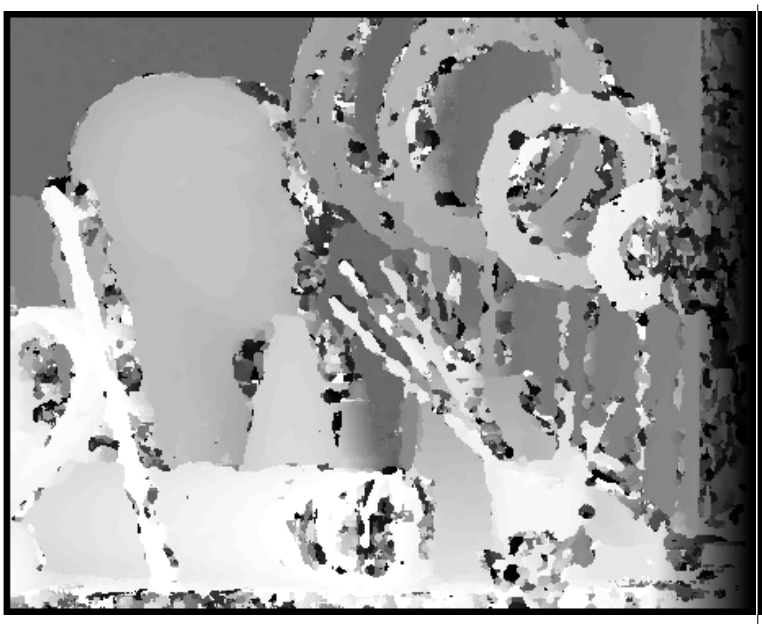


AD-Census, 5×5 aggregation



SAD-Census, 5×5 aggregation

***Fig 19**: Sparse vs normal windows using mean values. Note that equivalent window sizes were used in order to compare both approaches on the same terms: A normal WxW window corresponds to a (2xW-1)x(2xW-1) sparse window, both containing the same number of pixels. On the first row no aggregation is used and y-axis ranges from 0.1 to 0.8. On the second row, a 5×5 aggregation is used and the y-axis ranges from 0.6 to 0.85.*



Census 9×9, W_a=5

AD-Census 9×9, W_a=5

SAD-Census 9×9, W_a=5

***Fig 20**: Visual comparison of Census, AD-Census and SAD-Census with W=9 and W_a=5.*

Left/Right consistency check is demonstrated in Fig 21 for several values of the LRC threshold parameter. A high threshold tends to allow erroneous disparities through, while a low threshold discards too many pixels. A value of 4 was selected as a good compromise as we have to keep in mind that LRC will also determine which pixels will be processed by our belief propagation algorithm.

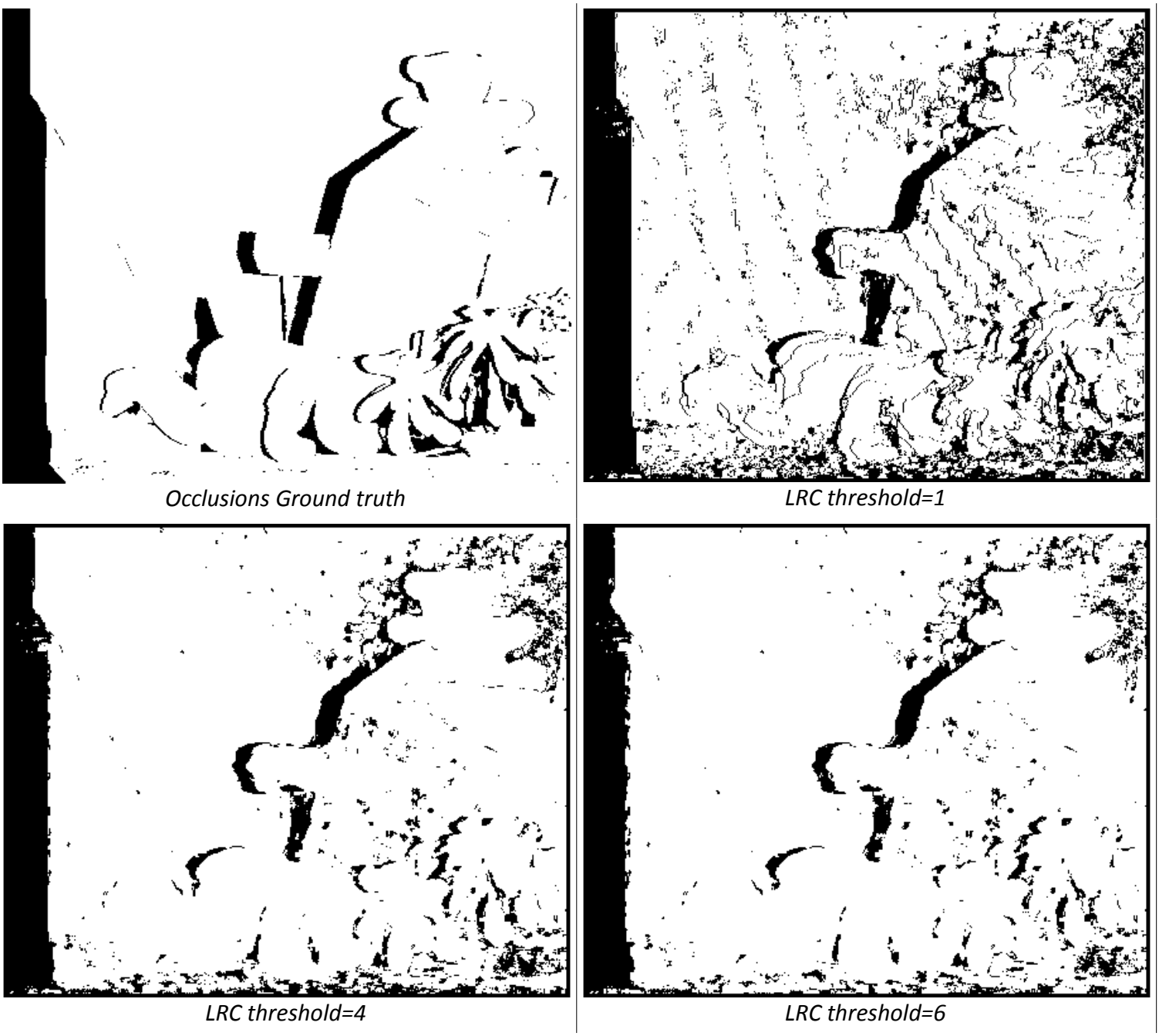


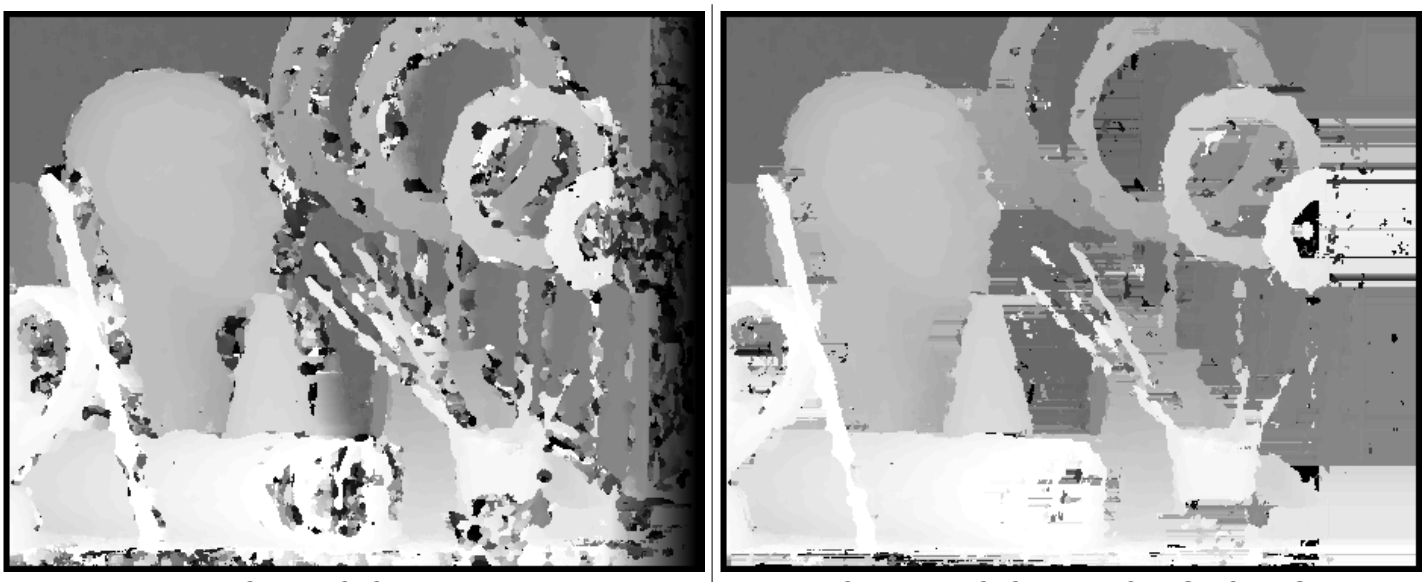
Fig 21: *Left/Right Consistency check for several values of the LRC threshold.*

Our belief propagation algorithm defines two additional parameters: Neighborhood queue size (S_{NQ}) and Confident Disparities Queue Size (S_{CDQ}). We will now discuss how these parameters alter the results.

S_{NQ} sets the size of our neighborhood, which is the size of our "population" of pixels where the local confident disparity will get propagated. Large values for this parameter allow the propagation of the local disparity average to more pixels and its positive results can be observed on regions with "thick" occlusions. For example, notice the black bar on right side of the occlusion maps in Fig 21: If S_{NQ} is larger than its width W, then the local confident disparity will get successfully propagated to all of it, eliminating all unknown disparities on that region. If, on the other hand, it's smaller, then a bar of width W- S_{NQ} of

unknown (black) pixels will remain on the left border. A large S_{NQ} has only positive effects on the quality as our algorithm follows an aggressive propagation policy by instantly propagating confident disparities only to unknown pixels in the neighborhood queue. Setting this parameter to around our maximum disparity should be enough.

S_{CDQ} however, can have both positive and negative effects. S_{CDQ} determines how many confident disparities are needed to initiate propagation as well as the size of the confident disparities population which we use to calculate the local confident disparity average (ConfDA). The fact that we empty the Confident Disparities Queue every time we stumble upon an occluded pixel, makes it difficult to initiate propagation in the first place. Thus it is clear, that by setting this parameter high, we will have a problem on areas where the occlusion map is noisy (e.g top right area on Fig 21) or with objects that appear thin on our disparity map. On the other hand, setting this parameter low, subtracts from the confidence of the ConfDA and initiates propagation too often, which appears as noise on the propagation areas. Fig 23 shows the quality of BP for several values of S_{CDQ}. A value of 4 appears to be best. Fig 22 contains a visual comparison of census vs census with LRC and belief propagation and Fig 24 presents a more thorough quality comparison based on the quality metric of percentage of good matches for all the matching costs. As belief propagation is a post-processing step, results are nearly identical between the different matching costs tested, with the slight variations explained by the small differences on the quality of the matching costs.



Census 9×9, W_a=5 | *Census BP 9×9, W_a=5, S_{NQ}=64, S_{CDQ}=6*

Fig 22: *Visual comparison of pure census vs census with the proposed belief propagation procedure. Notice how BP fills in previously occluded pixels on object borders and on the right side of the image.*

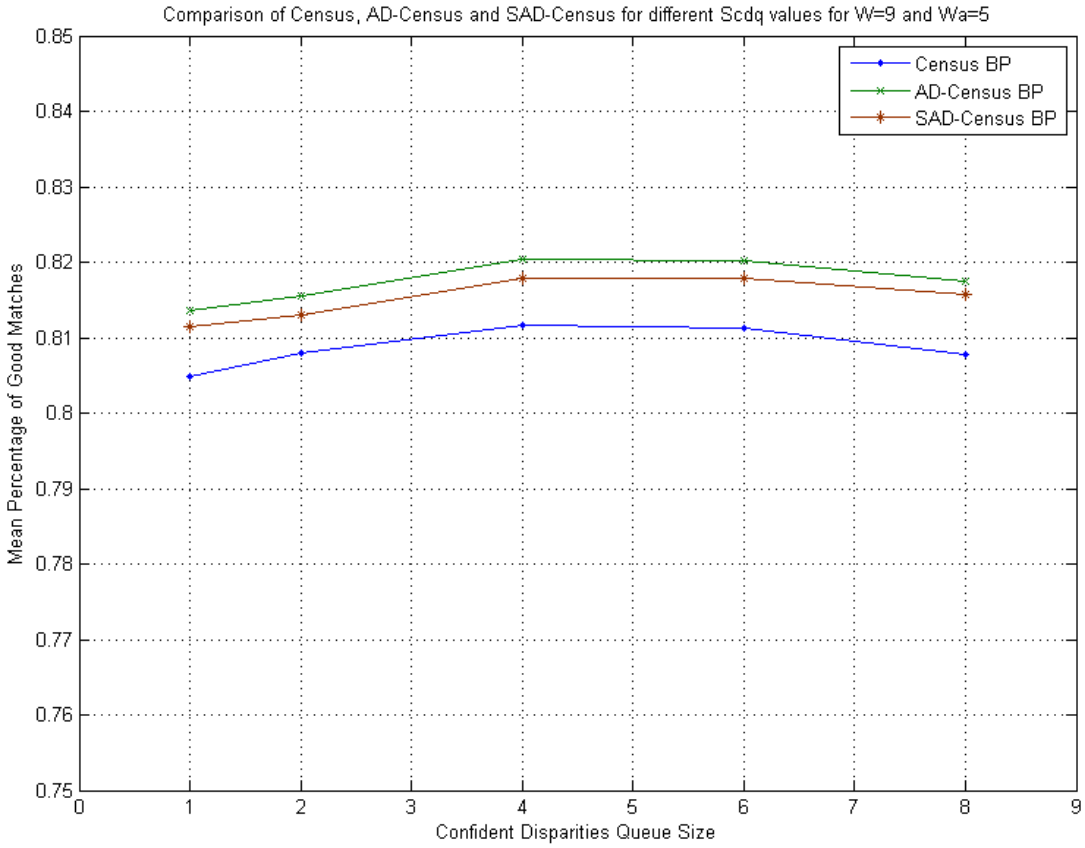


Fig 23: *Mean percentage of good matches metric for several values of the confident disparities queue size parameter. Census, SAD-Census and AD-Census have minimal differences.*

$W=9, W_a=5, S_{NQ}=64, S_{CDQ}=4$	Without BP	With BP
Census	75,93 %	81,16 %
AD-Census	77,09 %	82,03 %
SAD-Census	77,00 %	81,78 %

Fig 24: *Mean percentage of good matches for best configurations of all three matching costs, aggregated on all six Middleburry datasets.*

The maximum disparity search range (D_{max}) was set to 64, as this is a common setting for many current solutions. A low disparity setting limits the minimum detectable distance while a large one increases considerably the computational load.

Based on all the above data, we have made the decision to use an AD-Census matching cost, as it appears to perform best with little additional computational cost in regards to a pure census approach.

Chapter 4
Implementation

Design overview, parallelism and optimization

The algorithm offers parallelism in many levels. If we assume no structural hazards to access pixels from a frame buffer, we can calculate simultaneously any matching cost of any pixel of any frame. The algorithm offers pixel granularity parallelism. However, such a scheme would require a prohibitive amount of resources and would introduce very long delays due to huge gate fan-ins. On the other hand, such degree of freedom, allows us to construct the system architecture in any way we see fit. The only constraints come from the format the camera uses to deliver the data and the amount of resources we are willing to allocate.

Given the sequential manner in which we receive pixels from the cameras, the most efficient approach to process the data would be in a streaming fashion, which translates to D_{max} disparity evaluations per pixel clock. This constraint can be satisfied either by performing these evaluations in parallel or by operating the core on a multiple of the pixel clock and buffering the intermediate results.

It is important to assess the need for flexibility -in regards to algorithm parameters- and the gains of such a setup. First and foremost, we will build a system that is frame-agnostic. In other words our system will support a series of frame sizes within a range of choices. We regard this feature as obligatory. However, a limit on the maximum frame width was imposed, for reasons that we will explain in the Resources Analysis section. In addition, all the algorithmic parameters discussed in the Parameter Tuning section are adjustable. We also chose to structure our system in a modular way, in order to easily add/remove features. Features such as scanline belief propagation and aggregation can be turned on or off by the user of our system.

The effects of different window sizes and different aggregation window sizes can be seen in Fig 16 or in Fig 25 in more detail. There are negligible gains if we choose a larger W or W_a. The maximum achievable percentage of good matches was 78,36% for AD-Census (W=7, W_a=13), so there is no real benefit in choosing such a large aggregation window. It is thus our choice to fixate the window sizes on our implementation. Our design remains generic in any parameter aspect but it is not reconfigurable during run-time. This decision simplifies our hardware design. Our system was verified on a Virtex 5 XC5VLX110T as well as a Spartan 3 1000, setting the parameters accordingly to fit the platform at hand.

Constant W_a	W=7, W_a=5	**W=9, W_a=5**	W=11, W_a=5
AD-Census	76,50%	**77,09%**	77,47%
Constant W	W=9, W_a=3	**W=9, W_a=5**	W=9, W_a=7
AD-Census	75,02%	**77,09%**	77,89%

Fig 25

Our design's scope is the stereo vision algorithm, not the Input/Output process. We have implemented a very simple IO solution based on RS232, for verification purposes only.

Block Diagrams

The general block diagram of the main stage datapath of our algorithm (AD-Census cost initialization and aggregation) is shown in Fig 1.

As input, the system receives two 8-bit pixel values per clock period, each for the corresponding image in the stereo pair. A window buffer is constructed for each data flow in two steps. *Lines Buffer* stores W-1 scanlines of the image, each in a BRAM, conceptually transforming the single pixel input of our system to a W sized column vector. *Window Buffer* acts as a W sized buffer for this vector, essentially turning it into a W^2 matrix.

This matrix is subsequently fed into *Census Bitstring Generator*, which performs W^2-1 comparisons per clock, producing the census bit-string. *Central pixels/Bitstrings FIFO* stores 64 non-reference census bit-strings and window central pixels, which, along with the reference bit-string and central pixel are driven to 64 *Compute Cost* modules. This component performs the xor/summing that is required to produce the hamming distance for the census part of the cost, along with the absolute difference for the AD part and the necessary normalization and addition of the two. The maximum census cost is 80 as there are 81 pixels in the window. Likewise, the maximum AD cost is 255 as each pixel is 8 bits wide. As the two have different ranges, we scale the census part from its 0-80 range to a 0-255 range turning into an 8-bit value. To produce the final AD-Census cost we add the two parts together, resulting in a 9-bit cost to account for overflow. Truncating this cost to 6-bits produces a slight improvement in quality as well as reduced buffering requirements in the aggregation step, as discussed in the Resources Analysis section.

For the aggregation task, 22 line buffers (*Aggregation Lines Buffer*) are used for 64 streams of 6-bit costs, each lines buffer allocated to 3 streams. Like the *Lines Buffers* at the input, they conceptually transform the stream of data to W_a sized vertical vectors. Each vector is summed separately in the *Vertical Sum* components and driven to delay adders (*Horizontal Sum*), which output X(t) + X(t-1) + … + X(t-4). At the end of this procedure we have 64 aggregated costs.

Main stage Datapath

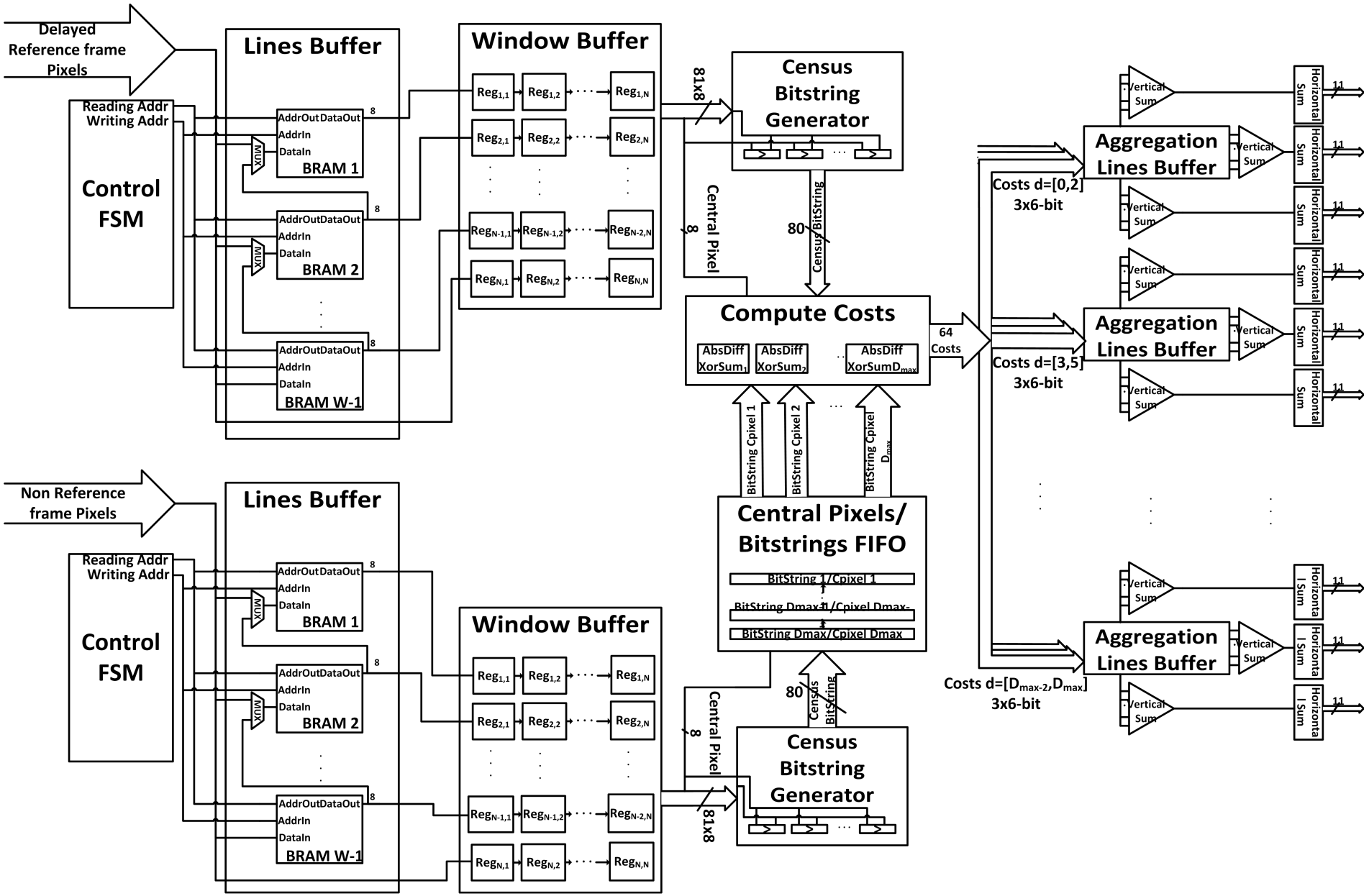


Fig 26

Left/Right Consistency check
Datapath

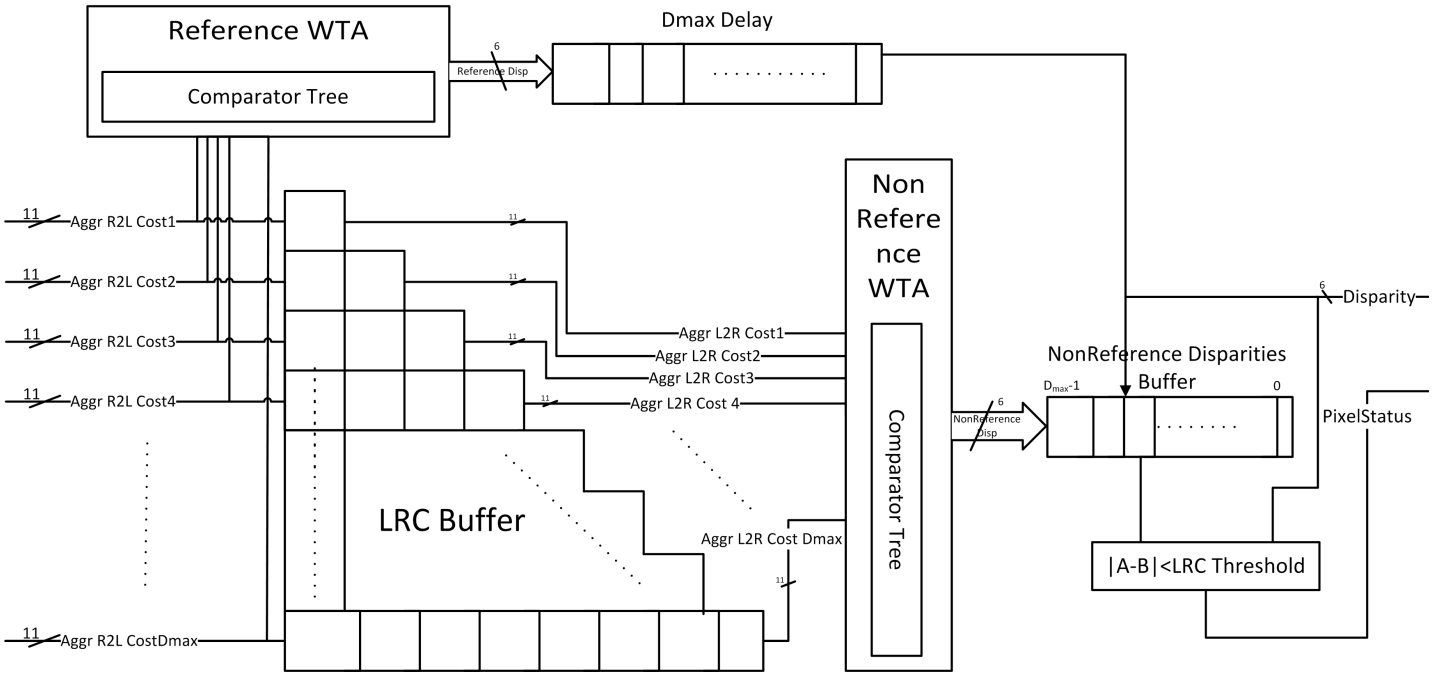


Fig 27

Following the aggregation of costs, the LRC component (Fig 27) filters out mismatches caused by occlusions or otherwise. The architecture of this component is based on the observation that by computing the right-to-left disparity at reference pixel p(x,y) (used costs between pixels (p(x,y),p'(x,y)) , (p(x,y),p'(x+1,y)) , … , (p(x,y),p'(x+D_{max}-1))), we have already computed the costs needed to extract the left-to-right disparity at non-reference pixel p'(x,y) (needs costs (p(x,y),p'(x,y)) , (p(x-1,y),p'(x,y)) , … , (p(x-D_{max}+1,y),p'(x,y))) (see Fig 29). *LRC Buffer* is a delay in the form of a ladder that outputs the appropriate left-to-right costs needed to extract the non-reference disparity. The *WTA* modules select the match with the best (lowest) cost using comparator trees. The reference disparity is delayed in order to allow enough time for the non-reference disparities space to build up in *NonReference Disparities Buffer* and then it is used to index said buffer. Finally the thresholded absolute difference of $Disp_{RL}$(x,y) with $Disp_{LR}$(x,y) indicates the false matches detected.

Scanline Belief Propagation Datapath

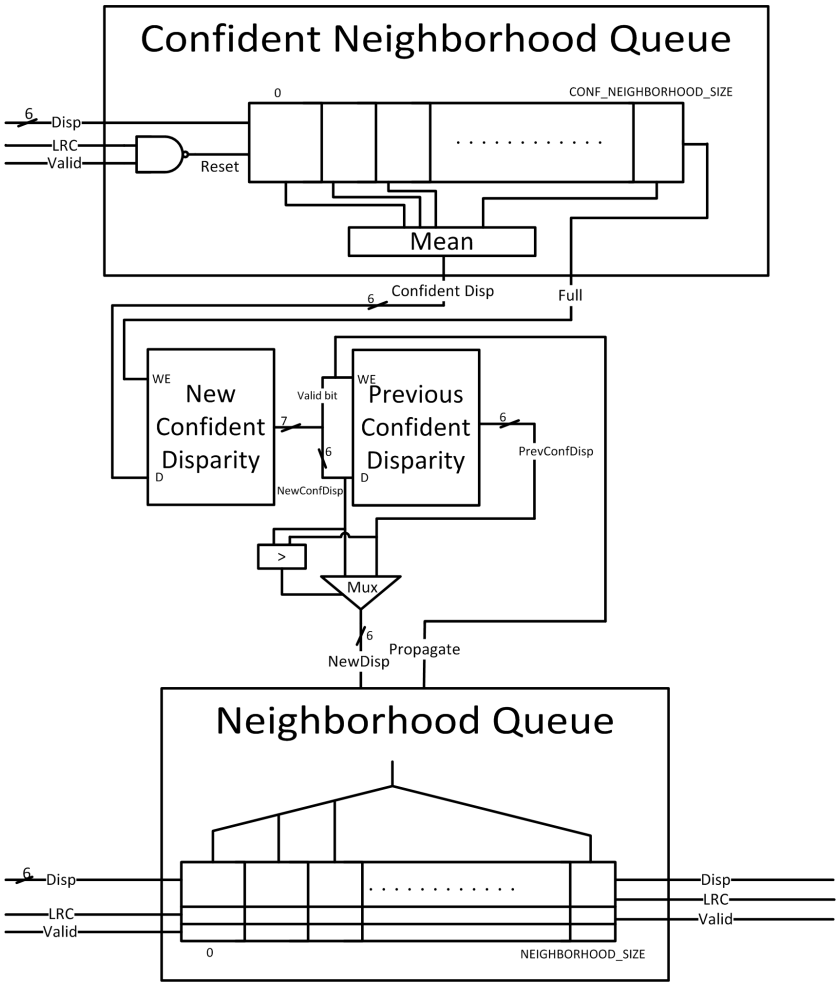


Fig 28

Finally the scan-line belief propagation algorithm discussed earlier was implemented as shown in the datapath in Fig 28. The function of this component is based on two queues: the *Confident Neighborhood Queue* and the *Neighborhood Queue*. As it is implied by its name, the *Confident Neighborhood Queue* places quality constraints on its contents, meaning that only disparities that pass the LR consistency check are written in it. Furthermore, at each cycle it calculates the average of the confident disparities, as this value will ultimately be propagated to unconfident ones in the neighborhood queue. This average is calculated by a constant multiplier, using fixed point arithmetic and rounding to reduce any number representation errors.

On the other hand, the *Neighborhood Queue* simply keeps track of local disparities and their LR status. When the *Propagate* signal is asserted (active when a new confident disparity is calculated and stored in the *New Confident Disparity* register), the *NewDisp* is written to all records with a false LRC flag. *NewDisp* is selected to be *Previous Confident Disparity* when this value is smaller than *New Confident Disparity* else it is assigned *New Confident Disparity*.

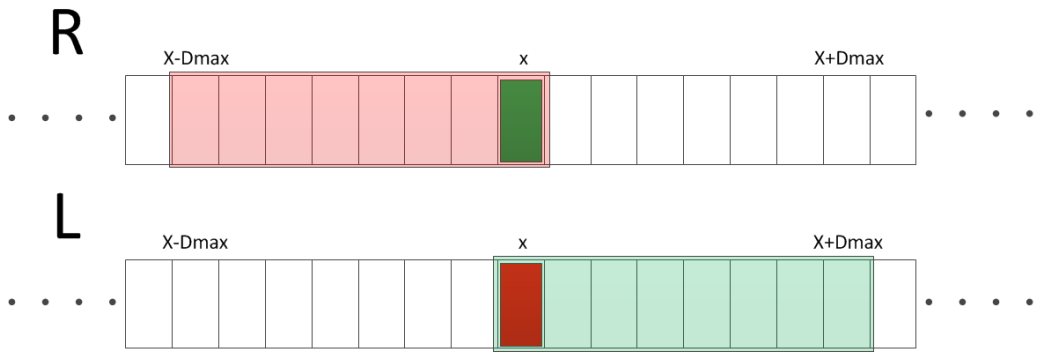


Fig 29: *At the top row are pixels of the right image, whereas at the bottom are pixels of the left image. The green pixel on the right image has a search space denoted by the green shade on the left image. In order to determine the validity of* $Disp_{RL}(x,y)$ *we need all left-to-right disparities in the green space and thus we need right-to-left costs up to* $x+D_{max}$.

Notes on operation

- A small frame of unknown disparities is formed around the final disparity image, where due to the image boundaries, the windows cannot be formed and thus disparities cannot be computed.
- Searching for matches near the image boundary leads to reduced or trivial disparity search spaces. In such cases our system finds the best match in the possible range. Left/Right Consistency check filters out any incorrect matches due to small search spaces.
- Belief propagation is activated on each end of line regardless of the LRC indication, in order to fill in the occluded right area of the image.
- In order for high resolutions to work acceptably the user needs to either increase the D_{max} accordingly or to reduce the inter-camera spacing. The second option is preferable as it has no computational consequences, whereas increasing D_{max} has a cost on FPGA resources. Moreover, as the features of the images are now more spaced out in pixel distances, the other parameters of the system need also to be adjusted, if we are to maintain output quality.

Performance

Our system receives one pixel pair per clock and after an initial latency, generates one disparity per clock. The most computationally heavy part of the main stage of the algorithm, is the xor/summing that takes place in *XOR Sum* where we have to compute the xor/sum of 64 80-bit strings at the same time. A similar situation appears in the *WTA*

module, where we have to perform 64 11-bit comparisons simultaneously. We cope with both bottlenecks through fully pipelined adder/comparator trees in order to maximize throughput.

After the first system implementation we added extra pipeline stages to further enhance performance. Below we present the differences between our initial unoptimized design and our final optimized system. Given the maximum achievable frequency of our system in both variations, we present the theoretical maximum throughput in terms of frames per second (fps) in Fig 30. We also show the difference in resources utilization between the two variations in Fig 31.

Resolutions	100×83	384×320	640×533	1024×853	1600×1333	1920×1200
Max Clock for unoptimized design: 131 MHz	15.783 fps	1.066 fps	384 fps	150 fps	61 fps	56 fps
Max Clock for optimized design: 201 MHz	24.216 fps	1.635 fps	589 fps	230 fps	94 fps	87 fps

Fig 30: *Maximum theoretical throughput of our system for a reference clock of 100Mhz and our maximum clock for various resolutions.*

Variation	Parameters	Slice Flip-Flops	LUTs	Slices	BRAMs/FIFO	Max Clock
Unoptimized	W=9, W_a=5, D_{max}=64	39.565 (57%)	37.107 (53%)	13.556 (78%)	59 (39%)	131.458 MHz
Optimized	W=9, W_a=5, D_{max}=64	41.792 (60%)	37.986 (54%)	14.239 (82%)	59 (39%)	201.518 MHz

Fig 31: *Resource utilization vs frequency for our two design variations.*

The resource utilization penalty for the greater performance design is negligible and thus our choice is clear. According to the tools the critical path lies on a control signal to the FSM of our aggregation line buffers and is only 16,6% attributed to logic while the rest 83,4% of the delay is caused by routing.

Resources analysis

Aggregation of the costs consumes most of our BRAM resources, as we have to construct D_{max} x W_a cost line buffers[1]. It is important to note that BRAM primitives in Virtex5 FPGAs support certain restricted aspect ratios. These primitives are used by an allocation algorithm to construct bigger memories. Memories with different widths/depths from those ratios are mapped to the closest possible solution but may not use the resources optimally. For example, memories with ratios of 1×16K (16.384 elements of 1 bit), 2×8K, 4×4K, 9×2K, 18×1K, 36×512 are guaranteed to utilize a single 18K primitive and thus use the resources optimally.

In addition, very large frame sizes causes parameter bloating. Specifically, images of 1800×1500 require D_{max} to exceed 180 for quality results (while not altering the current camera baseline). Keeping the other parameters constant (W=9, W_a=5), such a large D_{max} would require buffering of 180×1800×5 elements in the aggregation stage.

For the reasons discussed above we decided to impose a limit on the images width. Simply restricting frame width to 1024 pixels allows us to:

1. Pack at least two lines per 18K BRAM using a 9×2K primitive configuration. To each cost line we allocate 9×1024 bits.
2. Avoid excessive parameter bloating.

Using AD-Census, the costs are 9-bit long as explained earlier. This is something that benefits our design as BRAM primitives can be used optimally in a 9×2K configuration. Using pure Census, cost size is reduced to 7 bits. We can maximize BRAM usage by using 9-bit costs, so we have room to increase window size W up to 21×21, with little additional cost to resource usage.

If our cost size is less than 9 bits or if our frame width is less than 1024 we can pack more lines. This aspect of our design is also parametric, as depending on the frame size and cost size, each BRAM can pack up to 6 lines in a 36×512 BRAM configuration.

In an effort to reduce BRAM consumption even further, we performed a cost size-accuracy tradeoff analysis which can be seen in Fig 32. AD-Census was redefined as:

$$ADCensus' = min(ADCensus, SaturationValue)$$

Selecting saturation values to be powers of 2, we can reduce cost size and thus fit more data into the aggregation buffer BRAMs. Our analysis shows that there is even a slight benefit in doing so: For a saturation value of 63 (cost size is reduced to 6 bits) and for the default W and W_a values of 9 and 5 respectively, we observe a 0.5% improvement over the cost without saturation, which puts our quality almost on par with a W=11 and W_a=5 parameter set. This slight improvement is attributed to the reduction of the influence of outliers within the aggregation window by truncating the cost. With 6-bit costs, we can pack 3 streams of costs per *Aggregation Lines Buffer* thus reducing BRAM consumption even more. Note that FPGA resource utilization figures refer to all optimizations discussed in place.

1 A total of D_{max} x W_a x FRAME_WIDTH x COST_SIZE bits must be buffered.

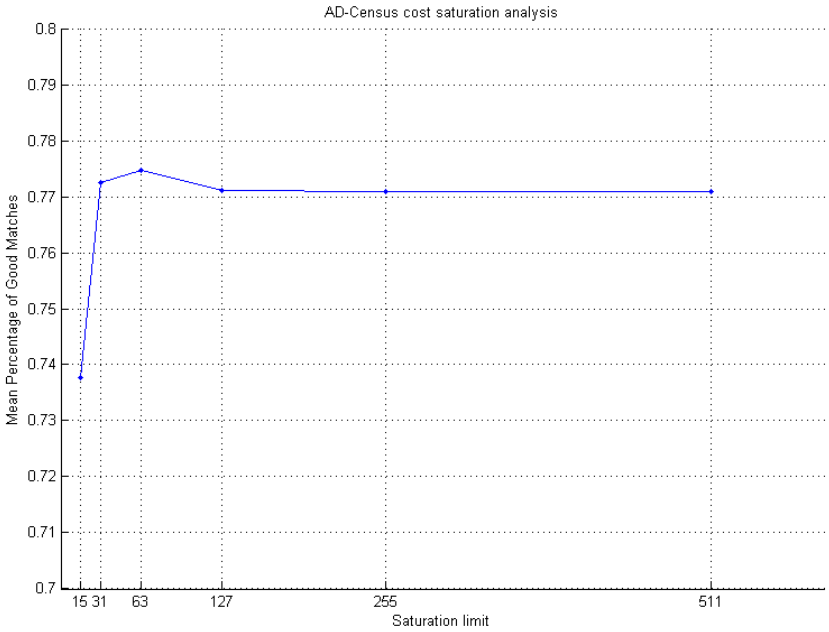


Fig 32: *Cost size/accuracy analysis. The peak value shifts to the right as the true maximum cost increases.*

Fig 33 shows the effect of the optimizations discussed above on our BRAM count for W=9, W_a=5, D_{max}=64 and a maximum frame width of 1024 pixels. It is advised to use smaller frame sizes for optimal algorithm performance.

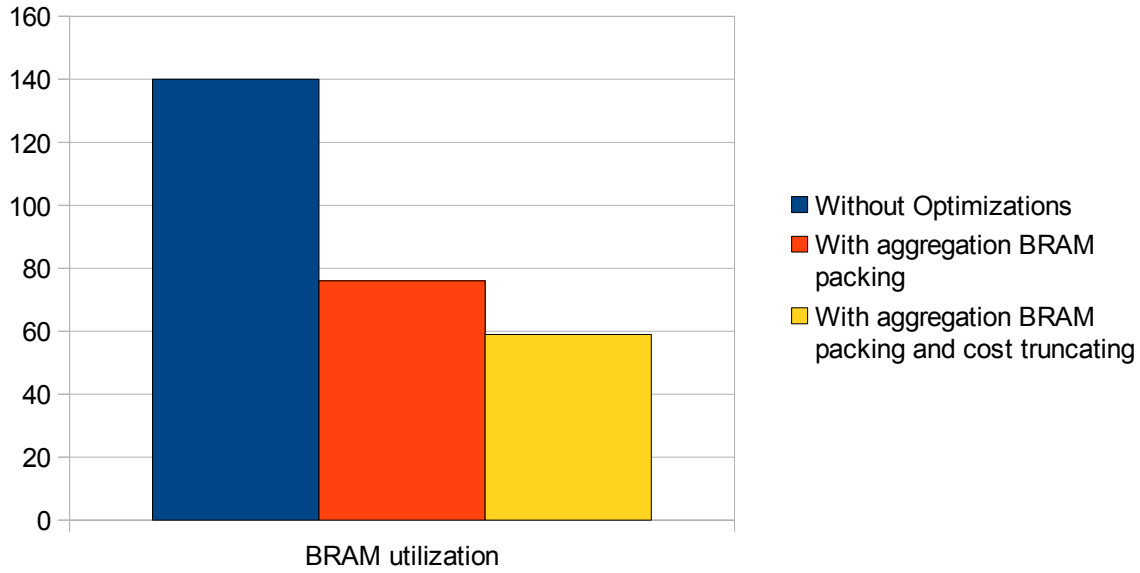


Fig 33: *Resource savings with optimized aggregation buffer structure.*

In order to conserve resources one could suggest to introduce two clock domains instead of one, where the core functions at a multiple of the pixel clock. This way the core could compute two times the data in the same time (e.g xor summing twice the census bit-strings and thus double the disparity search range). However such a scheme would stumble at the aggregation unit's architecture. The aggregation line buffers must provide enough data to construct windows for each disparity, expecting the adjacent pixel's D_{max} costs at each cycle, in order to complete buffering the current scan-line. Providing the full D_{max} costs in multiple cycles instead of one would lead to thrashing the buffers. In order to avoid this thrashing, we would need to double the size of the aggregation line buffers. Unfortunately that goes against our initial goal of reducing resource utilization, especially if we take into account the fact that our design has already high memory requirements.

Applying extensive pipelining has large demands in register resources. Fig 34 and Fig 35 show the resource utilization per subsystem on a Virtex 5 XC5VLX110T. Fig 36 shows the resource utilization of our system, for various configurations. All values are post place and route. Parameters are set to W=9, W_a=5 and D_{max}=64 unless specified otherwise.

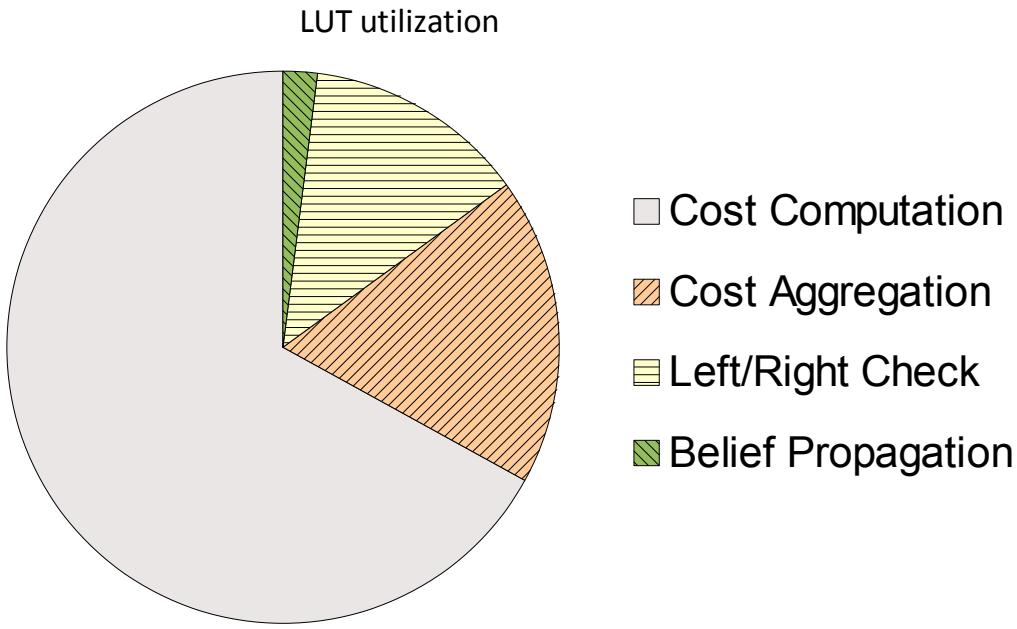


Fig 34: *LUT utilization denotes logic distribution in our design.*

D_{max}=64, W=9, W_a=5	LUTs (%)	Flip-Flops (%)	BRAMs (%)
Available	69.120	69.120	148
Total Consumed	37.986 out of 69.120 (55%)	41.784 out of 69.120 (60%)	59 out of 148 (40%)
AD Census	25.135 out of 37.986 (66%)	29.167 out of 41.784 (70%)	8 out of 59 (14%)
Aggregation	6.547 out of 37.986 (17%)	7.312 out of 41.784 (17%)	51 out of 59 (86%)
Left/Right Check	4.638 out of 37.986 (12%)	4.734 out of 41.784 (11%)	0 out of 59 (0%)
Scanline Belief Propagation	543 out of 37.986 (1,5%)	634 out of 41.784 (1,5%)	0 out of 59 (0%)

Fig 35: *Resources utilization per subsystem.*

Constant D_{max}, W_a	Slice Flip-Flops	LUTs	Slices	BRAMs/FIFO	Max Clock
W=5, W_a=5, D_{max}=64	21.866 (31%)	21.637 (31%)	8.204 (47%)	59 (39%)	201.086 MHz
W=7, W_a=5, D_{max}=64	31.840 (46%)	29.813 (43%)	11.222 (63%)	59 (39%)	201.113 MHz
W=9, W_a=5, D_{max}=64	41.784 (60%)	37.986 (55%)	14.239 (82%)	59 (40%)	201.518 MHz
Constant W, W_a	Slice Flip-Flops	LUTs	Slices	BRAMs/FIFO	Max Clock
W=9, W_a=5, D_{max}=16	12.531 (18%)	10.284 (14%)	4.216 (24%)	30 (20%)	201.207 MHz
W=9, W_a=5, D_{max}=32	22.687 (32%)	19.148 (27%)	8.274 (47%)	30 (20%)	201.045 MHz
W=9, W_a=5, D_{max}=64	41.784 (60%)	37.986 (55%)	14.239 (82%)	59 (40%)	201.518 MHz
Constant W, D_{max}	Slice Flip-Flops	LUTs	Slices	BRAMs/FIFO	Max Clock
W=9, W_a=1(off), D_{max}=64	33.047 (47%)	28.505 (41%)	11.713 (67%)	9 (6%)	201.005 MHz
W=9, W_a=3, D_{max}=64	38.660 (55%)	34.618 (50%)	13.546 (78%)	31 (20%)	201.167 MHz
W=9, W_a=5, D_{max}=64	41.784 (60%)	37.986 (55%)	14.239 (82%)	59 (40%)	201.518 MHz

Fig 36: *Resources utilization for the maximum performance implementation of the system for various configurations. Implemented on a Virtex5 XC5VLX110T, speed grade -1.*

Chapter 5
Validation

Our system's function was verified using significantly downscaled dataset images to accelerate simulation. The datasets were converted to pgm format and read into our testbench through a library. Subsequently the result was also written in a pgm file. Initial comparison with the software implementation was performed in the testbench, through a pixelwise absolute difference with the output. A threshold of 1.0 was in place as in [11]. Some variation was detected which was confirmed to be caused by a different format of inputs (MATLAB processed png images while the implementation was given pgm) and a small inconsistency in the disparity search range (MATLAB used a 0-64 range while the implementation used 0-63).

After simulation, the system was also verified on two systems, a Xilinx ML505 Board equipped with a Virtex 5 XC5VLX110T FPGA as well as a Digilent Spartan 3 1000 platform. Fig 37 shows what was validated in which platform. BRAMs on the Spartan 3 were limited, so we were forced to disable aggregation and belief propagation on that platform.

We now discuss the testing methodology we followed. Block RAM resources on the FPGAs were initialized through coe files with the test datasets. The stereo vision core processes the data and writes the result in a FIFO. The contents of the FIFO are subsequently sent through an RS232 connection, with 9600 b/s, to a host computer. In order not to overwhelm the input buffer of the host computer, which usually has a size of 4096 bytes, we implemented a simple flow control solution. Our system's output was in a custom encoding instead of ASCII or UTF-8, so we also implemented a small program on the host computer, which translates and saves the data to a pgm image file. Due to the low bitrate of RS232 compared to the throughput of our system (three orders of magnitude slower), the FIFO must be large enough to fit the whole frame.

Our validation setup can be seen in Fig 38. Fig 39 shows a comparison between our software and hardware implementations.

<table>
<tr><th></th><th colspan="3">Implemented (Placed & Routed)</th><th colspan="2">Validated on board</th></tr>
<tr><td>Spartan 3 1000</td><td>W=9, W_a=1 (off), D_{max}=16†</td><td>W=9, W_a=3, D_{max}=16</td><td>W=5, W_a=5, D_{max}=16</td><td colspan="2">W=9, W_a=1 (off), D_{max}=16 †</td></tr>
<tr><td>Virtex 5 XC5VLX110T</td><td colspan="3">All configurations in Fig 36</td><td>W=9, W_a=5, D_{max}=16</td><td>W=9, W_a=5, D_{max}=64</td></tr>
</table>

†: Without Belief Propagation

Fig 37: *Values of D_{max} = 16 correspond to 100×40 frame sizes whereas D_{max} = 64 was tested with 400×320 frames.*

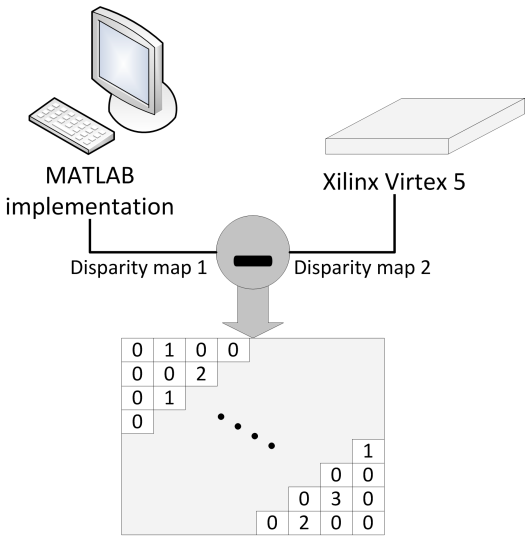


Fig 38: *Validation methodology.*

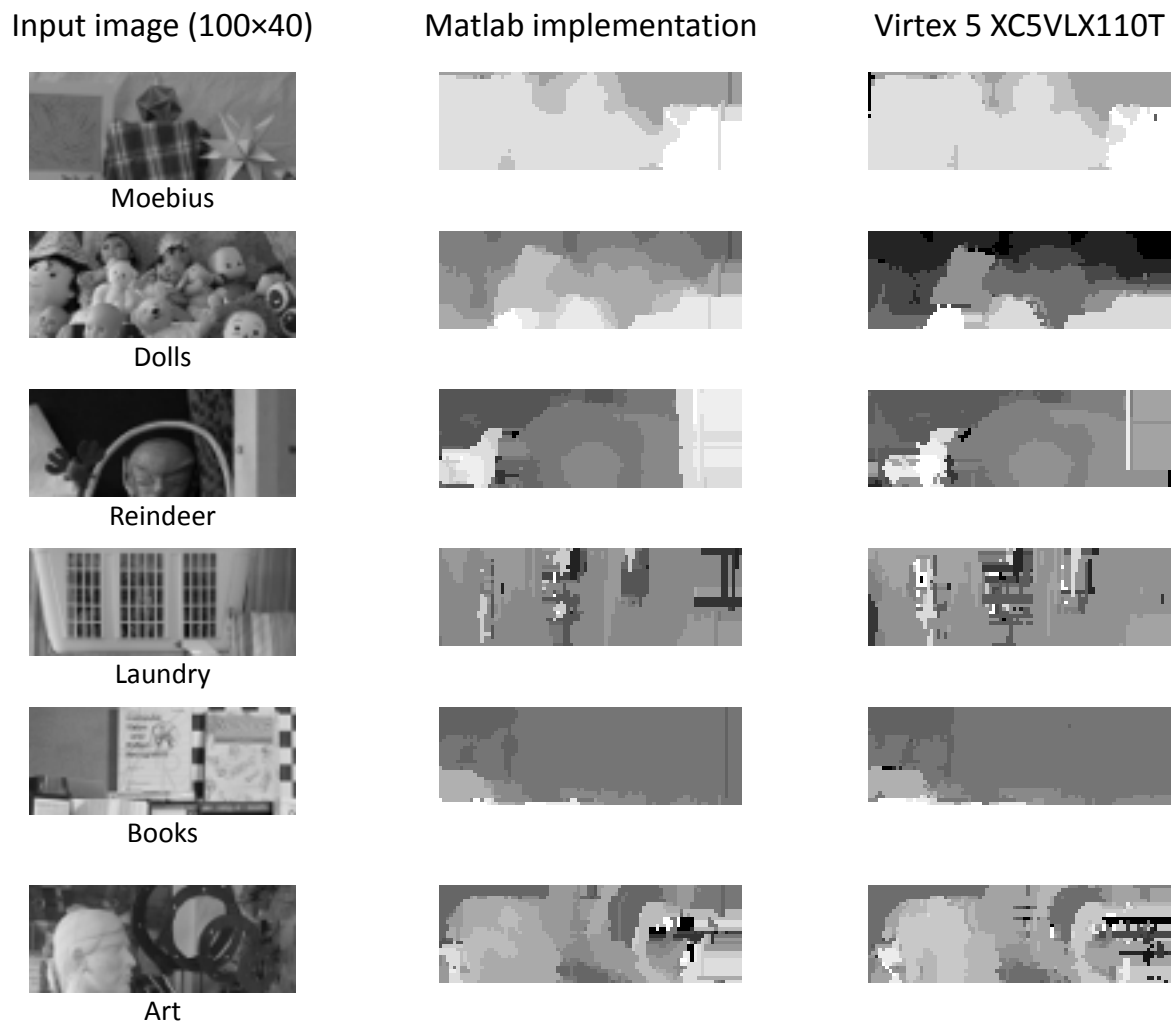


Fig 39: *Comparison between software and hardware implementation. Mean disparity error was calculated to 0.6/15 => 4%.*

The difference between software and hardware is attributed to variations in the Winner-Take-All process, where when comparing equal quantities, a different outcome takes place in the two implementations respectively. Software selects one of the two equal values randomly while hardware always selects the first value. Fig 40 demonstrates a pixel-wise comparison between software and hardware, where black pixels represent deviations larger than 1.

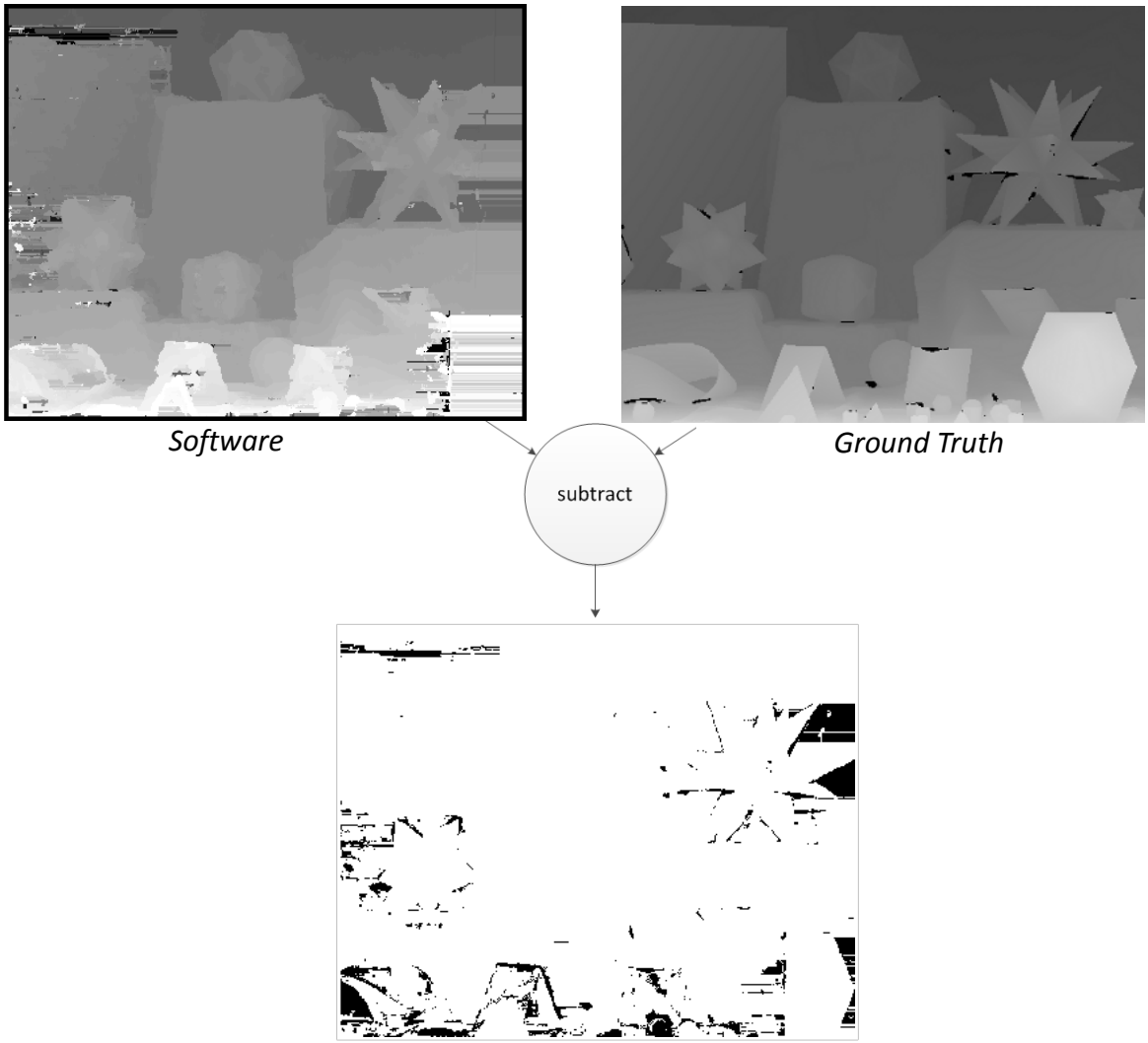


Fig 40: *Diff map between software and ground truth for the Moebius dataset in 400x320 resolution. Black stands for large variations.*

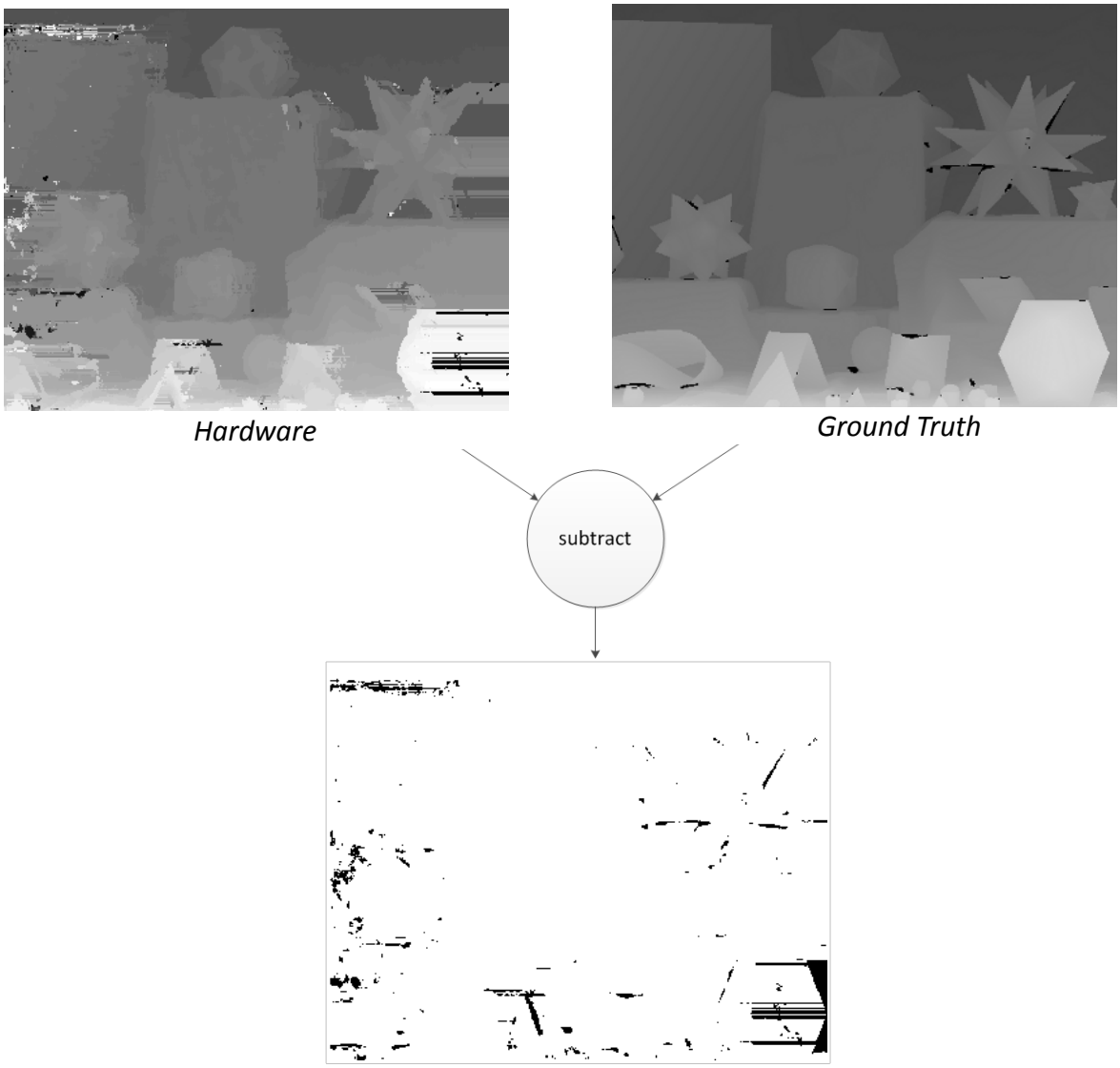


Fig 41: *Diff map between hardware and ground truth for the Moebius dataset in 400x320 resolution. Black stands for large variations.*

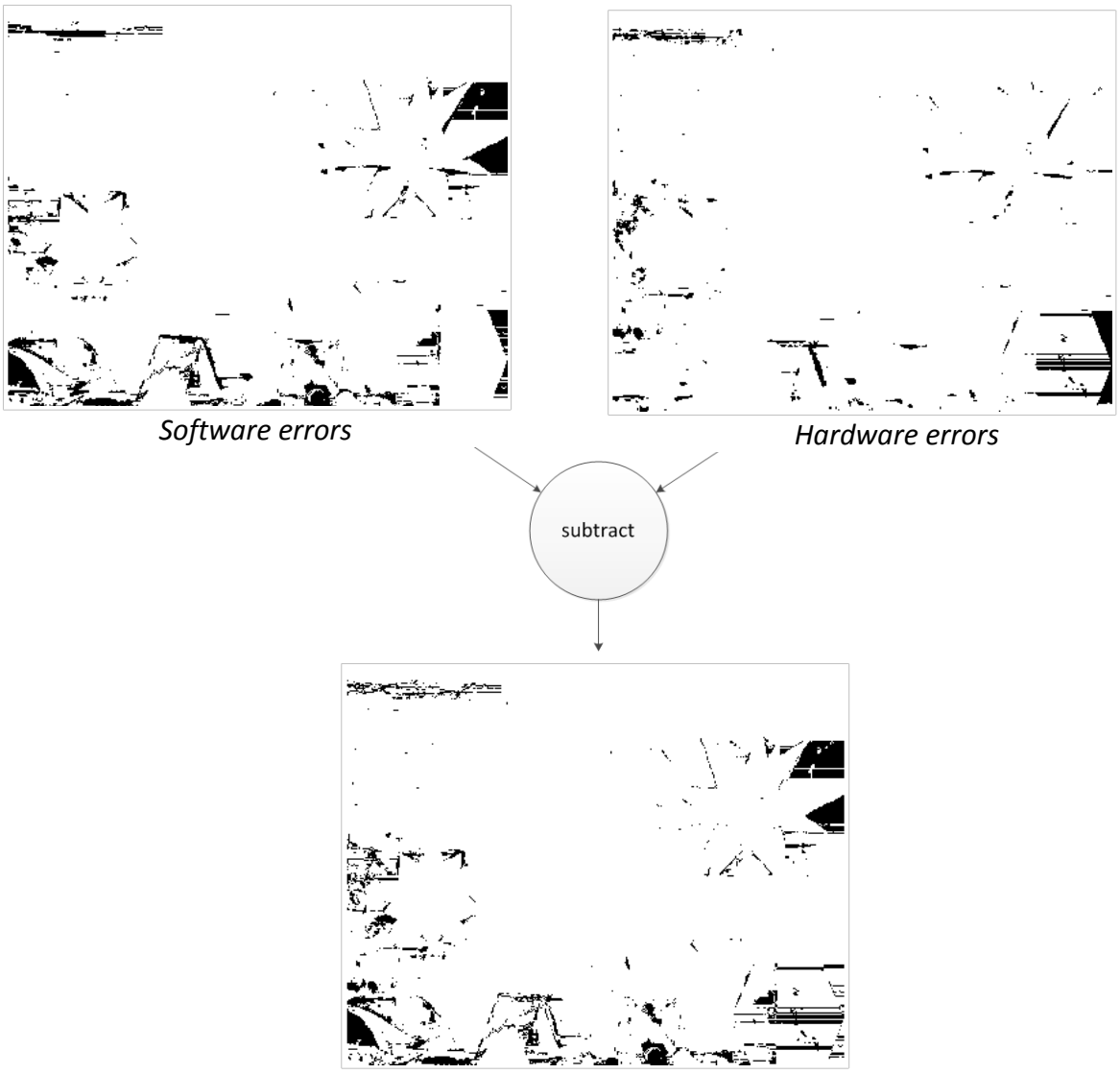


Fig 42: *Diff map between software and hardware diff maps.*

References

[1] N. K. Ratha, A. K. Jain, “Computer Vision Algorithms on Reconfigurable Logic Arrays”, *IEEE Transactions on Parallel and Distributed Systems*, Jan. 1999, Vol. 10, No. 1, pp. 29-43.

[2] C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt and M. L. Chang , “Low-Cost Stereo Vision on an FPGA”, in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* 2007, pp. 333-334.

[3] K. Ratnayake, A. Amer, “An FPGA-Based Implementation of Spatio-Temporal Object Segmentation”, in *IEEE International Conference on Image Processing (ICIP),* 2006, pp. 3265-3268.

[4] K. Konolige, “Small Vision Systems: Hardware and Implementation”, in *International Symposium on Robotics Research*, 1997, pp. 111-116.

[5] B. Rajan, S.Ravi, “FPGA Based Hardware Implementation of Image Filter With Dynamic Reconfiguration Architecture”, in *IJCSNS International Journal of Computer Science and Network Security*, Dec. 2006, Vol. 6, No. 12, pp. 121-127.

[6] D. K. Masrani, W. J. MacLean, “A Real-Time Large Disparity Range Stereo-System using FPGAs”, in *Proceedings of the IEEE International Conference on Computer Vision Systems*, 2006, pp. 42-51.

[7] C. Claus, A. Laikat, L. Jia, W. Stechele, “High performance FPGA based optical flow calculation using the census transformation”*,* in *IEEE Intelligent Vehicles Symposium*, 2009, pp. 1185-1190.

[8] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. K. Park, M. Kim, J. W. Jeon, “FPGA Design and Implementation of a Real-Time Stereo Vision System”, in *IEEE Transactions on Circuits and Systems for Video Technology*, Jan. 2010, Vol. 20, No. 1, pp. 15-26.

[9] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, M. Vincze, “A fast stereo matching algorithm suitable for embedded real-time systems”, article in *Computer Vision and Image Understanding,* Mar. 2010. [Online]. Available: www.elsevier.com/locate/cviu

[10] S. Hadjitheophanous, C. Ttofis, A. S. Georghiades, T. Theocharides, “Towards Hardware Stereoscopic 3D Reconstruction, A Real-Time FPGA Computation of the Disparity Map”, in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar 2010, pp. 1743-1748.

[11] D. Scharstein, R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms", in *International Journal of Computer Vision*, Apr. 2002, Vol. 47, No. 1-3, pp. 7-42.

[12] J. Salmen, M. Schlipsing, J. Edelbrunner, S. Hegemann, S. LükeReal-Time, “Stereo Vision: Making more out of Dynamic Programming”, in *Computer Analysis of Images and Patterns*, 2009, Vol. 5702/2009, pp. 1096-1103.

[13] R. Zabih, J. Woodfill, “Non-parametric local transforms for computing visual correspondence”, in *Proceedings of the third European conference on Computer Vision (ECCV)*, 1994, Secaucus, NJ, USA: Springer-Verlag New York, Inc., pp. 151-158.

[14] G. Xiong, X. Li, H. Chen, D. Lee, “Color Rank and Census Transforms using Perceptual Color Contrast”, in *International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Dec. 2010, pp. 1225-1230.

[15] C. Rhemann, A. Hosni, M.Bleyer, C. Rother, M. Gelautz, “Fast Cost-Volume Filtering for Visual Correspondence and Beyond”, in *Proceedings of IEEE Computer Vision and Pattern Recognition (CVPR)*, 2011.

[16] Stefano Mattoccia, "Stereo vision: algorithms and applications", VIALAB Bologna, November 2011. http://www.vision.deis.unibo.it/smatt/stereo.htm