ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/223838702

A fast stereo matching algorithm suitable for embedded real-time systems

Article *in* Computer Vision and Image Understanding · November 2010

DOI: 10.1016/j.cviu.2010.03.012 · Source: DBLP

CITATIONS
160

READS
3,346

5 authors, including:

Christian Zinner
AIT Austrian Institute of Technology
19 PUBLICATIONS **342** CITATIONS
SEE PROFILE

Markus Vincze
TU Wien
450 PUBLICATIONS **5,218** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

R3-COP View project

ER4STEM View project

All content following this page was uploaded by Martin Humenberger on 30 July 2018.

The user has requested enhancement of the downloaded file.

A fast stereo matching algorithm suitable for embedded real-time systems

Martin Humenberger[a], Christian Zinner[a], Michael Weber[a], Wilfried Kubinger[a], Markus Vincze[b],

[a]*AIT Austrian Institute of Technology, Donau-City-Strasse 1, 1220 Vienna, Austria*
[b]*Automation and Control Institute (ACIN), Vienna University of Technology, Gusshausstrasse 27-29, 1040 Vienna, Austria*

Abstract

In this paper, the challenge of fast stereo matching for embedded systems is tackled. Limited resources, e.g. memory and processing power, and most importantly real-time capability on embedded systems for robotic applications, do not permit the use of most sophisticated stereo matching approaches. The strengths and weaknesses of different matching approaches have been analyzed and a well-suited solution has been found in a Census-based stereo matching algorithm. The novelty of the algorithm used is the explicit adaption and optimization of the well-known Census transform in respect to embedded real-time systems in software. The most important change in comparison with the classic Census transform is the usage of a sparse Census mask which halves the processing time with nearly unchanged matching quality. This is due the fact that large sparse Census masks perform better than small dense masks with the same processing effort. The evidence of this assumption is given by the results of experiments with different mask sizes. Another contribution of this work is the presentation of a complete stereo matching system with its correlation-based core algorithm, the detailed analysis and evaluation of the results, and the optimized high speed realization on different embedded and PC platforms. The algorithm handles difficult areas for stereo matching, such as areas with low texture, very well in comparison to state-of-the-art real-time methods. It can successfully eliminate false positives to provide

Email addresses: martin.humenberger@ait.ac.at (Martin Humenberger), christian.zinner@ait.ac.at (Christian Zinner), michael.weber@ait.ac.at (Michael Weber), wilfried.kubinger@ait.ac.at (Wilfried Kubinger), vincze@acin.tuwien.ac.at (Markus Vincze)

reliable 3D data. The system is robust, easy to parameterize and offers high flexibility. It also achieves high performance on several, including resource-limited, systems without losing the good quality of stereo matching. A detailed performance analysis of the algorithm is given for optimized reference implementations on various commercial of the shelf (COTS) platforms, e.g. a PC, a DSP and a GPU, reaching a frame rate of up to 75 fps for 640×480 images and 50 disparities. The matching quality and processing time is compared to other algorithms on the Middlebury stereo evaluation website reaching a middle quality and top performance rank. Additional evaluation is done by comparing the results with a very fast and well-known sum of absolute differences algorithm using several Middlebury datasets and real-world scenarios.

Key words: stereo matching, real-time stereo, Census, embedded computer vision, DSP, GPU

---

1. Introduction

For modern mobile robot platforms, dependable and embedded perception modules are very important for successful autonomous operations like navigation, visual servoing, or grasping. Especially 3D information about the area around the robot is crucial for reliable operations in human environments. State-of-the-art sensors like laser scanners or time-of-flight methods deliver 3D information, that is either rough or has low resolution with respect to time and space. Stereo vision is a technology that is very well suited for delivering a precise description within its field of view. Stereo is purely a passive technology that primarily uses only two cameras and a processing unit to do the matching and 3D reconstruction.

However, for extracting dense and reliable 3D information from the observed scene, stereo matching algorithms are computationally intensive and require a large amount of hardware resources. Integrating such an algorithm in an embedded system, which is in fact limited in resources, scale, and energy consumption, is a delicate task. The real-time requirements of most robot applications complicate the realization of such a vision system as well. The key to success in realizing a reliable embedded real-time-capable stereo vision system is the careful design of the core algorithm. The trade-off between execution time and quality of the matching must be handled with care and is a difficult task. In contrast to the classic definition of the term real-time, e.g. by Kopetz [70], in this work it combines fast (at least 10 fps), constant, known and scene-independent processing time.

In this paper the challenge of fast stereo matching suitable for embedded real-time systems is tackled. An adapted, high speed and quality stereo matching algorithm especially optimized for embedded systems is presented. Furthermore, an evaluation of the results using the Middlebury stereo evaluation website and real-world scenarios is given and experimental results of reference implementations on a Personal Computer (PC), a Digital Signal Processor (DSP) and a Graphics Processing Unit (GPU) are presented. The paper is organized as follows: Section 2 introduces some fundamentals of stereo vision and the state-of-the-art in stereo matching algorithms. Section 3 gives a detailed description of the proposed real-time stereo engine. The algorithm's parameters are analyzed in detail in Section 4, and Section 5 shows the reference implementations on a PC, a DSP and a GPU. Finally, Section 6 presents evaluation results of our algorithm and Section 7 concludes the paper.

2. Stereo Vision

The main challenge of stereo vision, also called stereopsis, is the reconstruction of 3D information of a scene captured from two different points of view. This can be done by finding pixel correspondences between both images. The horizontal displacement of corresponding pixels is called disparity. Classical stereo vision uses a stereo camera setup built up of two cameras, called a stereo camera head, mounted in parallel. It captures a synchronized stereo pair consisting of the left camera's and the right camera's image. A typical stereo head is shown in Fig. 1; the distance between both cameras is called the baseline.

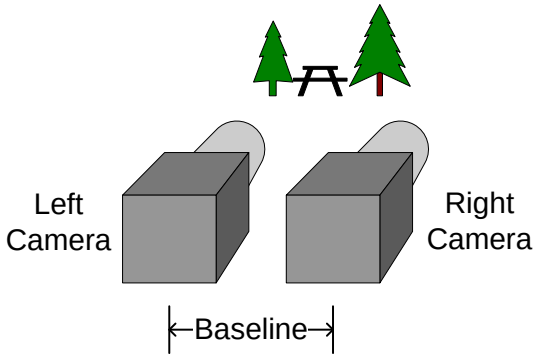


Figure 1: Typical stereo camera head.

Once the correct disparity for a pixel is found, it can be used to calculate the orthogonal distance between one camera's optical center and the projected scene

point with

$$z = \frac{b \cdot f}{d}, \tag{1}$$

where d is the disparity, b the baseline and f the camera's focal length. If 3D data should be given in camera coordinates, (2) can be used, where K is the camera calibration matrix, the pixel is given in homogeneous coordinates $(u \cdot z_c, v \cdot z_c, z_c)^T$ and z_c is calculated with (1).

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = K^{-1} \begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \end{pmatrix} \tag{2}$$

K and f have to be determined by camera calibration which is essential for fast stereo matching. On the one hand, camera lens distortion can be removed, and on the other hand, the images can be rectified. Rectified images fulfill the epipolar constraint, which means that corresponding pixel rows share the same v-coordinate, so the search for corresponding pixels is reduced to a search along one pixel row instead of through the whole image. During this process it is always assumed that the stereo image pairs are rectified. Details about camera calibration can be found in Zhang [7], Sonka et al. [2], Fusiello et al. [9] and Bradski et al. [6]. Commonly used implementations are the Caltech Calibration Toolbox, Bouguet [5], and in the OpenCV library [8].

2.1. Related Work in Stereo Matching Algorithms

A stereo matching algorithm tries to solve the correspondence problem for projected scene points and the result is a disparity map. This is an image of the same size as the stereo pair images containing the disparity for each pixel as an intensity value. In the ideal case, each scene point visible in both images which has exactly one representing pixel per image, can be determined uniquely. In practice, this is not so easy due to the vast number of scenery points in different distances which cause a pixel in the left image for instance to be mapped to a series of similar pixels in the right image. The most common problems stereo matching algorithms have to face are occluded areas, reflections in the image, textureless areas or periodically textured areas, and very thin objects. Textureless areas in particular are a major problem for stereo matching algorithms. Therefore the handling of those areas is an important aspect for the confidence of resulting matches. A good summary of many stereo matching algorithms can be found in Brown et al. [28] and Scharstein and Szeliski [29].

There are two main groups of stereo matching algorithms: feature-based and area-based algorithms. The first try to find proper features, like corners or edges, in the images and match them afterwards, while the second try to match each pixel independently to the image content. Feature-based algorithms result in a sparse disparity map because they only get disparities for the extracted features. Area-based algorithms calculate the disparity for each pixel in the image, so the resulting disparity map can be very dense. This section gives an overview of the basic matching techniques that are currently used. The techniques introduced are restricted to area-based algorithms because this work is concerned with to dense disparity maps.

Basically, an area-based stereo matching algorithm is built up as follows: First, usually some pre-processing functions are applied, e.g. a noise filter. Second, the matching costs for each pixel at each disparity level in a certain range (disparity range) are calculated. The matching cost determines the probability of a correct match. The smaller the cost, the higher the probability. Afterwards, the matching costs for all disparity levels can be aggregated within a certain neighborhood window (block). The following equations show a few popular cost calculation methods for the pixel (u,v) in I_1 as a reference image with an $n \times m$ aggregation where d is the disparity and I_2 is the corresponding image and the simplified notation

$$\sum_{i=n}\sum_{j=m} = \sum_{i=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \sum_{j=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor}. \tag{3}$$

is used. The first one is the most popular sum of absolute differences (4), the second one is called the sum of squared differences (5), the third one is the normalized cross correlation (6) and the last one is the zero mean sum of absolute differences (7). The last two make the costs invariant to additive or multiplicative intensity differences caused by different shutter times, lighting conditions or apertures of the cameras.

$$SAD = \sum_{i=n}\sum_{j=m} |I_1(u+i,v+j) - I_2(u+d+i,v+j)| \tag{4}$$

$$SSD = \sum_{i=n}\sum_{j=m} (I_1(u+i,v+j) - I_2(u+d+i,v+j))^2 \tag{5}$$

$$NCC = \frac{\sum_{i=n}\sum_{j=m} I_1(u+i,v+j) I_2(u+d+i,v+j)}{\sqrt{\sum_{i=n}\sum_{j=m} I_1(u+i,v+j)^2 \sum_{i=n}\sum_{j=m} I_2(u+d+i,v+j)^2}} \tag{6}$$

$$ZSAD = \sum_{i=n} \sum_{j=m} |(I_1(u+i, v+j) - \overline{I_1}) - (I_2(u+d+i, v+j) - \overline{I_2})| \qquad (7)$$

where

$$\bar{I} = \frac{1}{nm} \sum_{i=n} \sum_{j=m} (I(u+i, v+j)).$$

The cost aggregation makes the possible matches more unique with the assumption that pixels within the window share the same disparity level. This assumption fails on disparity discontinuities. The bigger the window size, the higher the chance for a correct match but with the drawback of quality loss at disparity discontinuities like object borders (borders become broader). Small windows increase the quality at borders and the localizing of matches is more accurate, but they can cause more false matches at difficult areas. To overcome this problem, Hirschmueller [3], Hirschmueller et al. [17] and others introduced multiple windowing aggregation strategies. With the use of integral images (Veksler [18]), the processing time is independent of the window size, so an adaptive aggregation can also be realized with a low processing time. Image pyramids can be used to reduce the disparity search range.

Finally, the algorithm searches for the best match for each pixel. At this step it can differentiate between local and global approaches. Local methods select the match with the lowest cost, independent of the other pixels aside from the nearest neighbors (because of aggregation). The most common is a simple winner-takes-all (WTA) minimum or maximum search over all possible disparity values. Global methods use the scanline or the whole image to assign a disparity value to each pixel. Global methods are dynamic programming (Ohta and Kanade [22], Birchfield and Tomasi [19], Gonzalez et al. [21], Forstmann et al. [20]), graph cuts (Boykov et al. [23], Kolmogorov and Zabih [25]) or belief propagation (Sun et al. [26]). The matching can be done from right to left and vice versa, so occlusions and uncertain matches can be filtered with a left/right consistency check. This means only disparities with the same value (within a certain range) for both directions are accepted.

The strategy of dynamic programming is to find the optimal path through all possible matches for each scanline. The ordering constraint that pixels in the reference image have the same order as their corresponding ones in the matching image specifies the possible predecessors of all matches. The one with the lowest matching and joining costs is chosen recursively. This leads to a path through the possible matches that implies a left/right consistency check. The resulting disparity maps usually suffer from horizontal streaks. Some implementations of

dynamic programming, such as the one from Birchfield and Tomasi [19] in the OpenCV library [8], have very low processing times.

The aforementioned drawback of dynamic programming is that it only considers horizontal smoothness constraints. An approach that overcomes this is graph cuts where vertical smoothness is also taken into consideration. Finding stereo correspondence with graph cuts formulates the correspondence problem as the search for the minimum cut or the maximum flow through a weighted graph. This graph has two special vertices: the source and the sink. Between those are nodes, which are connected by weighted edges. Each node represents a pixel at a disparity level and is associated with the matching cost. Each edge has an associated flow capacity that is defined as a function of the costs of the node it connects. This capacity defines the amount of flow that can be sent from source to sink. The maximum flow is equivalent to the minimum cut and is comparable to the optimal path along a scanline in dynamic programming, with the difference that it is not only consistent across one scanline, but also over the entire image. The computation of the maximum flow is very extensive, so it cannot be used for real-time applications. An implementation can also be found in the OpenCV library [8].

Another global disparity optimization approach is belief propagation. This iterative strategy uses rectangular Markov random fields for assigning the best matching disparities to the pixels. Each node is assigned to a disparity level and holds its matching cost. The belief (probability) that this disparity is the optimum arises from the matching costs and the belief values from the neighboring pixels. At every iteration, each node sends its belief value to all four connected nodes. The belief value is the sum of the matching costs and the received belief values. The new belief value is the sum of the actual and the received value and is saved for each direction separately. This is done for every disparity level and the best match is the one with the lowest belief values over all four directions. A real-time implementation on a graphics processing unit can be found in Yang et al. [27].

A different matching strategy is to first apply a local transform to the images and to match afterwards. Such transforms are the Census and the rank transform introduced by Zabih and Woodfill [16]. Both transforms are based on local intensity relations between the actual pixel and the pixels within a certain window. This relation is defined by the function

$$\xi(p_1, p_2) = \begin{cases} 0, & p_1 \leq p_2 \\ 1, & p_1 > p_2 \end{cases} \tag{8}$$

where p_1 and p_2 are pixels in the image. The Census transform uses (8) to create a bit string for every pixel in the image I, as shown in (9) where the operator $\bigotimes$

denotes a bit-wise catenation and $n \times m$ the window size.

$$I_{census}(u,v) = \bigotimes_{i=n} \bigotimes_{j=m} (\xi(I(u,v), I(u+i, v+j))) \tag{9}$$

The cost calculation for Census-transformed pixels is the calculation of the Hamming distance between the two bit strings as can be seen in (10) for a pixel (u,v) and an $n \times m$ aggregation.

$$\sum_{i=n} \sum_{j=m} Hamming(I_1(u+i, v+j), I_2(u+d+i, v+j)) \tag{10}$$

The processing time of Census-based matching strongly depends on the Census window size.

The rank transform changes each pixel to the sum of all pixels within a certain window whose intensities are less than the actual pixel, as shown in (11). For cost calculation, the SAD in (4) of the transformed pixels can be used.

$$I_{rank}(u,v) = \sum_{i=n} \sum_{j=m} (\xi(I(u,v), I(u+i, v+j))) \tag{11}$$

2.2. Available Real-Time Stereo Vision Systems

This work is related to embedded real-time stereo vision systems, so a brief overview of the available systems (not only pure embedded solutions) is given in Table 1 ordered by publication date. The processing speed of the different systems is given in frames per second (fps) and, more meaningfully, in million disparity evaluations per second (Mde/s=width×height×disps×fps).

The description of the systems introduced here is restricted to the system platform, the basic matching strategy, the image size, the number of evaluated disparities and the frame rate achieved. All performance data are directly taken from the authors' papers; the Mde/s is self-calculated. Detailed information about certain pre- or post-processing steps can be found in the literature. A few of them can also be found in the Middlebury stereo evaluation database, as will be described in Section 6.

An early system is introduced by Faugeras et al. [35], which has near real-time performance. Different normalized correlations were tested and the system was implemented for an MD96 DSP board (Mathieu [47]), a Sparc 2 workstation and an FPGA board (PeRLe-1). The input images have a size of 256×256 and the disparity range is 32. The FPGA implementation is by far the fastest, with a frame rate of 3.6 fps. The others are far away from real-time: 9.6 s for the DSP

Table 1: Stereo vision systems.

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]	2555.904	200	Census	ASIC
Forstmann et al. [20]	188.928	12.3	DP	CPU
Point Grey [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
OpenCV [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

and 59 s for the Sparc 2 version. Kanade et al. [33] introduce an SSAD (Sum of SAD) stereo system based on a custom hardware with a Texas Instruments C40 DSP array and a stereo head with up to 6 cameras. They reach real-time performance (30 fps) for images sized 200×200 with 32 disparity evaluations. The first Census-based system was designed by Woodfill and Von Herzen [34]. The matching is realized on a custom hardware board (PARTS) consisting of 16 FPGAs. A very high frame rate of 42 fps could be achieved for 320×240 images and 24 disparities. Another SSAD approach has been introduced by Kimura et al. [42]. Here, a stereo head of 3×3 cameras is used and a frame rate of 20 fps for QVGA images with 25 disparity levels is achieved. The processing is done on 2 PCI boards. Another SAD stereo matching system on FPGA has been developed by Miyajima and Maruyama [36]. The frame rate achieved is 20 fps for VGA images and 80 disparities. The first fully embedded solution, introduced here, was developed by Tyzx [39]. It is an ethernet connected embedded stereo system with integrated cameras. The stereo processing is done with a custom ASIC (Woodfill

et al. [37]) that can process images sized 512×480 and 52 disparity evaluations at 200 fps. For stereo matching, Census correlation with a window size of 7×7 is used. This enormous speed can be explained by the use of a certain stereo matching processor. The drawback to this system is its high price. A real-time stereo system based on dynamic programming has been introduced by Forstmann et al. [20]. On an AMD AthlonXP 2800+ CPU, a frame rate of 12.3 fps for VGA images and a disparity range of 50 is achieved. A detailed benchmark can be found in the reference paper. Point Grey [40] developed a PC-based stereo matching system that reaches the highest performance (83 fps) on a 2.8 GHz Intel Pentium 4 processor with an image size of 320×240 and a disparity range of 32. If higher image resolution and more disparities are needed, e.g. 640×480 and 96, the frame rate drops to 4.4 fps. The stereo matching is based on SAD. Over the years, Graphics Processing Units were used for stereo matching. A GPU-based system was introduced by Yang et al. [45]. Using an SAD-based algorithm, they achieved a frame rate of 11.5 fps for an image size of 512×512 and 94 disparities on an ATI Radeon 9800 XT graphics card. The second GPU implementation introduced here was developed by Ernst and Hirschmueller [44], who tried to overcome the drawbacks of local matching with a semi-global matching approach (SGM). Their implementation on a GeForce 8800 ULTRA GPU reaches a frame rate of 4.2 fps at 640×480 with 128 disparities and 13 fps at 320×240 with 64 disparities. The second fully embedded solution comes from Videre Design [41]. Their Stereo on a Chip (STOC) is an SAD-based, IEEE 1394 interfaced stereo system that reaches a frame rate of 30 fps at VGA image sizes with a disparity range of 64. The next two systems introduced here are both digital signal processor (DSP) implementations. Chang et al. [43] use SAD and a special 4×5 jigsaw aggregation window to achieve a frame rate of 50 fps at images sizes of 384×288 and 16 disparity levels. Khaleghi et al. [46] implemented a Census-based matching algorithm and placed emphasis on the miniaturization of the system. It is fully integrated and fits within a $5cm \times 5cm$ package. The drawback is the low image resolution of 160×120 and the small Census window size of 3×3. With these settings and a disparity search range of 30, the system achieves a frame rate of 20 fps. In the open computer vision library, OpenCV [6], Konolige published a very fast SAD-based stereo matching algorithm. It reaches a frame rate of 66.67 fps on a CPU clocked at 3 GHz for 384×288 images with 16 disparities.

One real-time algorithm, available in the Middlebury database, is published by Gong and Yang [68]. It is a dynamic programming approach implemented on an ATI Radeon 9800 XT GPU reaching 23.8 fps for 384×288 images with 16 disparities. Another GPU implementation is introduced by Wang et al. [48]. It is

based on SAD and dynamic programming and reaches a frame rate of 43 fps for QVGA images and 16 disparity levels on an ATI Radeon XL1800 graphics card. A global optimizing real-time algorithm was developed by Yang et al. [27]. The authors use hierarchical belief propagation and reach 16 fps for QVGA images with 16 disparities on a GeForce 7900 GTX graphics card. A PC-based system is introduced by Tombari et al. [67]. The authors try to overcome the problem at object borders by the use of a segmentation-based costs aggregation strategy. A frame rate of 5 fps could be achieved for 384×288 images with 16 disparities on an Intel Core Duo clocked at 2.14 GHz.

The most recently published approaches are also available in the Middlebury online ranking. Kosov et al. [73] use variational, adaptive, and multi-level methods to solve the corresponding problem and reach a frame rate of about 2.15 fps for the Tsukuba dataset (384×288, 16 disparities) on a 2.83 GHz CPU. A highly optimized GPU implementation of bitwise fast voting is presented by Zhang et al. [71]. The algorithm reaches about 57 fps on a GeForce 8800 GTX graphics card for the Tsukuba dataset. Salmen et al. [72] optimized a dynamic programming approach on a 1.8 GHz PC platform and achieved a frame rate of 5 fps for the Tsukuba dataset.

As can be seen, most stereo matching systems use correlation, especially SAD. Census-based matching promises better results than SAD but is computationally very extensive and needs parallelization for real-time performance. Thus, it is well suited for embedded systems with the potential for parallel processing.

3. Real-Time Stereo Engine

The stereo matching algorithm introduced in this section is well chosen for embedded real-time systems, especially for robotic applications such as obstacle detection (Cucchiara et al. [12]), scene classification (Burschka and Hager [15]) and robot navigation (Murray and Jennings [14], Murray and Little [11], van der Mark et al. [10], Konolige et al. [13]). First, a few requirements that have to be taken into consideration for embedded systems are explained and afterwards the stereo matching algorithm is described in detail.

3.1. Requirements for Embedded Real-Time Stereo Matching

A big advantage of stereo sensors is that they deliver a huge number of 3D points with a single measurement. This is what makes these so attractive for robotic applications. Of course, to ensure fast reactions of a robot to environmental changes, the sensor has to deliver data at high frame rates and low latencies.

A minimum of 10 fps should be achieved in any case and the algorithm has to be suitable for real-time applications, which means the calculation has to be finished within that time frame and has to be independent from the actual scene. An area-based Census correlation algorithm fulfills all these requirements.

The reliability of 3D data is also important. For instance, only 3D points with a high probability of correctness should be delivered and used for navigation. To fulfill this demand a confidence and a texture map are calculated which gives an opportunity to identify and filter uncertain matches and textureless areas. Due to the systematic constraint of stereo that depth resolution decreases with increasing distance, matches have to be as accurate as possible. In most cases, especially at larger distances, the correct disparity lies between integer disparity levels. Subpixel refinement attempts to overcome this imprecision.

Mobile robotic platforms have to often deal with different lighting conditions, so the matching algorithm has to be very robust in terms of different scene illumination of the stereo cameras. Because of its robustness and its high-quality results, the Census transform is again a good choice. The last requirement is low memory consumption because of the limited amount of resources on embedded systems. As will be described in Section 5.2, the proposed algorithm is very memory-aware.

Due to the requirement of reliable matches, the filling of occluded and textureless areas, as well as unmatched pixels, is not absolutely necessary because of the low confidence of such extrapolated disparities.

3.2. Stereo Matching Algorithm

The stereo matching algorithm proposed in this paper is used in a stereo vision system. The input stereo images are delivered by two digital cameras mounted such that they are as parallel as possible. The inputs are the calibration parameters, the disparity range, and the confidence and texture thresholds. The outputs are the disparity map, the depth image (z-map), the 3D point cloud in camera coordinates, a confidence map and a texture map.

Figure 2 shows the principle workflow of the proposed stereo matching system. Before a continuous stereo matching can be done, the stereo camera head has to be calibrated offline. Here, the undistortion and rectification maps for the stereo camera head that hold the image coordinates of the undistorted and rectified images are calculated. Once these maps are calculated, they can be used for all stereo image pairs captured with the calibrated stereo head. Details about the calculation of these maps can be found in Bradski et al. [6].

The first step of the workflow is the image acquisition by the stereo head. The proposed algorithm uses monochrome input images, so it would be advan-

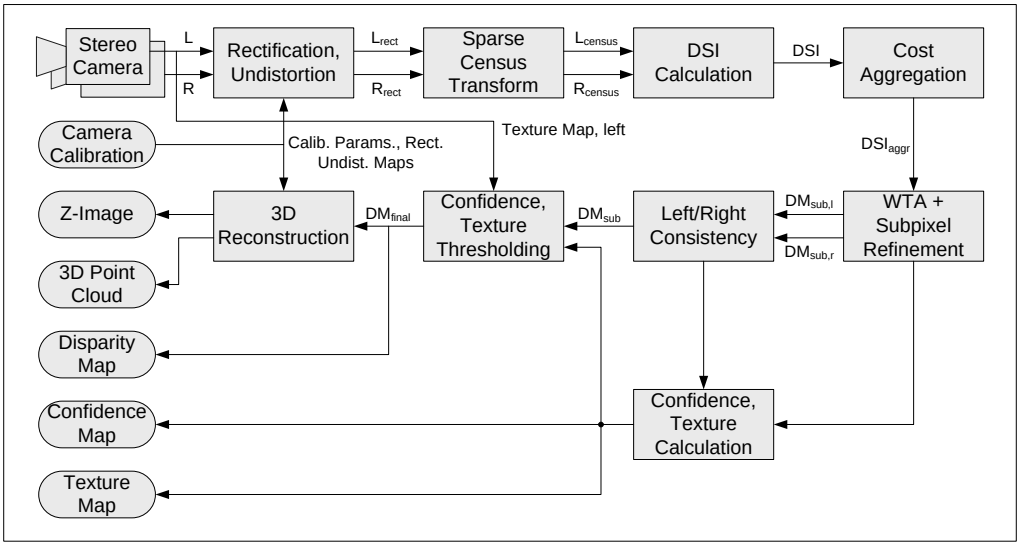


Figure 2: Stereo matching block diagram.

tageous to use monochrome cameras instead of converting color to grayscale. Monochrome cameras deliver more accurate intensity images than color cameras equipped with a Bayer filter. Another important aspect is the exact synchronicity of the stereo image capture. Especially when the camera head or the captured scene is in motion, acquisition has to be as simultaneous as possible. (Many cameras have an external trigger input, which offers the possibility of triggering two cameras at exactly the same time.)

Once the stereo images, L and R, are captured, undistortion and rectification follows with

$$L_{rect}(u,v) = L(mapx_l(u,v), mapy_l(u,v)) \tag{12}$$

and

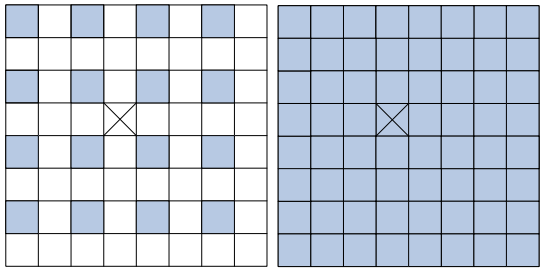
$$R_{rect}(u,v) = R(mapx_r(u,v), mapy_r(u,v)). \tag{13}$$

where the offline calculated undistortion and rectification maps, $mapx_l$, $mapy_l$, $mapx_r$, and $mapy_r$, are used to remap the images. Bilinear interpolation is used to calculate the pixel value because the maps are given in subpixel accuracy. After that, the images are Census transformed. Based on the balanced tradeoff between quality loss and performance gain shown in Section 4, a Census transform, hereinafter referred to as sparse Census transform, is used. Zabih [24] also mentions the idea of efficient Census matching in his dissertation. He used the fact that if pixel P' lies within the Census mask of pixel P, the relative value between these

pixels is calculated twice (see equation (8) for more detail). The use of certain neighborhoods allows the avoidance of double calculation and reduces the total number of comparisons. The mask configurations obtain a rather irregular structure which is very unfavorable for performance optimized implementations on modern processors. The advantage in saving half of the pixel comparisons would be overcompensated by the overhead caused by the irregular memory accesses.

The approach used in this work, keeps the mask size as large and symmetric as possible by using only every second pixel and every second row of the mask for the Census transform, as shown in Fig. 3 for an 8×8 mask. The filled squares are the pixels used for the Census and the sparse Census transform. Avoiding the double comparisons here is not the key to minimize the processing time, but it is assumed that large sparse Census masks perform better than small normal (dense) Census masks with the same weight of the resulting bit strings. Thus it is anticipated that sparse 16×16 Census performs better than 8×8 normal Census, where both have a bit string weight of 64 and thus need the same processing time. A detailed analysis of this can be found in Section 4. This approach still yields high matching quality while still enabling efficient implementations on modern processor architectures.

Of course, the used checkerboard could be adapted to fulfill Zabih's approach, avoiding point symmetries according to the center pixel, but analysis showed that it yields no improvement for this kind of neighborhood.



(a) sparse (b) normal

Figure 3: Census masks.

After evaluating different mask sizes, as explained in Section 4, a mask size of 16×16 was chosen for optimized implementation of the proposed algorithm. The reasons for this are firstly the high quality which this Census mask size delivers, and secondly the efficient memory access of registers with a size that is a multiple of 32 bits. The drawback of even mask sizes is that the anchor point cannot be

exactly in the middle. Figure 3 shows that the sparse Census transform overcomes this drawback because the rightmost column and the bottom row are discarded. The calculation of the sparse Census-transformed images L_{census} and R_{census} is based on (9) and performed with

$$L_{census}(u,v) = \bigotimes_{n \in N} \bigotimes_{m \in M} \xi(L_{rect}(u,v), L_{rect}(u+n, v+m)) \tag{14}$$

and

$$R_{census}(u,v) = \bigotimes_{n \in N} \bigotimes_{m \in M} \xi(R_{rect}(u,v), R_{rect}(u+n, v+m)) \tag{15}$$

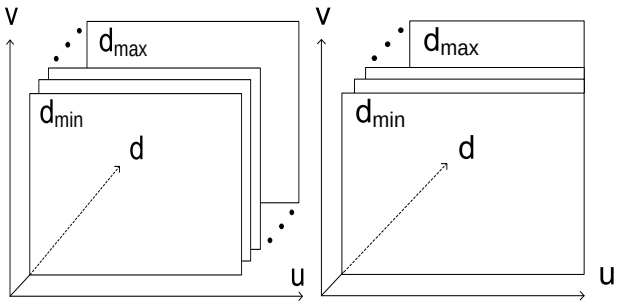
where

$$\xi(p_1, p_2) = \begin{cases} 0, & p_1 \leq p_2 \\ 1, & p_1 > p_2 \end{cases} \tag{16}$$

and

$$N = M = \{-7, -5, -3, -1, 1, 3, 5, 7\}. \tag{17}$$

The next step is the matching part itself. For each pixel, the costs for each possible match have to be calculated. The calculated costs are stored in the so-called disparity space image (DSI), which is a three-dimensional data structure of size $disps \times width \times height$. For each disparity level there exists a slice of a 3D matrix as shown in Fig. 4.



(a) normal (b) Memory-reduced

Figure 4: Two DSI possibilities.

On the left side, a classic DSI and on the right side a memory-reduced version are illustrated. It is possible to reduce the size of the DSI because per disparity level, only $width - d$ pixels are possible matching candidates. If the matching is done from right to left, the pixels on the right side of the right image have

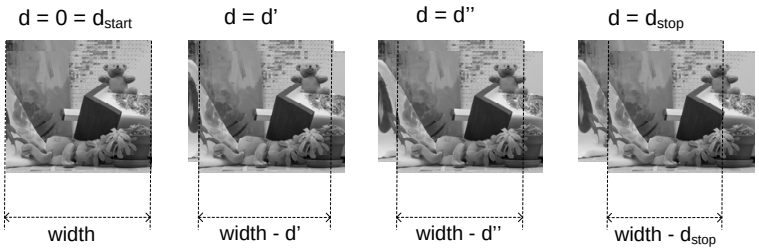


Figure 5: The widths of the DSI levels shrink with the disparity.

no matching candidates, as can be seen in Fig. 5. The amount of these pixels increases with the disparity level.

Particularly with the use of the classic Census transform, the calculation of the DSI is computationally very extensive because the Hamming distance has to be calculated for each pixel at each possible disparity level (from d_{start} to d_{stop}) over bit strings of 256-bit weights. The sparse Census transform reduces the bit weight to 64 bits, since only every fourth pixel is used. The DSI is calculated according to

$$\forall d \in [d_{start}, d_{stop}] : DSI_d(u,v) = Hamming(R_{census}(u,v), L_{census}(u+d,v)). \quad (18)$$

It is assumed that neighboring pixels, except at disparity discontinuities, have a similar disparity level, so a cost aggregation makes the matches more unique. The larger the block size used, the larger the impreciseness at object borders. A fairly good compromise has been found at a size of 5×5 (Section 4). In order to keep the good trade-off between quality and processing time, a simple squared window aggregation is used. The aggregation itself is a sum over a window with the specified block size (convolution), defined as

$$\forall d \in [d_{start}, d_{stop}] : DSI_{d,aggr}(u,v) = \sum_{n \in N} \sum_{m \in M} DSI_d(u+n, v+m). \quad (19)$$

After calculating all possible matches, the best match has to be found. As explained above, the best match is the one with the lowest cost. Figure 6 shows a typical cost function. The black circles show the costs at integer disparity levels.

It can be seen that the lowest cost is at disparity level d_{min}. This level wins by the use of a winner-takes-all (WTA) minimum search. It delivers integer disparities, but the true disparities lie between them in most cases. To calculate the so-called subpixel disparities, a parabolic fitting is used. The best integer disparity and its neighbors are used to span the parabola shown in Fig. 6, and its minimum

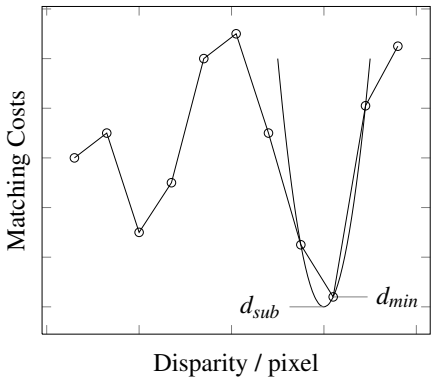


Figure 6: Example of a cost function.

gives the disparity in subpixel accuracy. From now on, $y(d)$ means the cost of a match at disparity d for a certain pixel. The subpixel disparity for one pixel is calculated with

$$d_{sub} = d_{min} + \frac{y(d_{min}+1) - y(d_{min}-1)}{2(2y(d_{min}) - y(d_{min}-1) - y(d_{min}+1))}. \tag{20}$$

where the coordinates (u,v) are omitted in the equation. The whole disparity map in subpixel accuracy for both matching directions is calculated with

$$DM_{sub,l}(u,v) = d_{sub,l}(u,v) \tag{21}$$

and

$$DM_{sub,r}(u,v) = d_{sub,r}(u,v). \tag{22}$$

To filter occluded and uncertain matches, a left/right consistency check is applied to $DM_{sub,l}$ and $DM_{sub,r}$ in

$$a = DM_{sub,l}(u,v) \qquad b = DM_{sub,r}(u-a,v) \tag{23}$$

and

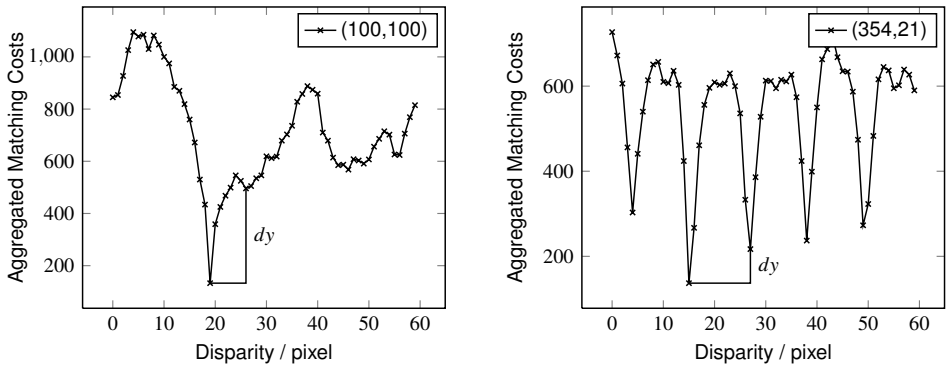
$$DM_{sub}(u,v) = \begin{cases} |\frac{a+b}{2}|, & |a-b| \leq 1 \\ 0, & \text{else} \end{cases}. \tag{24}$$

Another result calculated from the cost function is a confidence value that indicates the reliability of the match. It is defined by the relation of the absolute cost

difference between the best two matches, as denoted with dy in Fig 7, and the maximum possible cost ($y_{max} = 64 \times 5 \times 5 = 1600$, for a 16×16 sparse Census mask). The whole confidence map is calculated by

$$CM(u,v) = min\left(255, 1024\frac{dy(u,v)}{y_{max}}\right). \tag{25}$$

For better visualization, the confidence value is scaled by the factor 1024 and saturated at 255. Figure 7 shows cost functions of two different pixels. The first one contains a clear minimum, so the confidence will be high. The second has many similar peaks, which results in a low confidence level.



(a) (b)

Figure 7: Cost functions for two different pixels.

Particularly in real-world environments, textureless areas are quite common and this is a known problem for stereo matching algorithms. It is impossible to match complete textureless areas larger than the Census mask size with a local optimization technique. Many applications only deal with sufficiently safe matches, so textureless areas should be masked out. A texture map is calculated for this purpose by

$$TM(u,v) = \frac{1}{nm}\sum_{i=n}\sum_{j=m} L(u+i,v+j)^2 - \frac{1}{nm}\left(\sum_{i=n}\sum_{j=m} L(u+i,v+j)\right)^2, \tag{26}$$

by using an $n \times m$ variance filter. In this work, the kernel size is experimentally defined and set to 11×11.

Finally, on the left/right-checked disparity map, the confidence and texture maps are applied by a simple thresholding as shown in (27). The thresholds are γ for confidence and τ for texture.

$$DM_{final}(u,v) = \begin{cases} DM_{sub}(u,v), & CM(u,v) \geq \gamma \quad \wedge \quad TM(u,v) \geq \tau \\ 0, & \text{else} \end{cases} \tag{27}$$

The last step is the calculation of the z-image using (1) and the 3D point cloud with respect to the left camera's coordinate system using (2).

4. Parameter Analysis

In this section, the algorithm parameters described in Section 3 are analyzed in detail. The goal was to find a proper Census mask size, aggregation block size and suitable confidence and texture thresholds for target applications. Additionally, the advantages of the use of a sparse Census transform will be presented. To find the most suitable parameters, the terms of matching quality and processing time play the key role. As reference for the matching quality, 31 datasets from the Middlebury stereo evaluation website ([29, 30, 31, 32]) are used. More details about that can be found in Section 6.2. These datasets are captured with low noise and high resolution cameras that allow the high quality of the ground truth images. Because this is not close to the expected environment in the target application, the datasets are additionally overlayed with random noise. For the proposed experiments the matching quality is always analyzed twice, once with the original and once with the noisy datasets. Figure 8 shows one example dataset with its original left image, the noisy one and its ground truth. As evaluation criterion for the matching quality, the average percentage of the true positives (tp) over all 31 datasets is used on the one hand. This rates the accuracy of the results. On the other hand, the average percentage of the correct matched pixels (total) in relation to the total number of pixels (with available ground truth values) in the image is used. This rates the density of the resulting disparity maps, or in other words it specifies how many pixels of the whole image are matched correctly. Due to the use of subpixel refinement, an error threshold of 0.5 pixels is used. In this work a squared window with the anchor point in the middle is used for the Census mask size as well as for the aggregation block size. In the following charts the sizes give the side length of the square.

The first parameter to analyze is the Census mask size. Figure 9 shows the matching quality for increasing mask sizes without costs aggregation, hence the attention is completely paid to the Census transform. As can be seen, evaluating

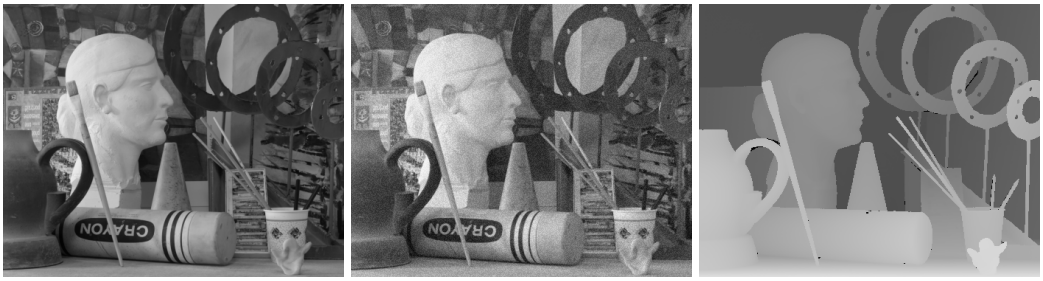
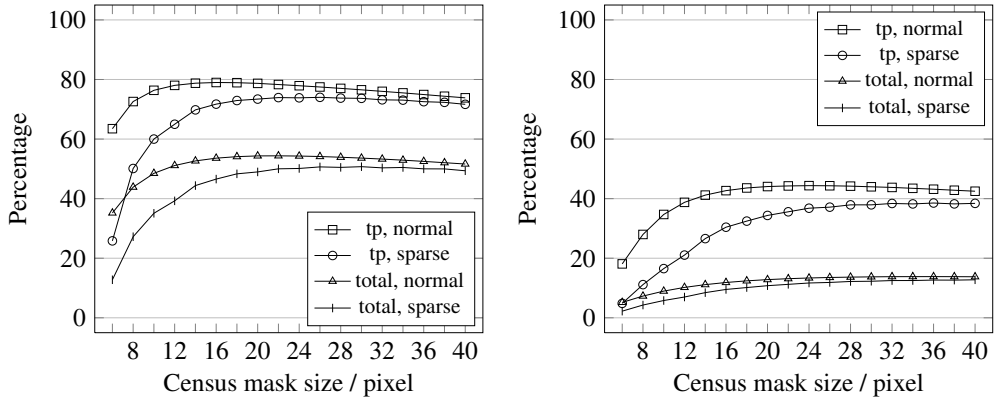


Figure 8: Middlebury dataset Art, from Scharstein and Pal [31]. Left image, noisy left image, and ground truth.

the original datasets results in a maximum at a size of 16×16. The noisy ones show that difficult scenes match better with larger mask sizes, whereas 16×16 also performs well. Both evaluations exhibit the fact that very large Census masks, 24×24 and above, decrease the matching quality. This can be traced back to the fact that large Census masks broaden the object borders. As a reminder, large Census masks mean large bit strings in the Census-transformed images. This means high computational effort during costs calculation as can be seen in Fig. 10 where the processing times of normal and sparse Census transforms are compared. A plain software solution is used for this comparison, but very similar results are expected for the other platforms.



(a) original (b) noisy

Figure 9: Matching quality for different Census mask sizes without costs aggregation.

After finding a proper Census mask, the aggregation block size has to be analyzed. A well-known problem concerning aggregation block sizes are disparity

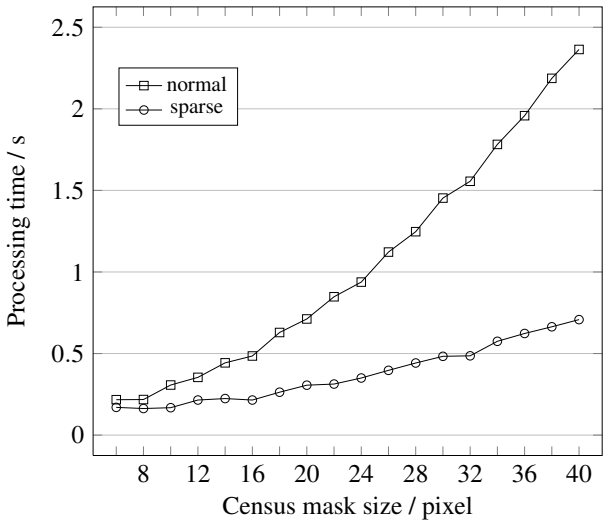


Figure 10: Processing time for different normal and sparse Census mask sizes. The times are measured using the plain software version with optimized Hamming distance calculation.

discontinuities as explained in Section 2.1. In Fig. 12, the matching quality for increasing block sizes at a Census mask size of 16×16 is shown. The chart of the original datasets clearly shows that the discontinuity problem decreases the matching quality from a block size of 5×5. The noisy datasets prove that large blocks increase the matching quality in difficult scenes for instance. Another very important consequence of costs aggregation is that it closes the conspicuous matching quality gap (Fig. 9) between normal and sparse Census transform.

Figure 11 shows the true positives for the noisy datasets of sparse and normal Census masks. The mask sizes are chosen to produce the same bit string weights to reach nearly the same computational effort. It shows that larger sparse Census masks perform better than small normal masks, which in turn justifies their usage.

Since the noisy scenes represent a worst case scenario and the cameras used in the application are expected to be much better, a block size of 5×5 is assumed to fit well.

After analyzing Census mask and aggregation block size separately, a summarizing evaluation of more combinations of both is given. As can also be seen in Fig. 9, the noisy datasets in Fig. 13 show that large Census mask and aggregation block sizes improve the matching quality for images of poor quality. In contrary, if high quality images are given, the evaluation of the original datasets shows that the matching quality profits from 5×5 aggregation at Census mask

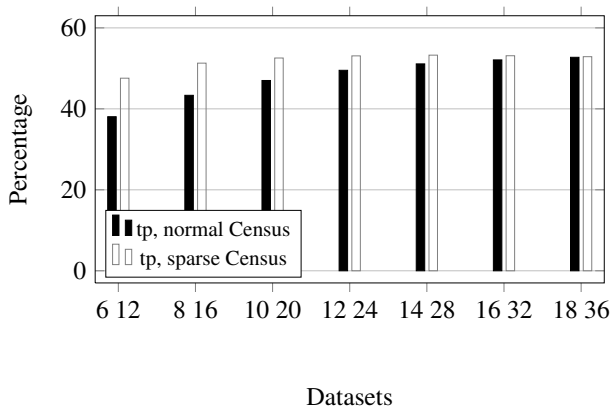
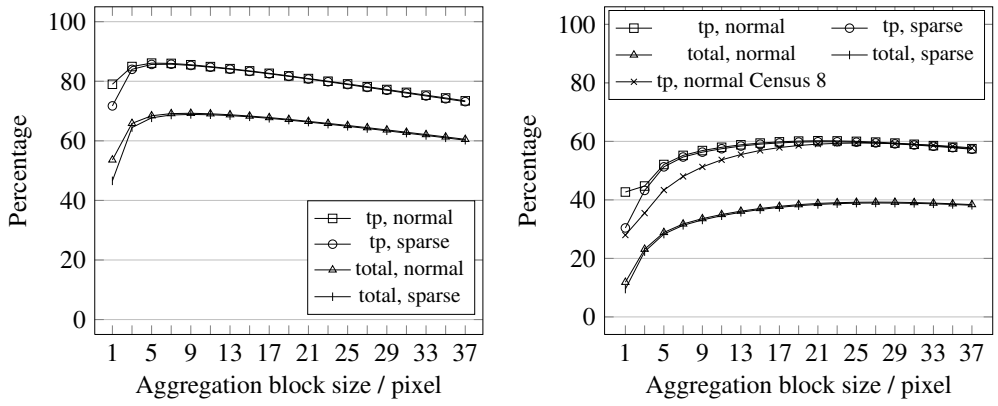


Figure 11: Matching quality of sparse Census versus normal Census masks with the same bit string weights.

sizes beyond 10×10. In the real-world application, rather good image quality but difficult scenes are expected, so a compromise of large Census mask, 16×16, and a relative small aggregation size of 5×5 is used.

Finally, the matching quality on disparity discontinuities and object borders is analyzed for different Census mask and aggregation block sizes. As reference, the Middlebury database provides four datasets with disparity discontinuity masks. These datasets are also used for the ranking described later in the evaluation section. Thus, only pixels on disparity discontinuities are evaluated here. The error threshold is set to 1 because the Tsukuba dataset has no subpixel accuracy and one image of four would influence the results distinctly. Furthermore, neither confidence nor texture threshold is used. As can be seen in Fig. 14, the Census mask as well as the aggregation block size deliver better results, the smaller they are. This is a good argument for small blocks and masks, but an aggregation block of 5×5 is also a good compromise.

The last two parameters are the thresholds for confidence and texture. Because the values have to be adapted for each camera head and the operating environment, they can be changed at runtime. Figure 15 shows the effect on the matching quality with respect to increasing confidence values for a 16×16 sparse Census transform with 5×5 aggregation. The ideal curves would be increasing true positives and constant total matches, which would mean that only false positive matches are eliminated. As can be seen in Fig. 15, the true positives actually increase, but the total matches decrease as well. This means that false as well as true positives are eliminated. For the original and the noisy datasets a confidence threshold of about



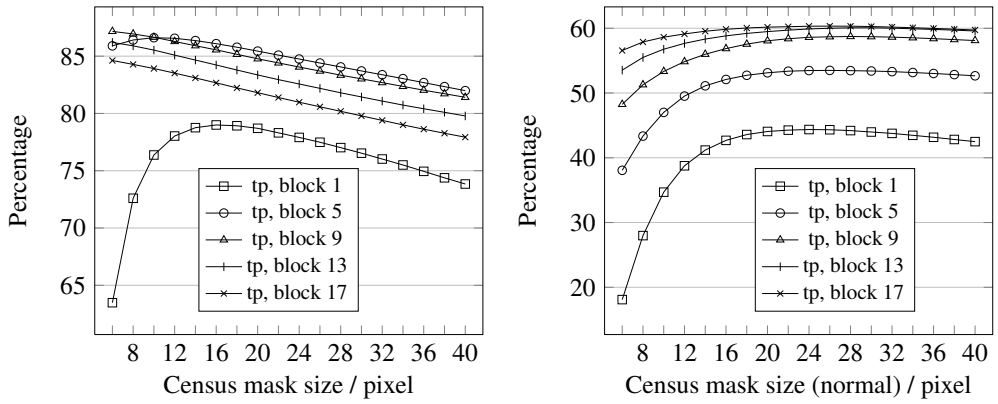
(a) original (b) noisy

Figure 12: Matching quality for different aggregation block sizes and a 16×16 sparse Census transform.

35 would be a good compromise between increasing true positives by filtering wrong matches and losing correct matches. Especially the noisy datasets show that a well chosen confidence value helps to increase the reliability of the matches more than it harms the results.

Finally, the texture threshold has to be adjusted. As explained in Section 3, it filters textureless areas where correlation-based, locally optimizing, matching algorithms fail in most cases. This parameter can also be used to adjust the algorithm to the noise characteristics of the image sensors. This is very useful in textureless dark areas for example, where the cameras' noise produces random textures, the matching fails and the confidence threshold is insufficient. As can be seen in Fig. 16, the original datasets are well textured so the texture threshold must be set very low to avoid losing too many true positives. The simulated noise in the noisy datasets is unfortunately too strong, so the texture threshold does not truly improve the results.

This section provided a detailed analysis of various parameters and their influence. It was illustrated that especially the confidence and texture threshold are strongly dependant on the application and camera. For the Census mask and the aggregation block size, a well chosen compromise between processing time and high quality matching was given.



(a) original (b) noisy

Figure 13: Matching quality for different Census mask and aggregation block sizes.

5. Reference Implementations

In this section, four reference implementations of the proposed stereo matching algorithm are introduced. First, a plain software solution for common central processing units is presented, then an optimized software version, followed by an implementation on two NVIDIA graphics processing units and finally a version for a Texas Instruments digital signal processor. Emphasis is placed on the GPU and DSP implementations because of their importance in embedded systems. All the implementations have in common that the calibration is performed with the OpenCV library [8] and the undistortion and rectification are realized by calculating the transform maps offline and applying a remap to the images online. Also, all implementations are flexible in terms of image dimensions and disparity levels. A performance comparison can be found in Section 6.3.

5.1. Plain Software

The first reference implementation is a plain software solution written in C/C++. The main idea of this implementation was the development of functional behavior of the proposed algorithm. It is embedded in a Microsoft Foundation Class (MFC) graphical user interface and uses the OpenCV library as its image processing basis. The strength of this implementation is the flexibility in Census mask and aggregation block size. The only optimized routine is the Hamming distance calculation, which is the counting of the set bits after calculating the xor product of two Census-transformed pixel values. The counting is done with the $\mathcal{O}(\log n)$ population count algorithm from AMD's *Software Optimization Guide*

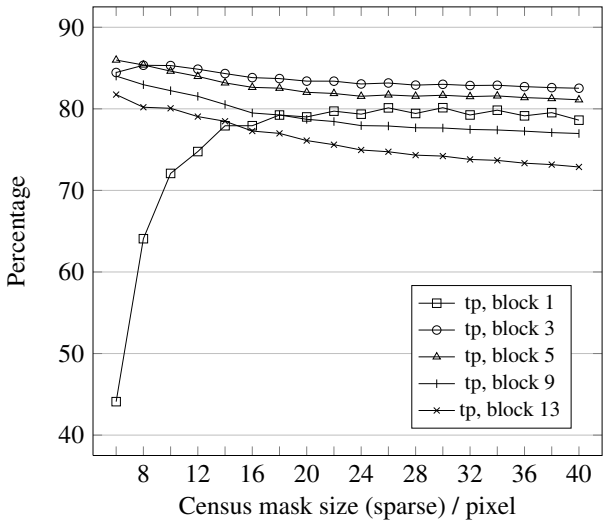


Figure 14: Matching quality on disparity discontinuities for different Census mask and aggregation block sizes.

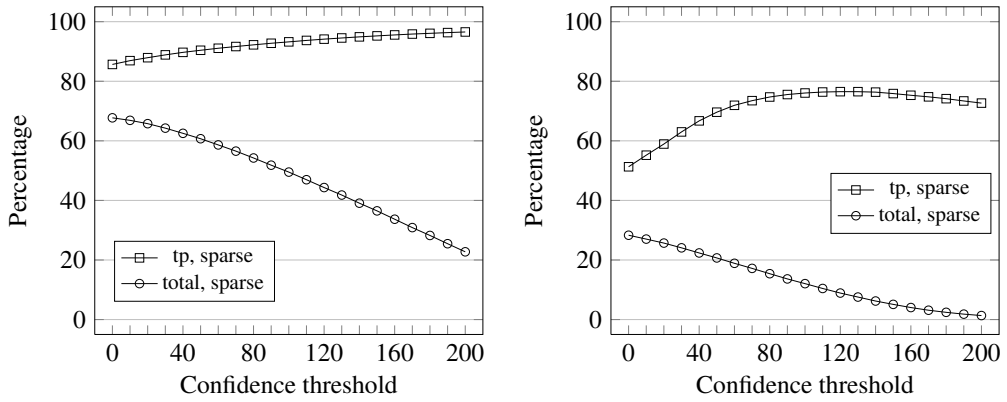
for AMD64 Processors (AMD [54]). This implementation was also used for the evaluations in Section 4.

5.2. Optimized Software

This implementation is based on the plain software version and is performance-optimized for standard PC hardware without using any graphics card acceleration but with extensive use of the Streaming SIMD Extensions (SSE) and the multi-core architectures of state-of-the-art PC CPUs.

The main target platform for this performance-optimized implementation is an Intel Mobile Core 2 Duo processor (model T7200) clocked at 2 GHz (Intel [56]). This CPU model is commonly used not only in notebook PCs, but also in industrial and so-called "embedded" PCs. A detailed description of this implementation can be found in Zinner et al. [58]. The major optimization topics are:

Sparse Census Transform. To optimize the Census transform, the SSE provides the `_mm_cmplt_epi8` instruction, which compares 16 pairs of signed 8-bit values at once. The drawback is that it cannot be used directly because the image pixel intensities are unsigned numbers. Pixel values greater than 127 would be interpreted as negative, leading to a incorrect result. To overcome this, 128 has to be added to each pixel value before using the SSE instruction.



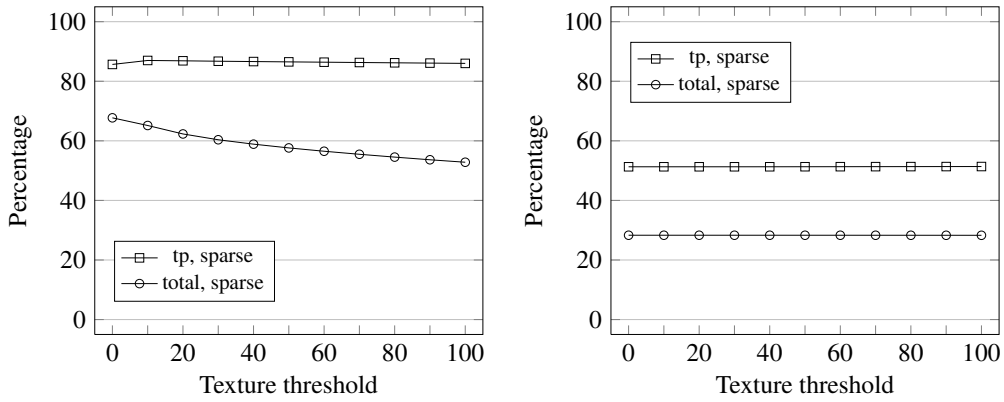
(a) original (b) noisy

Figure 15: Matching quality for different confidence threshold values with 16×16 Census transform and 5×5 aggregation.

Hamming Distance. The Hamming distance calculation is an important part to optimize because it is executed most often ($width \times height \times disps$). The calculation method was inspired from the BitMagic library (BitMagic [57]), which is similar to the one from AMD. A simple loop counting of set bits after computing the xor product of two 256-bit strings requires over 1100 CPU clock cycles. With this optimized version, the same calculation can be done in only 64 cycles.

DSI and Aggregation. The plain software implementation first calculates the whole DSI, then aggregates and selects the final disparity afterwards. Due to the huge amount of memory required, this approach has to be optimized so that it can also be used for embedded systems. The chosen method is to process the stereo image pair line-by-line. The standard DSI (Fig. 4) is transformed such that one layer represents one image line with all possible matches, so only a number of layers equal to the aggregation mask size has to be stored for one line. The transformed DSI is shown in Fig. 17 and the same method is used in the DSP implementation in Section 5.4 because of its memory awareness.

Processing on Multiple Cores. The stereo algorithm is well suited for parallelization for multiple CPU cores. By using the OpenMP capabilities of modern compilers, it was possible to achieve a speedup factor of almost 1.9 when using both cores of the T7200 processor.



(a) original (b) noisy

Figure 16: Matching quality for different texture threshold values with 16×16 sparse Census transform and 5×5 aggregation.

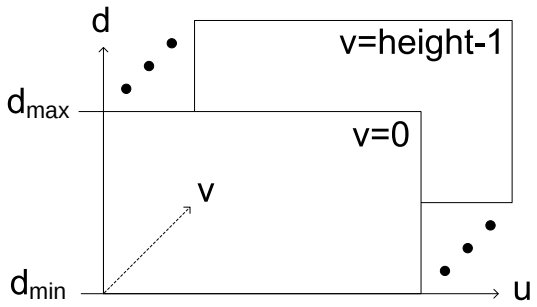


Figure 17: The transformed DSI for line-by-line processing.

5.3. Graphics Processing Unit

The GPU can be used with languages such as Cg, HLSL and OpenGL as well as with GPU programming libraries such as Brook, AMD/ATI's Close To Metal (CTM), now called Stream SDK, and NVIDIA's Compute Unified Device Architecture (CUDA). An overview is given in Houston [49].

For this work, CUDA was chosen for the implementation of the GPU-specific part. CUDA provides an interface based on standard C with only a few additional keywords. It abstracts the underlying hardware and does not necessitate knowing in-depth details about programming the hardware itself.

5.3.1. Hardware

Two different graphic card generations were used in this work. The first one is the GeForce 9800 GT (NVIDIA [52]), which is based upon the G92 chip generation. The second one is the GeForce GTX 280 (NVIDIA [50]) based on the latest chip generation, GT200. Since the CUDA Toolkit only supports NVIDIA graphic cards, only these where used as compared in Table 2.

Table 2: Graphic cards used.

Specifications	**GeForce 9800 GT**	**GeForce GTX 280**
Multiprocessors	14	30
Processor Cores[1]	112	240
Processor Clock (MHz)	1500	1296
Registers	8192	16384
Peak GFLOPS	504	933
Memory (MB)	1024	1024
Memory Clock (MHz)	900	1107
Memory Interface Width (Bit)	256	512
Peak Memory Bandwidth (GB/s)	57.6	141.7
Compute Capability	v1.1	v1.3

[1]Each multiprocessor contains 8 cores.

On devices with compute capability 1.2 and higher, the memory access on the GPU is less restrictive than on cards with a lower compute capability. The newer GTX 280 card is clocked at 1.295 GHz, compared to 1.5 GHz of the older 9800 GT. Nevertheless, the slower clock compared to the 9800 GT is over compensated by more than double the number of processors on the card as well as a 2.4 times greater memory bandwidth.

5.3.2. Overall Strategies

The bandwidth between the host and the graphics card via the PCI-Express bus (PCISIG [53]) is quite low compared to the graphic cards memory bandwidth as can be seen in Fig. 18. Hence, data transfer between GPU and CPU is a major bottleneck.

Due to that fact, one goal is to do as much work as possible on the GPU, rather than on the CPU, to minimize the transfers via the PCI-Express bus.

Table 3 shows different types of available memory on the GPU. Apparently registers and shared memory are well suited for fast access, but unfortunately their size is very small. The images and intermediate results usually reside in

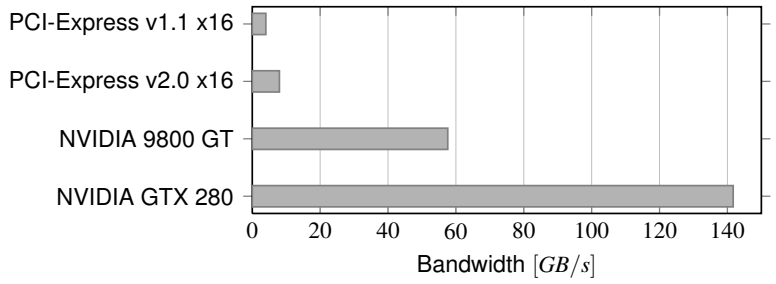


Figure 18: GPU memory bandwidth versus PCI-Express bus bandwidth

global memory. Access to global memory can also be very fast, as long as adjacent memory addresses are accessed by the threads. This way, memory access coalesces and results in one 64-byte or 128-byte memory transaction. Using the texture memory can also be advantageous. It provides a texture cache that is optimized for 2D spatial locality. Thus, addresses that are close together are read from the cache. The texture memory also supports clipping for addresses which are outside of $[0,N)$. Indices below 0 are set to 0 and values greater or equal to N are set to $N-1$.

Table 3: Available memory on the GPU using CUDA

Memory	**Scope**	**Access**	**Latency**[1]	**Cached**[2]	**Persistent**
Global Memory	global	read + write	400-600	no	yes
Constant Memory	global	read	400-600	yes	yes
Texture Memory	global	read	400-600	yes	yes
Shared Memory	block	read + write	4	no	no
Local Memory	thread	read + write	400-600	no	no
Registers	thread	read + write	0	no	no

[1] In clock cycles.
[2] Upon a cache-hit so the access is as fast as a register access.

Unlike the iterative processing on the CPU, where the image data are processed column-by-column and row-by-row by one thread, on the GPU one separate thread is usually used for each data element. This way, the execution occurs almost in parallel, which promises high throughput. The image data are partitioned into multiple blocks and each block is processed independently by an individual multiprocessor. To avoid access to the slow global memory, each block is loaded into the on-chip shared memory once. In case surrounding data are re-

quired, they are also transferred into shared memory as shown in Fig. 19. Within the shared memory, read and write operations can take place with almost no latency. After all the processing steps are complete, the result is written back into the persistent global memory.

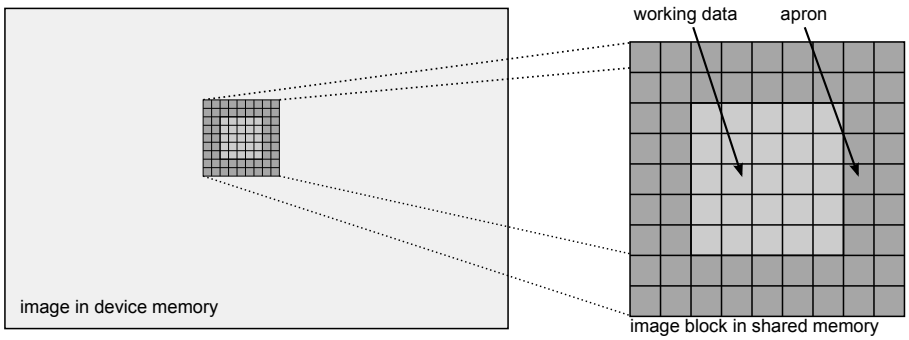


Figure 19: Image data are loaded block-wise with a surrounding apron.

The minimum parallel computing unit is a warp which contains 32 threads. The individual threads within one warp start together at the same program address but can execute and branch independently. A warp executes most efficiently when all of its 32 threads have the same execution path. As threads within one warp may diverge due to a data-dependent conditional branch, the warp executes each branch serially taken. When all the branches finish, the threads converge back to the same execution path. To obtain high performance it is therefore necessary to minimize the number of different branches. As controlling structures such as `if`, `switch` and loops may lead to different branch paths, such structures should be avoided wherever possible. Although the compiler may optimize the code to avoid different branches, complex code must be optimized manually.

5.3.3. Algorithm Implementation

Left/right consistency check, confidence and texture calculation, confidence and texture thresholding, and 3D reconstruction are fairly straightforward. They are adopted from the existing CPU implementation by removing the iteration over the image and instead launching a single thread for each pixel.

Rectification & Undistortion. As the translation map for undistortion and rectification remains the same for each image pair, it is calculated once and copied into a 2D texture memory on the GPU for fast access. In addition to the clipping of (u, v) coordinates which are out of bounds, the texture memory offers hardware support for bilinear interpolation.

Sparse Census Transform. Even though the image pairs are stored in the texture memory and memory access is cached, the data are loaded block-wise, including an apron into the shared memory as can be seen in Fig. 19, to speed up subsequent reads. This provides a favorable effect regarding the runtime as each pixel value is read 65 times in total.

DSI Calculation. The Hamming distance is also calculated using the $\mathcal{O}(\log n)$ population count algorithm from AMD's *Software Optimization Guide for AMD64 Processors* (AMD [54]). This algorithm performs its calculation on a 32-bit integer within only 12 operations. For parallel calculation, the GPU starts a thread for each pixel. In detail, it starts $\left\lceil \frac{width}{128} \right\rceil * 128 * height$ threads. If 128 is not an integer divisor of the image width, the GPU starts a few dummy threads, but only with this drawback can the fast shared memory be used efficiently. Each thread evaluates (18) for its pixel.

Cost Aggregation. Cost aggregation can be easily implemented using convolution, but it is the most extensive part for the GPU because the GPU cannot efficiently execute iterative algorithms such as moving box filters. For this reason, three different strategies were examined. In Gong et al. [55], six approaches are compared, among which the square-window approach performed best. The square-window can be implemented using a box filter (convolution) that can be horizontally and vertically separated and by using integral images [18]. Figure 20 shows the processing time with respect to different convolution mask sizes. As can be seen, integral images are independent from the mask sizes but are only profitable for very large masks. Due to the use of a 5×5 aggregation, the standard convolution is chosen because it performs best at this mask size.

Subpixel Refinement. Because of the relatively high data volume and because each value is read only once, the shared memory cannot be used in a reasonable manner. Furthermore, the execution partially depends on the DSI data. Therefore conditional branches in the code have to be accepted. Due to the data structure and the fact that the data access depends on the data itself, reads hardly coalesce, which in turn has a negative impact on data throughput. To accelerate reads within the global memory, a 1D texture is bound onto the memory. Contrary to 2D and 3D textures, a 1D texture can be used on global memory, but does not offer the same advantages. Nevertheless, the texture cache helps to speed up the unaligned read access.

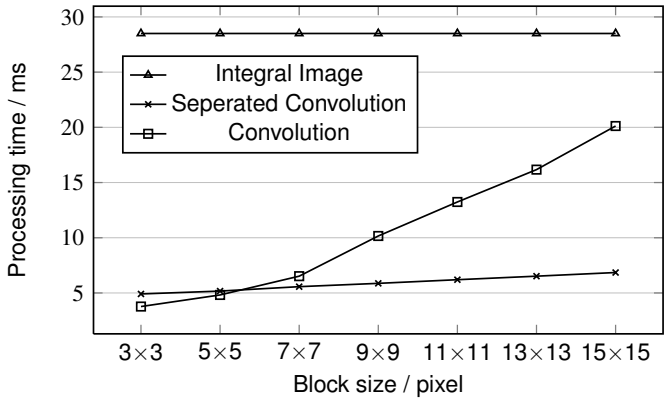


Figure 20: The computation time needed for the aggregation with different mask sizes using convolution, separated convolution, and integral images. The runtime was measured on a GTX 280 graphics card. For each computation 50 disparities with an 512×512 image were used.

5.4. Digital Signal Processor

The TMS320C64x family from Texas Instruments contains various high-end DSP models with clock frequencies of up to 1.2 GHz and single core peak performances of 9600 MIPS. With power ratings below 2 W (DSP only), these processors make very small and energy-efficient embedded systems possible. The market offers various industrial smart cameras that are equipped with such DSPs. Cost-efficient stereo vision systems could be realized either by combining two smart cameras or by using a smart camera that features two imaging sensors. Currently, the absence of fast and reliable high-quality software stereo engines for DSPs is the main hurdle in realizing such systems. This is the key motivation for the DSP reference implementation of the proposed stereo matching algorithm.

5.4.1. Platform Characteristics

The primary DSP platform for the reference implementation is a C6416 fixed-point DSP at 1GHz, details of which can be found in TI [62]. Minor adaption steps will follow to enable the use of processors with the enhanced C64+ core architecture, up to the recently announced C6474 multicore DSP (TI [60]) with up to three times greater performance compared to single core DSPs.

The very good ratio between computation speed and power consumption of this platform is gained by several architectural characteristics that are significantly different from PC CPUs for instance. Due to its Very Long Instruction Word (VLIW) architecture, the DSP features an instruction level parallelism employing eight functional units that can operate in parallel. The TMS320C64x has

no floating point unit. Floating point operations have to be emulated in software. Performance-critical programs must avoid floating point operations as much as possible. This processor lacks instructions for fast integer division. A 32-bit division takes up to 42 machine cycles. Thus, divisions must also be avoided in critical inner loops. DSP machine registers are 32 bits wide, which is another handicap compared to the 128-bit SSE registers on recent PC CPUs. The DSPs have less on-chip memory for fast access and/or data caching purposes.

These items have a severe influence on the manner of realizing a performance-optimized software implementation of the stereo algorithm that is able to exploit as many of the platform's capabilities as possible.

5.4.2. DSP Performance Optimization

Starting from ordinary ANSI-C code, the DSP platform offers an enormous potential for performance gain once several optimization techniques have been applied. Although TI delivers very sophisticated optimizing Ccompilers, remains the fact that the programmer's skill and particular knowledge about processor architecture and compiler behavior continue to exert significant influence on the resulting performance.

For this implementation a special embedded performance primitives library, the PfeLib, was used. In addition to a basic set of optimized routines, it also provides a framework for optimizing new low-level image processing functions, including a test environment that enables thorough simulator-based performance analysis and optimizations. The principles of the PfeLib are presented in Zinner et al. [61].

Almost all the subfunctions have been optimized at the hardware level by using compiler intrinsics that allow for explicit access to certain machine instructions. In an initial stage, the optimizations were made for each function in an isolated test environment using only on-chip memory. The goal was to maximize data throughput and thus minimize the required processor cycles per pixel (cpp) for each function. Algorithmic correctness was verified against a generic ANSI-C version of the function.

After a brief discussion of the most important functions, the resulting speedups compared to the generic version are shown in Fig. 21.

Census Transform. The `_cmpgtu4()` intrinsic is able to perform four 8-bit comparisons within a single instruction. This enables quite a fast implementation of the Census transform. For the 16×16 sparse Census transform, which requires 64 comparisons per pixel, a final performance value of 18 cpp was achieved.

Hamming Distance. The DSP offers an instruction for counting the set bits of four 8-bit values, which can be accessed via the `_bitc4()` intrinsic. Counting the set bits of a 64-bit word thus requires two `_bitc4()` instructions which deliver 8 partial bit counts that have to be summed together to the final Hamming distance. The optimized version accomplishes all this, including the loading of two 64-bit operands and the writing of one 16-bit result, at an average expense of less than 2.5 processor cycles.

Aggregation. A dedicated 16-bit 5×5 sum filter function was implemented. In addition to using various intrinsics, the inner loop was extended to iterate over chunks of 8 pixels, which results in better utilization of the DSP units. This has been shown to be the fastest way of realizing a cost aggregation rather than, for example, using integral images. The optimized version achieves a filtering speed of 2.71 cpp.

WTA - Minimum Search. For the purpose of saving memory bandwidth, searching for the minimum cost values is combined with the subpixel interpolation and the calculation of the confidence values within a single function. The minimum search is optimized in such a way that four columns of cost values in the DSI are scanned in parallel by using intrinsics such as `_cmpgt2()` and `_min2()`, which perform 16-bit comparisons and minimum operations, respectively. The optimizations resulted in a performance enhancement from 10 to 1.9 cycles per evaluated cost value.

Subpixel Refinement. The refinement is done by evaluating (20). Conversion into the fixed point domain achieved a remarkable increase in speed, but the division operation is still processor-intensive. This integer division was then substituted by an approximation method applying Newton's method. A first estimate is gained by using the `_norm()` intrinsic and then three of Newton's iterations are applied to achieve sufficient accuracy.

Confidence Calculation. An evaluation of (25) also requires a division. As the denominator y_{max} is constant during the program run, the division is replaced by a multiplication by the reciprocal value.

5.4.3. DSP Memory Management

The memory hierarchy of the TMS32C64x DSPs has several stages, namely the fast L1 Caches and the additional on-chip memory (IRAM). Further external memory (SDRAM or DDR-RAM) already have much slower access times and

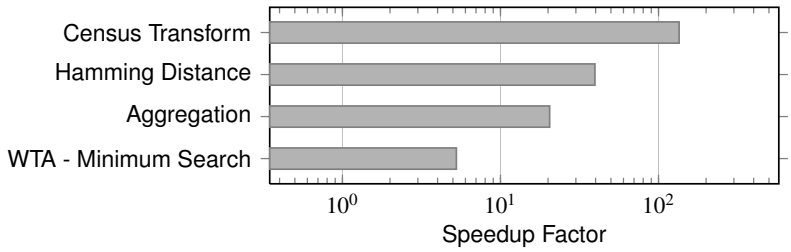


Figure 21: DSP low-level function performance optimization speedups

bandwidths. IRAM is usually a very limited resource; the C6416 DSP has 1 MiB of IRAM and other models offer even less. Thus, large amounts of image data have to be stored in ERAM. Although a portion of the on-chip memory can be configured to serve as L2-cache for the ERAM, performance can still be much worse compared to keeping all the data in IRAM.

A remedy for this problem is using a DMA double data buffering scheme, which is also part of the PfeLib. The method is called ROS-DMA and is described in detail in Zinner and Kubinger [59]. Fig. 22 is a case study that uses the Census transform function within three different memory configurations. IRAM means that any image data reside in fast on-chip memory, which is the optimal setting, hence this configuration performs best. ERAM + L2 Cache is a configuration with image data in external memory and activated L2 cache. The function now takes more time. In the third configuration, data still reside in ERAM, but the ROS-DMA method is used instead of L2 cache. This results in a relatively small performance loss compared to IRAM.

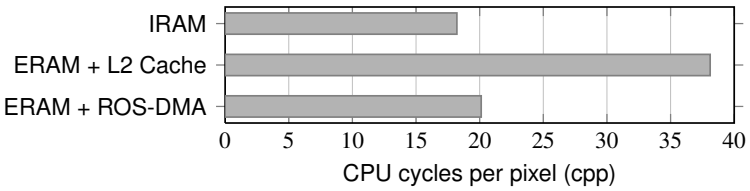


Figure 22: Impact of different memory configurations on the performance of the Census transform

5.4.4. Overall Performance

The generic ANSI-C version of the algorithm took 2.1 s on a 1 GHz C6416 DSP for one stereo pair of 450×375 pixels for 60 disparities. The overall performance of the optimized DSP implementation now achieves 129.18 ms, which is a

speedup factor of 16.26.

6. Evaluation

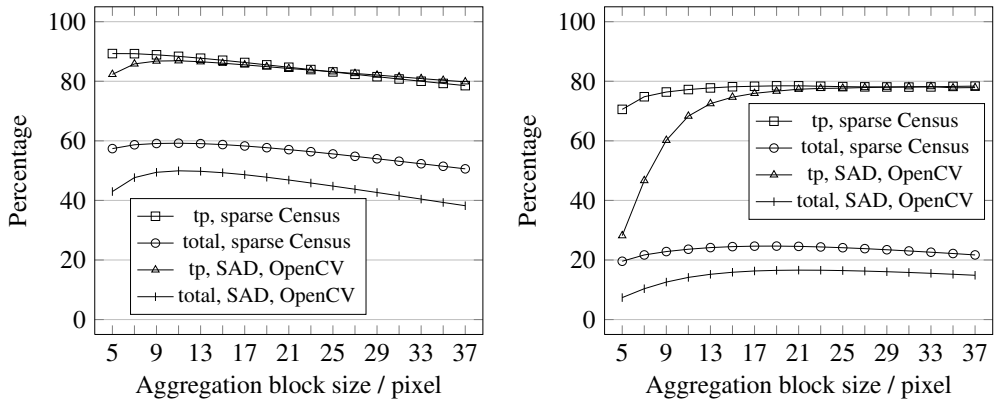
This section gives a detailed evaluation of the proposed algorithm in terms of results, quality and processing time. First, the resulting disparity maps are evaluated by comparing the results of indoor, outdoor and Middlebury scenes from Section 4 with a standard SAD algorithm. Afterwards, the rank on the Middlebury stereo evaluation website by Scharstein and Szeliski [29] is given. Finally, the processing times of the proposed reference implementations are compared and analyzed.

6.1. Algorithm Comparison

The chosen algorithm for results evaluation is the SAD blockmatching algorithm from Konolige published in the OpenCV library [6]. For ground truth comparison, the 31 datasets from Middlebury, in the original and the noisy version, are also used. Figure 23 shows the true positives (tp) and the total matches (total) of the proposed algorithm in comparison to the SAD with an error threshold of 0.5. All parameters are constant over all datasets. The confidence threshold is 35 in the original and 50 in the noisy datasets. The texture threshold is not used in this evaluation. The SAD also has a few post-processing parameters to adjust. The uniqueness parameter is set to 40 in the original and 20 in the noisy datasets. The texture filter is also not used. The confidence threshold and the uniqueness parameter are always set in a way that the true positives are maximized without losing to many good matches as shown in the charts in Section 4.

Unfortunately, the SAD algorithm produces a black border of the width of the maximum disparity. One strength of the algorithm proposed in this paper is the use of the maximum possible matching range. Nevertheless, to provide a completely fair comparison, the black border produced by the SAD is artificially added in the results of the Census algorithm. This border can be seen in Fig. 25 on image (e) and (f). As can be seen in both charts, the proposed algorithm performs significantly better for small aggregation block sizes and nearly equal for large sizes in terms of true positives. An important improvement is the significant higher percentage of total matches, which means a higher density of the disparity maps.

An advantage of the Census transform is the robustness on disparity discontinuities because of a good outlier tolerance as described by Woodfill et al. [38]. Figure 24 shows that the matching quality at object borders is clearly better than

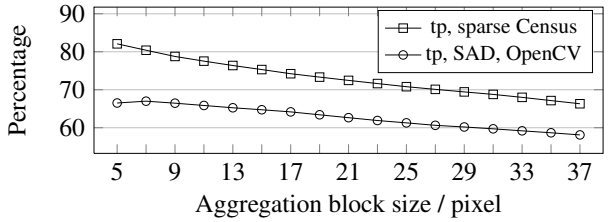


(a) original (b) noisy

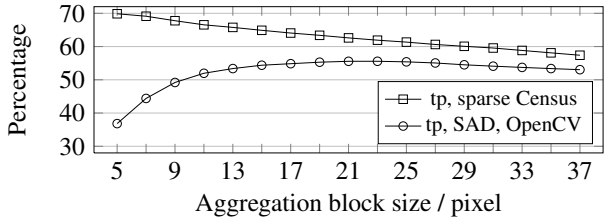
Figure 23: Matching quality comparison between the 16×16 sparse Census transform and the SAD algorithm.

using SAD. The experiments with the original images show that obviously the percentage of the true positives shrinks with increasing block size for both algorithms due to the disparity discontinuity effect which causes object borders to become broader. This is also true for the noisy images when using the Census matching, but the SAD has different characteristics. Due to the noise, small block sizes deliver rather little true positives. Only after a block size of about 23×23 does the disparity discontinuity effect become active and the number of true positives begin to decrease. As in Section 4, the error threshold is set to 1 for this analysis and no post-processing is done.

In real-world environments, it can easily happen that the left and the right camera images suffer from different illuminations. Hirschmueller and Scharstein [32] evaluated many cost functions in terms of different illumination in their work discovered that the rank transform works best of all the correlation methods. Due to the fact that the Census transform, as well as the rank transform, is based on local pixel intensity differences, which seem to be independent to constant gain and brightness differences in the stereo pair, similar results for the Census transform are expected. Figure 25 shows the results of the proposed algorithm and the SAD for five different Middlebury datasets in true positives and total matches charts and the Art dataset for visual comparison. The input stereo pair is also given, where the illumination differences can be seen clearly. The Census matching delivers in all cases more true positives than the SAD. A considerable difference can be seen in the percentage of the total matches, where the Census transform clearly



(a) original



(b) noisy

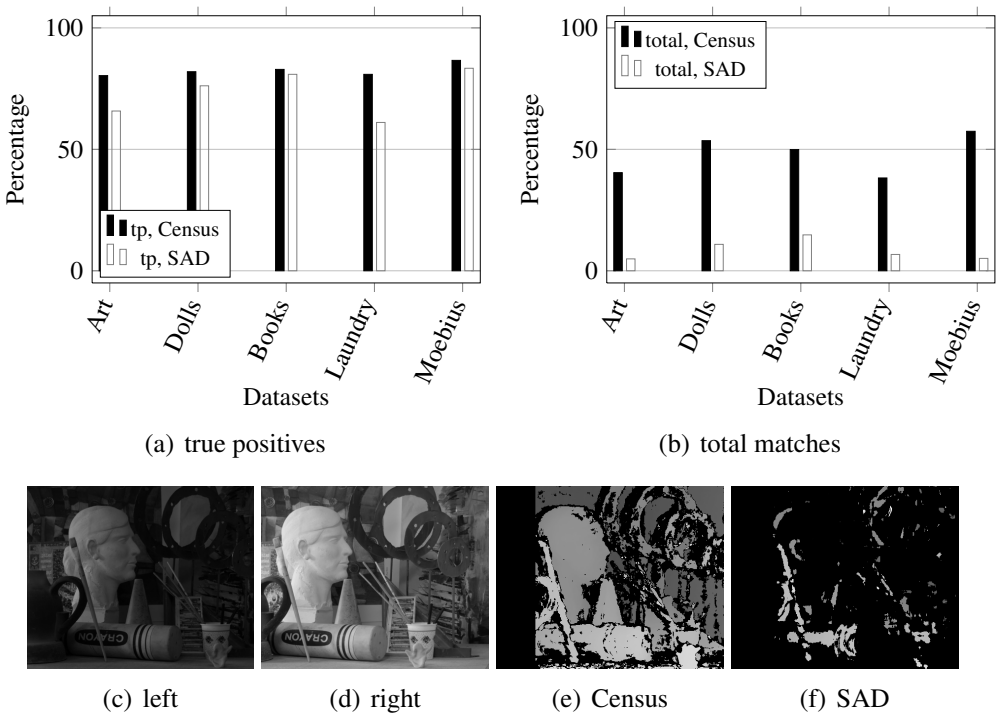
Figure 24: Matching quality comparison between the 16×16 sparse Census transform and the SAD on disparity discontinuities.

outperforms the SAD.

For visual comparison of the matching quality, some real-world scenes are given in Fig. 26. The aggregation block size selection was performed with the charts in Fig. 23. For each algorithm the best matching configuration for the original scenes was chosen. The algorithm has a 16×16 Census mask, a 5×5 aggregation block size, a 40 confidence and 0 texture threshold. The SAD has on the one hand a block size of 11×11 and on the other of 21×21. The uniqueness parameter is set to 40. Additionally, the disparity maps of the sparse Census algorithm without confidence and texture thresholds are given. It can be seen that obviously false matched pixels are filtered out well and apparent true matches are kept valid.

6.2. Middlebury Evaluation

Scharstein and Szeliski [29] have developed an online evaluation platform, the Middlebury Stereo Evaluation [65], which provides a huge number of stereo image datasets consisting of the stereo image pair and the appropriate ground truth image. Four of these datasets, shown in Fig. 27, are used to evaluate area-based stereo matching algorithms and to compare the results with many others



(a) true positives (b) total matches

(c) left (d) right (e) Census (f) SAD

Figure 25: Matching quality comparison between the 16×16 sparse Census transform with 5×5 aggregation and the 11×11 SAD for scenes with illumination difference between the stereo pair.

online. Since this evaluation is very well-known and state-of-the-art, the proposed algorithm in this work is also evaluated in this manner.

To evaluate an algorithm on this website, disparity maps of all four datasets have to be generated and uploaded. The disparity maps have to correspond to the left stereo image and the disparities have to be scaled by a certain factor. The evaluation engine calculates the percentage of bad matched pixels (false positives), within a certain error threshold, by pixel-wise comparison with the ground truth image. This is done three times for each dataset. First for all pixels where a ground truth value is available; second, for all non-occluded pixels; and third, for all pixels at disparity discontinuities. Many stereo algorithm developers, approximately 74 entries to date, use this platform for evaluation. This gives a significant overview of how the developed algorithm performs in comparison to other algorithms. The platform is up-to-date and constantly growing.

Figure 27 shows the four evaluation datasets and the resulting disparity maps for two different sparse Census configurations and the SAD from Section 6.1.

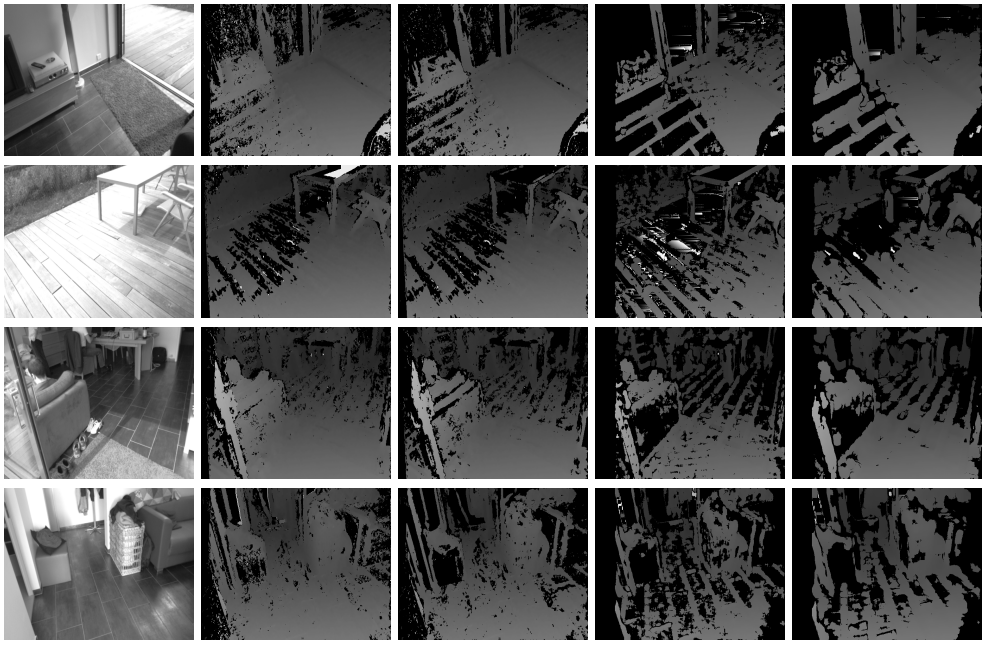


Figure 26: Real-world scenes. From left to right: Left stereo image, Census 16 (16×16 sparse Census mask and 5×5 aggregation without thresholds), Census 16t (16×16 sparse Census mask and 5×5 aggregation with confidence threshold), SAD 11t (SAD with 11×11 block size and uniqueness threshold 40), SAD 21t (SAD with 21×21 block size and uniqueness threshold 40).

Census 16 is the main configuration and another configuration, namely Census 10, is used for the Middlebury evaluation. The difference between them is the Census mask size of 10×10 and an aggregation block size of 3×3 in contrast to 16×16 Census and 5×5 aggregation. The reason for this is because Fig. 13 in Section 4 shows that smaller Census masks are well suited for high quality input images. For the SAD an aggregation of 11×11 is used. Additionally, a 9×9 median filter is applied as a post-processing step for Census 10 and SAD. These selected parameters are chosen in such a way as to achieve the best possible rank on the website. Finally, to fulfill the evaluation rules, the missing values in the disparity maps have to be extrapolated. Due to the smaller Census mask and aggregation block size of the proposed algorithm, and the simple post-processing, the processing time is not negatively influenced.

After a short study of the leading algorithms it can be seen that nearly none of them is developed for high frame rates. Many of them use global optimization techniques and focus on matching quality only. To include processing time in

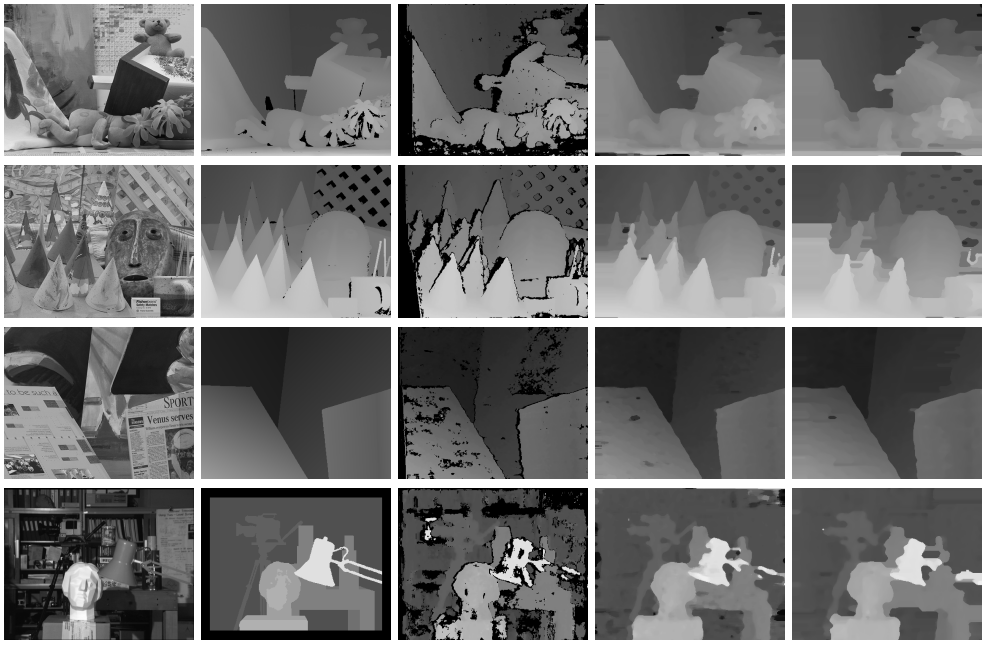


Figure 27: Middlebury stereo datasets [29, 30]. From left to right: Left stereo image, ground truth, Census 16 (16×16 sparse Census mask, 5×5 aggregation, confidence 40 and texture 0), Census 10 (10×10 sparse Census mask, 3×3 aggregation, confidence 40 and texture 0), SAD (11×11 block size, uniqueness 40 and texture 0).

the evaluation, only algorithms declared capable of real-time or near real-time, or at least faster than one second, will be mentioned here. As can be seen in Table 5 later on, the meaning of real-time and the corresponding processing time is interpreted differently by the authors of the algorithms. Also, the SAD algorithm from Section 6.1 is evaluated even if it is not in the permanent table. Of course, due to the fact that it produces the black border on the left side of the image, it suffers more from extrapolating. The processing times for published algorithms are taken from the literature, the SAD algorithm and the proposed algorithm are measured by the authors.

The main ranking of the algorithms published on the Middlebury stereo website is ordered by the average rank over all twelve bad matching percentage columns with an error threshold of 1 and is shown in Table 4. A visual comparison of the disparity maps is given in Fig. 28 and Fig. 29. As can be seen, only one of the real-time algorithms ranks among the top ten in performance. The others, including the proposed algorithm, rank in the middle and bottom positions. Additionally,

the average percentage of bad pixels over all twelve columns is given. This value is very meaningful and shows how close together the algorithms lie. The distance between the best real-time algorithm and the proposed algorithm is only 3.5 percent.

Interesting is the comparison of the processing times in Table 5. It was attempted to use the same input image sizes and disparity ranges for all the algorithms when possible. The proposed algorithm is currently the fastest, and with the small difference in the bad matches percentage it is thus considered to be a very good compromise between matching quality and processing time. As can be seen, in contrast to the others the presented algorithm reaches real-time capability not only on GPU, but also on DSP and PC.

Table 4: Middlebury main ranking (error threshold 1) with real-time algorithms only. The SAD is not included in the online table, so the average ranks in this version are not exactly the same. At the time of writing this paper, the main ranking consists of a total of 75 (SAD included) algorithms. All values, except the ranks, are given in percentages. The small numbers are the specific ranks of the columns. Avg. (%) is the average percent of bad pixels over all 12 columns.

Algorithm	**Avg.**	**Tsukuba**			**Venus**			**Teddy**			**Cones**			**Avg.**
	Rank	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	**(%)**
PlaneFitBP	18.0	0.97 7	1.83 19	5.26 7	0.17 10	0.51 15	1.71 6	6.65 19	12.1 22	14.7 10	4.17 34	10.7 35	10.6 32	5.78
RealtimeVar	37.9	3.33 55	5.48 60	16.8 64	1.15 47	2.35 52	12.8 58	5.88 11	7.25 5	14.9 12	4.61 38	6.59 3	12.9 50	7.85
RealtimeBP	40.0	1.49 29	3.40 43	7.87 35	0.77 38	1.90 49	9.00 49	8.72 48	13.2 29	17.2 30	4.61 29	11.6 45	12.4 46	7.69
RealtimeBFV	40.2	1.71 34	2.22 32	6.74 21	0.55 33	0.87 29	2.88 22	9.90 57	15.0 48	19.5 46	6.66 59	12.3 48	13.4 53	7.65
FastAggreg	43.5	1.16 11	2.11 30	6.06 14	4.03 67	4.75 66	6.43 41	9.04 49	15.2 50	20.2 51	5.37 51	12.6 51	11.9 41	8.24
OptimizedDP	46.5	1.97 39	3.78 48	9.80 45	3.33 65	4.74 65	13.0 59	6.53 18	13.9 38	16.6 22	5.17 48	13.7 57	13.4 54	8.83
Prop. Alg.	49.7	5.08 70	6.25 66	19.2 68	1.58 54	2.42 53	14.2 60	7.96 37	13.8 35	20.3 54	4.10 32	9.54 24	12.2 43	9.73
RTimeGPU	50.4	2.05 42	4.22 52	10.6 50	1.92 58	2.98 56	20.3 64	7.23 30	14.4 45	17.6 34	6.41 57	13.7 56	16.5 61	9.82
ReliaDP	53.2	1.36 22	3.39 42	7.25 30	2.35 60	3.48 62	12.2 55	9.82 55	16.9 59	19.5 47	12.9 72	19.9 71	19.7 63	10.7
TreeDP	56.2	1.99 41	2.84 39	9.96 47	1.41 52	2.10 50	7.74 47	15.9 68	23.9 69	27.1 68	10.0 66	18.3 66	18.9 62	11.7
BlockMatch	68.6	6.61 73	7.91 73	28.9 74	2.73 62	3.57 63	29.2 72	14.9 66	22.6 65	33.5 72	10.3 67	17.9 64	27.9 72	17.2

In Section 4 and Section 6.1 the error threshold is set to 0.5 because the algorithm delivers disparities in subpixel accuracy. The Middlebury ranking also supports subpixel error thresholds but the real-time algorithms calculate integer disparities only. Thus, a direct comparison would be unfair. To overcome this problem, Yang et al. [63] have introduced a post-processing step to enhance the resolution of range images to subpixel accuracy. They proved that it was suitable for all algorithms of the Middlebury ranking at the time of publication and published the results on their website[1].

The final evaluation step of the introduced algorithm is a subpixel comparison

[1] http://vis.uky.edu/ liiton/publications/super_resolution/

Table 5: Performance comparison of the declared real-time algorithms in the Middlebury main ranking. The frame rates and platforms are taken from the papers wherever they were mentioned. If different implementations are published, the fastest one is taken. Size denotes input image resolution and disparity search range. The proposed algorithm is given for all three implementations.

Algorithm	**Rank**	**Fps**	**Size**	**Platform**
PlaneFitBP [66]	11	1	512×384, 48	PC 3.2 GHz, GeForce 8800 GTX
RealtimeVar [73]	37	2.15	384×288, 16	PC 2.83 GHz
RealtimeBP [27]	35	16	320×240, 16	PC 3 GHz, GeForce 7900 GTX
RealtimeBFV [71]	41	57	384×288, 16	GeForce 8800 GTX
FastAggreg [67]	40	5	384×288, 16	Intel Core Duo 2.14 GHz
OptimizedDP [72]	53	5	384×288, 16	PC 1.8 GHz
Proposed Alg.	**47**	**573.7**	**320×240, 15**	**GeForce GTX 280**
Proposed Alg.	**47**	**62.9**	**320×240, 15**	**PC Intel Core2 Duo 2 GHz**
Proposed Alg.	**47**	**26.4**	**320×240, 15**	**DSP 1 GHz TI TMS320C6416**
RealTimeGPU [48]	49	43.48	320×240, 16	PC 3 GHZ, Radeon XL1800
ReliabilityDP [68]	51	23.8	384×288, 16	PC 3 GHz, Radeon 9800 XT
TreeDP [69]	52	1-5	Middlebury[1]	n/a
OpenCV [6]	61	66.67	384×288, 16	PC 3 GHz

[1] "It runs in a fraction of a second for the Middlebury images."

with the subpixel enhanced versions of the real-time algorithms. Unfortunately, only 4 of them are available up to now. The rankings on the mentioned website are out-of-date, so the ranking criterion is now the bad pixel percentage. The proposed algorithm is the leader with 14.34%, followed by RealtimeBP with 14.56%, followed by ReliabilityDP with 16.57%, followed by RealtimeGPU with 16.72% and at last TreeDP with 19.29%.

6.3. Processing Time

Table 6 gives a comparison of the processing times of the different implementations. The GPU implementation, with a considerable frame rate of 105.4 fps for the teddy dataset, is by far the fastest, followed by the optimized software with 12.89 fps and the DSP with 7.74 fps.

Figures 30, 31 and 32 show the performance of the implementations for different image sizes and disparity search ranges, given in frames per second (fps) and million disparity evaluations per second (Mde/s). Please note that commonly used image dimensions were chosen for the data points in the diagrams. Thus the pixel count does not increase linearly along the x-axis. Mde/s increase with increasing disparities in all three charts, which is as expected because some algorithm steps,

Table 6: Performance of the reference implementations. Image dimensions are 450×375 and disparity search range is 60. Subpixel refinement includes the confidence map calculation and thresholding includes texture and confidence. All values, except frame rate and Mde/s, are in ms.

Function	**Plain SW**	**Opt. SW**	**9800 GT**	**GTX 280**	**DSP**
Lens Undistortion + Rectification	34	2.2	0.15	0.05	5.4
Sparse Census Transform	168	8.7	0.89	0.52	8.97
DSI Calculation	332	22.61	4.76	2.42	33.08
Texture Map Calculation	29	4.89	0.51	0.24	4.02
Cost Aggregation	573	17.0	7.86	4.03	33.57
Subpixel Refinement	555	18.69	4.43	2.08	39.11
LR/RL Consistency Check	8	1.4	0.25	0.09	2.55
Thresholding	7	0.68	0.06	0.04	2.48
3D Reconstruction	32	1.41	0.03	0.02	N/A[1]
Total (ms)	**1738**	**77.58**	**18.94**	**9.49**	**129.18**
Total (fps)	**0.575**	**12.89**	**52.8**	**105.4**	**7.74**
Mde/s	**5.82**	**130.5**	**534.6**	**1067**	**78.38**

[1] 3D reconstruction is not yet implemented on the DSP

e.g. the rectification, have a constant complexity which is independent from the number of disparities. On the PC the Mde/s are relatively constant for different image dimensions. This means that the PC is able to deliver a quite constant memory bandwidth presumably due to its large data caches. On the GPU, the Mde/s clearly increase with increasing image dimensions. Processing larger images results in more thread blocks being launched. The thread scheduler on the GPU can then work more efficiently. On the DSP platform, the effects of the DMA buffering schemes and the behavior of the L1 data cache are too manifold to be able to identify a clear trend in the behavior of the Mde/s according to varying image dimensions. The DSP indeed delivers by far the most stable performance since processing times of consecutive frames are practically equal without any significant outliers. However, the given frame completion times of the PC and GPU implementations must be taken as average values over several frames because they may range across several percent. This is caused by the bad influences of large data caches and high level operating systems on the predictability of the worst case execution time on these platforms. Under this aspect of real time capability, only the DSP platform offers truly guaranteed maximum execution times.

6.4. Power Consumption

The target platform used for the power consumption measure of the optimized software implementation is a MacMini with an Intel Core 2 Duo clocked at 2 GHz. The NVIDIA GTX 280 GPU is used within an Intel Core 2 Quad system clocked at 2.4 GHz and equipped with 4GB RAM. For the DSP implementation a Texas Instruments DSP Starter Kit (6416DSK) is used. Table 7 shows the power consumption of the three real-time implementations. All measurements were carried out for the entire system respectively without cameras. Power consumption was measured in idle mode as well as during stereo processing.

Table 7: Power consumption of the reference implementations.

Platform	**Idle (W)**	**Processing (W)**	**Efficiency (Mde/J)**
Opt. software (MacMini)	13	57	2.29
GPU (Intel Q6600 2.4 GHz)	126	205	5.21
DSP (TI DSK)	3	5	15.68

An additional evaluation parameter, million disparity evaluations per Joule (Mde/J), is introduced to show the power efficiency in terms of disparity calculation of the different platforms. As can be seen, the DSP is most power efficient. It calculates 15.68 million disparity evaluations per Joule. The GPU, although it has the highest power consumption, is twice as efficient as the MacMini platform.

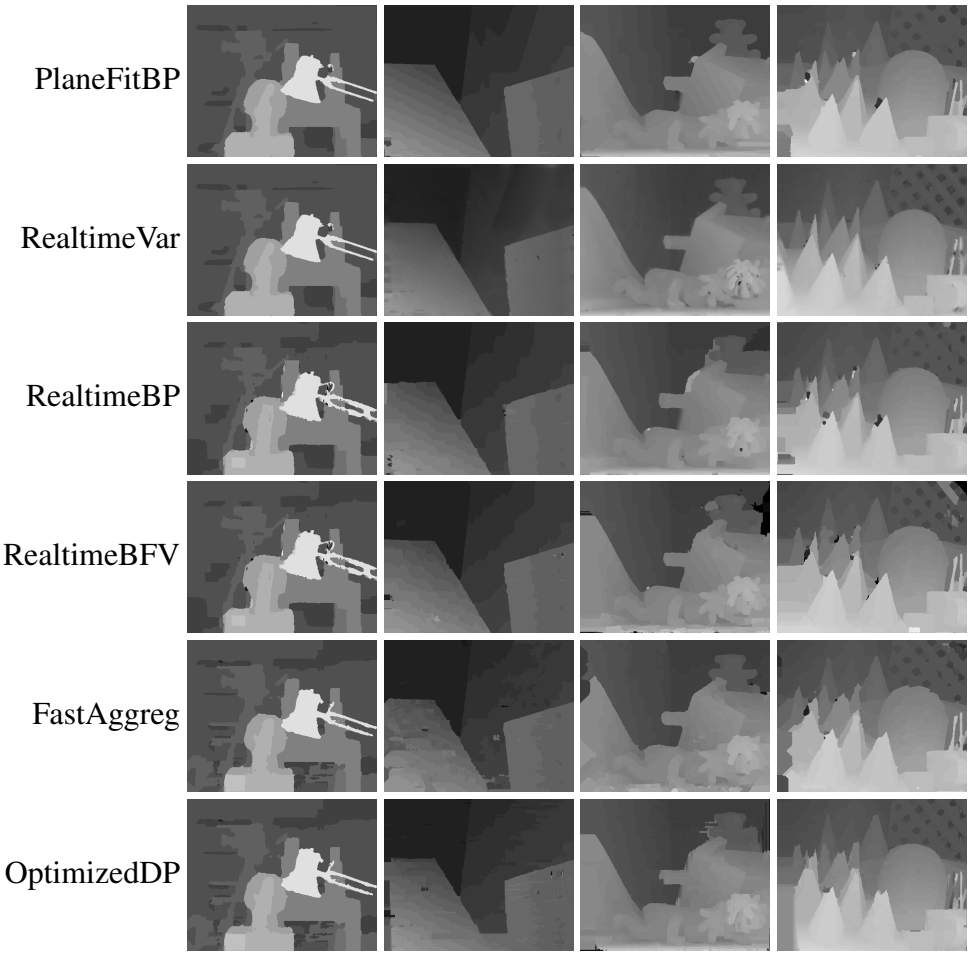


Figure 28: Visual comparison of the real-time algorithms on the Middlebury website, sorted by the main ranking in Tab. 4, part 1.

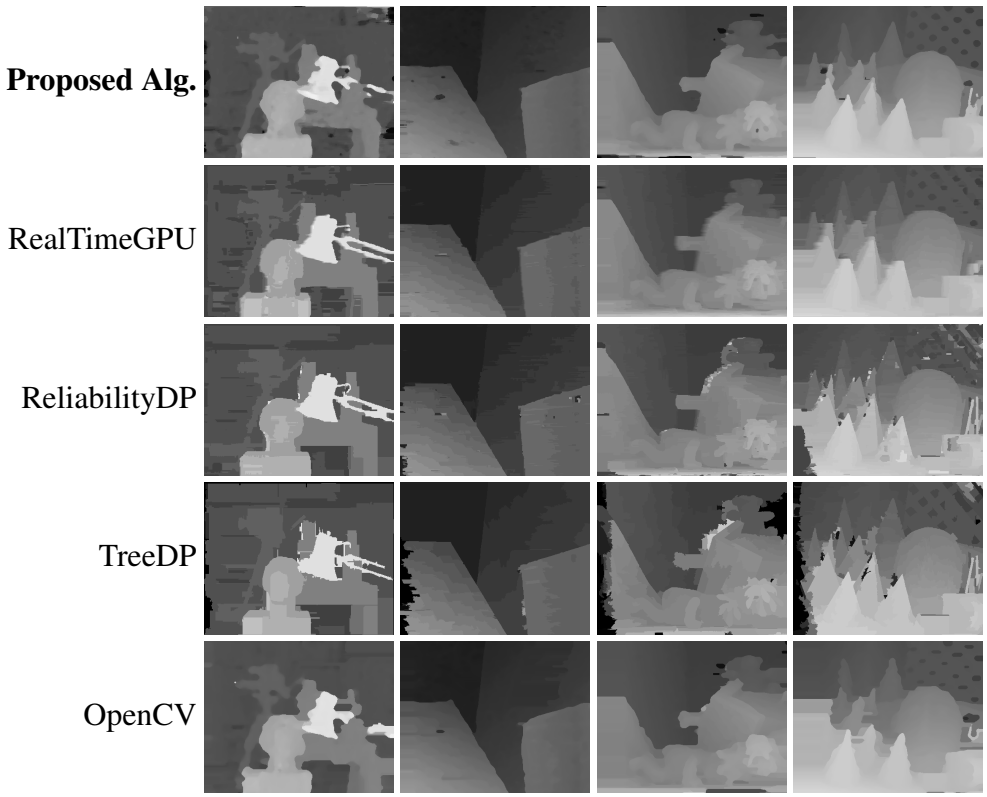


Figure 29: Visual comparison of the real-time algorithms on the Middlebury website, sorted by the main ranking in Tab. 4, part 2.

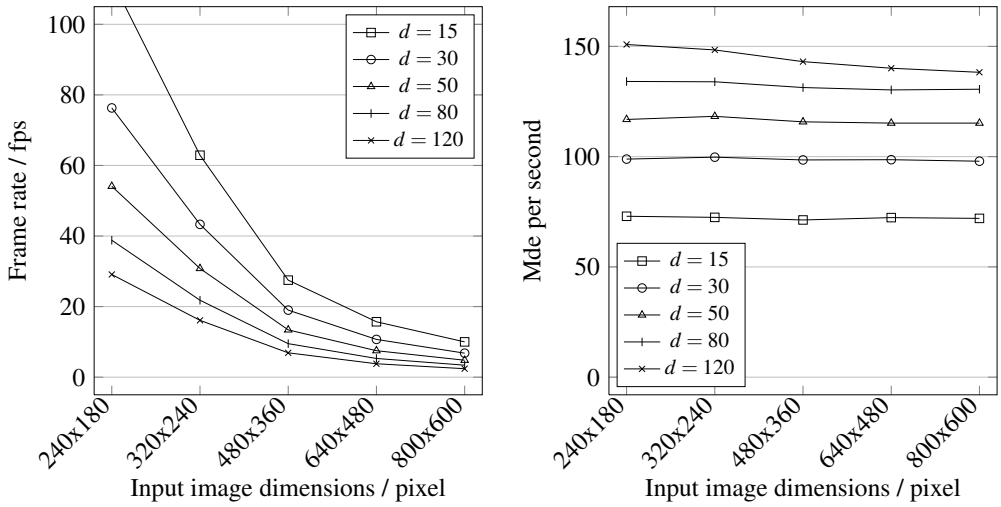


Figure 30: Optimized software implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on an Intel 2GHz Core 2 Duo CPU.

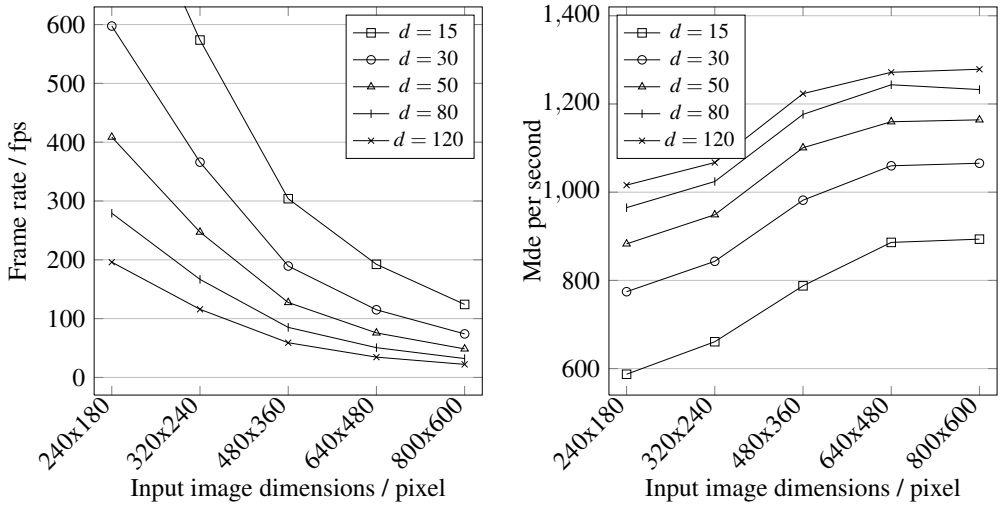


Figure 31: GPU implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on an NVIDIA GeForce GTX 280.

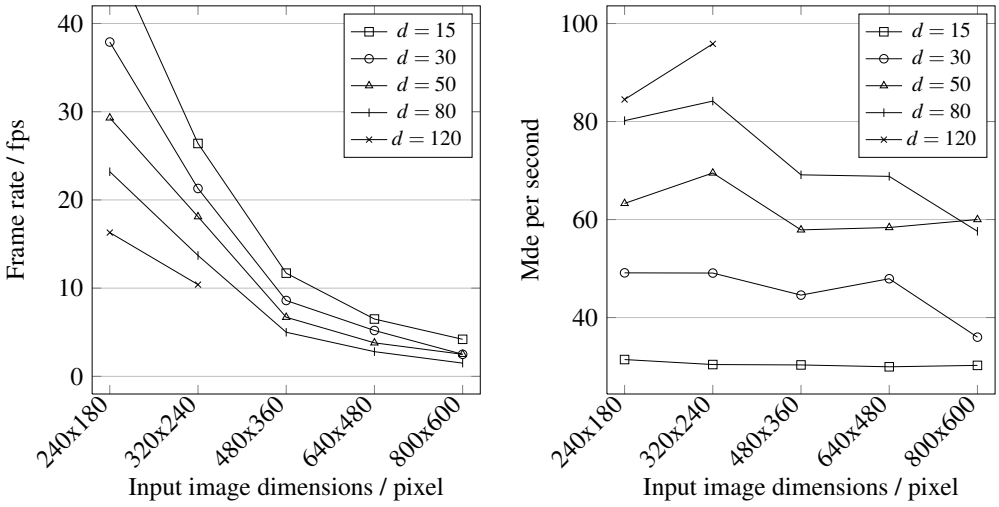


Figure 32: DSP implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on a 1GHz TMS320C6416 single core DSP. Some data points are missing due to memory restrictions of the evaluation board (6416DSK).

7. Summary and Outlook

After a general overview of stereo matching algorithms and systems, in this paper an algorithm for fast, Census-based stereo matching on embedded systems is presented. Because of the gain in processing time and the insignificant loss in quality, a sparse Census transform is used. The algorithm has been implemented on a PC, a GPU and a DSP. All implementations, aside from the plain software, reach real-time performance, whereby the GPU is by far the fastest but has the highest power consumption. The resulting disparity maps are evaluated on the Middlebury stereo website and perform well in comparison to other real-time algorithms. Especially in terms of processing times, the proposed algorithm performs very well in comparison to other algorithms.

The algorithm and its according reference implementations have several strengths. First of all, the proposed algorithm achieves high performance on several, including resource-limited, systems without losing good quality of stereo matching. The algorithm itself is robust, easy to parameterize and it delivers a good matching quality under real-world conditions. The implementations offer high flexibility in terms of image dimensions, disparity range, image bit-depth and frame rates, enabling the use of a wide variety of camera hardware. As a pure software solution, for embedded and non-embedded systems, it is able to run on a broad spectrum of COTS platforms which enables cost efficient stereo sensing systems as well as the integration of additional functionality in existing platforms.

Especially at disparity discontinuities, object borders and textureless areas, an improvement of the proposed algorithm would be of interest. Therefore, an integration of advanced matching techniques, such as global optimization approaches, will be investigated to improve the quality of the algorithm. Furthermore, a more sophisticated costs aggregation strategy could lead to better results. Indoor environments often suffer from difficult lighting conditions, so high dynamic range cameras will be used in the future to capture stereo image pairs. A processing time improvement for DSPs can be achieved by the use of upcoming multi-core DSPs.

References

[1] P. H. S. Torr, A. Zisserman, Feature Based Methods for Structure and Motion Estimation, Vision Algorithms: Theory and Practice, LNCS, Vol. 1883, 2000, pp. 278-294

[2] M. Sonka, V. Hlavac, R. Boyle, Image Processing, Analysis, and Machine Vision, Thomson-Engineering, 2nd Edition, 1998

[3] H. Hirschmueller, Improvements in Real-Time Correlation-Based Stereo Vision, IEEE Workshop on Stereo and Multi-Baseline Vision, 2001

[4] O. Faugeras, Three-Dimensional Computer Vision, A Geometric Viewpoint, The MIT Press, 4th Edition, 2001

[5] J. Y. Bouguet, Camera Calibration Toolbox for Matlab, http://www.vision.caltech.edu/bouguetj/calib_doc/, 2008

[6] G. Bradski, A. Kaehler, Learning OpenCV, Computer Vision with the OpenCV Library, O'Reilly, 2008

[7] Z. Zhang, Flexible Camera Calibration by Viewing a Plane from Unknown Orientations, ICCV, 1999, pp. 666-673

[8] OpenCV, http://sourceforge.net/projects/opencvlibrary/

[9] A. Fusiello, E. Trucco, A. Verri, A Compact Algorithm for Rectification of Stereo Pairs, Machine Vision and Applications, 2000, pp. 16-22

[10] W. van der Mark, F. C. A. Groen, J. C. van den Heuvel, Stereo Based Navigation in Unstructured Environments, IEEE Instrumentation and Measurement Technology Conference, 2001, pp. 2038-2043

[11] D. Murray, J. J. Little, Using Real-Time Stereo Vision for Mobile Robot Navigation, Autonomous Robots, 8, 2000, pp. 161-171

[12] R. Cucchiara E. Perini, G. Pistoni, Efficient Stereo Vision for Obstacle Detection and AGV Navigation, IEEE Conference on Image Analysis and Processing, 2007, pp. 291-296

[13] K. Konolige, m. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, B. Gerkey, Outdoor Mapping and Navigation Using Stereo Vision, Experimental Robotics, Springer Tracts in Advanced Robotics, Vol. 39, 2008, pp. 179-190

[14] D. Murray, C. Jennings, Stereo Vision Based Mapping and Navigation for Mobile Robots, IEEE Conference on Robotics and Automation, Vol. 2, 1997, pp. 1694-1699

[15] D. Burschka, G. Hager, Scene Classification from Dense Disparity Maps in Indoor Environments, IEEE Conference on Pattern Recognition, Vol. 3, 2002, pp. 708-712

[16] R. Zabih, J. Woodfill, Non-Parametric Local Transforms for Computing Visual Correspondence, European Conference on Computer Vision, 1994, pp. 151-158

[17] H. Hirschmueller, P. R. Innocent, J. Garibaldi, Real-Time Correlation-Based Stereo Vision with Reduced Border Errors, International Journal of Computer Vision, 2002, pp. 229-246

[18] O. Veksler, Fast Variable Window for Stereo Correspondence using Integral Images, IEEE Computer on Vision and Pattern Recognition, 2003, pp. 551-561

[19] S. Birchfield, C. Tomasi, Depth Discontinuities by Pixel-to-Pixel Stereo, International Journal of Computer Vision, Vol. 35:3, 1996, pp. 269-293

[20] S. Forstmann, Y. Kanou, J. Ohya, S. Thuering, A. Schmitt, Real-Time Stereo by using Dynamic Programming, IEEE Conference on Computer Vision and Pattern Recognition Workshop, Vol. 3, 2004, pp. 29-36

[21] R. C. Gonzalez, J. A. Cancelas, J. C. Alvarez, J. A. Fernandez, J. M. Enguita, Fast Stereo Vision Algorithm for Robotic Applications, IEEE Conference on Emerging Technologies and Factory Automation, Vol. 1, 1999, pp. 97-104

[22] Y. Ohta, T. Kanade, Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 7, 1985, pp. 139-154

[23] Y. Boykov, O. Veksler, R. Zabih, Fast Approximate Energy Minimization via Graph Cuts, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, 2001, pp. 1222-1239

[24] R. Zabih, Individuating Unknown Objects by Combining Motion and Stereo, Dissertation, Department of Computer Science, Stanford University, 1994

[25] V. Kolmogorov, R. Zabih, Computing Visual Correspondence with Occlusions using Graph Cuts, IEEE Conference on Computer Vision, 2001, pp. 508-515

[26] J. Sun, N. N. Zheng, H. Y. Shum, Stereo Matching using Belief Propagation, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, 2003, pp. 787-800

[27] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, D. Nister, Real-Time Global Stereo Matching using Hierarchical Belief Propagation, The British Machine Vision Conference, 2006, pp. 989-998

[28] M. Z. Brown, D. Burschka, G. D. Hager, Advances in Computational Stereo, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, 2003, pp. 993-1008

[29] D. Scharstein, R. Szeliski, A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence, International Journal of Computer Vision, Vol. 47, 2002, pp. 7-42

[30] D. Scharstein, R. Szeleski, High-Accuracy Stereo Depth Maps Using Structured Light, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, 2003, pp. 195-202

[31] D. Scharstein, C. Pal, Learning Conditional Random Fields for Stereo, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2007

[32] H. Hirschmueller, D. Scharstein, Evaluation of Cost Functions for Stereo Matching, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2007

[33] T. Kanade, A. Yoshida, K. Oda, H. Kano, M. Tanaka, A Stereo Machine for Video-rate Dense Depth Mapping and Its New Applications, IEEE Conference on Computer Vision and Pattern Recognition, 1996, pp. 196-202

[34] J. Woodfill, B. Von Herzen, Real-Time Stereo Vision on the PARTS Reconfigurable Computer, IEEE Symposium on FPGAs for Custom Computing Machines, 1997, pp. 242-250

[35] O. Faugeras, B. Hotz, H. Matthieu, T. Vieville, Z. Zhang, P. Fua, E. Theron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, C. Proy, Real-Time Correlation-Based Stereo: Algorithm, Implementations and Applications, INRIA Technical Report 2013, 1993

[36] Y. Miyajima, T. Maruyama, A Real-Time Stereo Vision System with FPGA, LNCS, Field-Programmable Logic and Applications, Vol. 2778, 2003, pp. 448-457

[37] J. I. Woodfill, G. Gordon, R. Buck, Tyzx DeepSea High Speed Stereo Vision System, IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2004, pp. 41-45

[38] J. I. Woodfill, B. v. Herzen, R. Zabih, Frame-rate Robust Stereo on a PCI Board, 1998

[39] Tyzx, Inc., Product Datasheet, http://www.tyzx.com/PDFs/tyzxDS_deepSeaG2VISION2.pdf

[40] Point Grey Research Inc., Triclops, Technical Manual, http://www.ptgrey.com/products/triclopsSDK/triclops.pdf

[41] Stereo-on-a-Chip Stereo Head User Manual 1.3, http://www.videredesign.com/vision/stoc.htm, 2007

[42] S. Kimura, T. Shinbo, H. Yamagucchi, E. Kawamura, K. Nakano, A Convolver-Based Real-Time Stereo Machine (SAZAN), IEEE Conference on Computer Vision and Pattern Recognition, Vol. 1, 1999, pp. 457-463

[43] N. Chang, T.-M. Lin, T.-H. Tsai, Y.-C. Tseng, T.-S. Chang, Real-Time DSP Implementation on Local Stereo Matching, IEEE Conference on Multimedia and Expo, 2007, pp. 2090-2093

[44] I. Ernst, H. Hirschmueller, Mutual Information Based Semi-Global Stereo Matching on the GPU, International Symposium on Visual Computing, 2008, pp. 228-239

[45] R. Yang, M. Pollefeys, S. Li, Improved Real-Time Stereo on Commodity Graphics Hardware, IEEE Conference on Computer Vision and Pattern Recognition Workshop, 2004, pp. 36-42

[46] B. Khaleghi, S. Ahuja, Q. Wu, An Improved Real-Time Miniaturized Embedded Stereo Vision System (MESVS-II), IEEE Conference on Computer Vision and Pattern Recognition Workshop, 2008, pp. 1-8

[47] H. Mathieu, A Multi-DSP 96002 Board, INRIA Technical Report 153, 1993

[48] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nistr, High-Quality Real-Time Stereo Using Adaptive Cost Aggregation and Dynamic Programming, International Symposium on 3D Data Processing, Visualization and Transmission, 2006, pp. 798-805

[49] M. Houston, High Level Languages for GPUs Overview, ACM SIGGRAPH 2007 courses, 2007

[50] NVIDIA, GeForce GTX 280, , http://www.nvidia.com/object/geforce_gtx_280.html, 2008

[51] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide, NVIDIA Corporation, version 2.0, 2008

[52] NVIDIA, GeForce 9800 GT, NVIDIA Corporation, http://www.nvidia.com/object/product_geforce_9800gt_us.html, 2008

[53] PCISIG, PCI Express, Specification, http://www.pcisig.com/specifications/pciexpress/, 2009

[54] Advanced Micro Devices, Inc. (AMD), Software Optimization Guide for AMD64 Processors, rev. 3.06, 2005

[55] M. Gong, R. Yang, L. Wang, M. Gong, A Performance Study on Different Cost Aggregation Approaches Used in Real-Time Stereo Matching, International Journal of Computer Vision, Vol. 45/2, 2007, pp. 283-296

[56] Intel Corporation, Intel Core2 Duo Processors and Intel Core2 Extreme Processors for Platforms Based on Mobile Intel 965 Express Chipset Family, Document Number:316745-005, 2008

[57] A. Kuznetsov, BitMagic Library: Document about SSE2 Optimization, http://bmagic.sourceforge.net/bmsse2opt.html, 2008

[58] C. Zinner, M. Humenberger, K. Ambrosch, W. Kubinger, An Optimized Software-Based Implementation of a Census-Based Stereo Matching Algorithm, Lecture Notes in Computer Science, Advances in Visual Computing, Vol. 5358, 2008, pp. 216-227

[59] C. Zinner, W. Kubinger, ROS-DMA: A DMA Double Buffering Method for Embedded Image Processing with Resource Optimized Slicing, Proceedings

of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06), 2006, pp. 361-372

[60] Texas Instruments, TMS320C6474 Multicore Digital Signal Processor, Lit. Number: SPRS552, 2008

[61] C. Zinner, W. Kubinger, R. Isaacs, Pfelib: A Performance Primitives Library for Embedded Vision, EURASIP Journal on Embedded Systems, Vol. 2007(1), 2007

[62] Texas Instruments, TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, Lit. Number: SPRS226K, 2003

[63] Q. Yang, R. Yang, J. Davis, D. Nister, Spatial-Depth Super Resolution for Range Images, IEEE Conference on Computer Vision and Pattern Recognition, 2007

[64] E. R. Davies, Machine Vision Theory, Algorithms, Practicalities, Academic Press, 3rd Edition, 2005

[65] Middlebury Computer Vision, Stereo Evaluation, http://vision.middlebury.edu/stereo/

[66] Q. Yang, C. Engels, A. Akbarzadeh, Near Real-time Stereo for Weakly-Textured Scenes, British Machine Vision Conference, 2008

[67] F. Tombari, S. Mattoccia, L. Di Stefano, E. Addimanda, Near Real-Time Stereo Based on Effective Cost Aggregation, International Conference on Pattern Recognition, 2008

[68] M. Gong, Y.-H. Yang, Near Real-time Reliable Stereo Matching Using Programmable Graphics Hardware, IEEE Conference on Computer Vision and Pattern Recognition, Vol. 1, 2005, pp. 924-931

[69] O. Veksler, Stereo Correspondence by Dynamic Programming on a Tree, IEEE Conference on Computer Vision and Pattern Recognition, Vol. 2, 2005, pp. 384-390

[70] H. Kopetz, Real-Time Systems, Design Principles for Distributed Embedded Applications, Springer, 1997

[71] K. Zhang, J. Lu, G. Lafruit, R. Lauwereins, L. V. Gool, Real-Time Accurate Stereo with Bitwise Fast Voting on CUDA, IEEE International Conference on Computer Vision, 5th Workshop on Embedded Computer Vision, 2009, pp. 794-800

[72] J. Salmen, M. Schlipsing, J. Edelbrunner, S. Hegemann, S. Lueke, Real-Time Stereo Vision: Making more out of Dynamic Programming, LNCS: Computer Analysis of Images and Patterns, Vol. 5702/299, 2009, pp. 1096-1103

[73] S. Kosov, T. Thormahlen, H. P. Seidel, Accurate Real-Time Disparity Estimation with Variational Methods, to appear in International Symposium on Visual Computing, 2009

View publication stats