TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DISSERTATION

Real-Time Stereo Matching for Embedded Systems in Robotic Applications

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Markus Vincze

eingereicht an der Technischen Universität Wien
bei der Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. (FH) Martin Humenberger
Matrikelnummer: 0003452
Webgasse 37/1/36
1060 Wien
humenberger@gmail.com

Wien, im April 2011

Abstract

In 3D perception with stereo vision two digital cameras are used to capture a target scene. A canonical stereo setup is used where the cameras are mounted in parallel onto a common, mechanically rigid structure. Each scene point visible in both images is projected onto the image planes of both cameras. Thus, the projections correspond to each other and can further be used to reconstruct the 3D position of the scene point. The horizontal displacement is the disparity which is inversely proportional to the depth of the projected point. The main task of a stereo vision system is to find these correspondences or, in other words, to solve the correspondence problem. A corresponding pixel pair is called a match.

In this thesis, we tackle the challenge of fast stereo matching for robotic applications and embedded systems. Motivated by the fact that limited resources, e.g. memory and processing power, and most importantly real-time capability on robot platforms do not permit the use of most existing sophisticated stereo matching approaches, we evaluated the strengths and weaknesses of different matching approaches and found a well-suited solution in a Census-based stereo matching approach. We adapt and optimize our algorithm in a way that the well-known Census transform can be used in embedded real-time systems without the need of dedicated hardware such as application-specific integrated circuits (ASIC) or field programmable gate arrays (FPGA). In contrast to the classic Census transform we use a sparse Census mask which halves the processing time with nearly unchanged matching quality. This is due to the fact that large sparse Census masks perform better than small dense masks with the same processing effort. We show the evidence of this assumption with the results of experiments with different mask sizes. Beside the algorithm we also present the complete stereo matching system as well as the detailed analysis and evaluation of the results. The system is robust, easy to parameterize and offers high flexibility in the target application field.

Furthermore, we give a detailed performance analysis of the algorithm for optimized reference implementations on various commercial off-the-shelf platforms, e.g. a personal computer, a digital signal processor, and a graphics processing unit, where our algorithm achieves a frame rate of up to 75 fps for 640×480 images and 50 allowed disparities. We compare the matching quality and processing time of our approach to other algorithms on the Middlebury stereo evaluation website where it achieves rank 73 out of 104 submissions in the main ranking and rank 23 if subpixel accuracy is presumed. Additionally, we evaluate the algorithm by comparing the results with a fast and well-known sum of absolute differences algorithm and a modified version of semi-global matching using several Middlebury datasets and real-world scenarios, which shows the enhanced performance of our algorithm.

Kurzfassung

Zur dreidimensionalen Datenerfassung mit Stereo Vision wird eine Szene von zwei digitalen Kameras erfasst. Hierfür werden die Kameras parallel zueinander auf einer Basis montiert. Jeder sichtbare Punkt der Szene wird auf die Bildebene beider Kameras projiziert. Die Projektionen korrespondieren miteinander und das Pixelpaar kann zur Rekonstruktion der dreidimensionalen Position des Szenenpunktes verwendet werden. Der horizontale Versatz eines korrespondierenden Pixelpaars ist die Disparität und ist indirekt proportional zur Tiefe des projizierten Punktes. Die Hauptaufgabe von Stereo Vision Systemen ist diese Korrespondenzen zu finden oder, in anderen Worten, das Korrespondenzproblem zu lösen. Ein gefundenes Pixelpaar wird Match genannt.

In dieser Arbeit stellen wir uns der Herausforderung von echtzeitfähigem Stereomatching, das den Anforderungen von Anwendungen in der Robotik genügen soll. Dazu gehört, neben hoher Rechengeschwindigkeit, auch die Möglichkeit einer eingebetteten Realisierung. Solche eingebetteten Systeme haben beschränkte Ressourcen, vor allem bei dem Speicher und der Rechenleistung, und erlauben deshalb keinen Einsatz von den meisten technisch ausgefeilten und anspruchsvollen Stereomatching-Algorithmen. Aus diesem Grund evaluierten wir die Stärken und Schwächen der bekannten Methoden und fanden eine passende Lösung in einem Census-basierten Verfahren. Gezielte Änderungen und Erweiterungen sowie die Optimierung des bekannten Verfahrens machen die Census Transformation echtzeitfähig und ermöglichen die Realisierung auf eingebetteten Systemen, ohne dass dedizierte Hardware, wie eigens hierfür erzeugte Halbleiterchips, verwendet werden müssen. Wir verwenden, im Gegensatz zur klassischen Census Transformation, ein spärlich besetztes Census-Fenster, das bei gleich bleibender Matchingqualität eine nahezu doppelt so schnelle Verarbeitung ermöglicht. Dies ist auf die Tatsache zurückzuführen, dass sich große, spärlich besetzte, Census-Fenster besser eignen als kleine vollständig besetzte Fenster. Diesen Sachverhalt zeigen wir durch Experimente mit unterschiedlichen Fenstergrößen. Neben dem Algorithmus stellen wir das komplette Stereo Vision System, mit einer genauen Evaluierung der Ergebnisse, vor. Das System ist robust, lässt sich leicht parametrisieren und ist für den Einsatz im Anwendungsfeld der Robotik flexibel und skalierbar.

Außerdem präsentieren wir eine detaillierte Analyse der Laufzeiten von hochoptimierten Realisierungen des Algorithmus auf unterschiedlichen kommerziell erhältlichen Plattformen wie einem Personalcomputer (PC), einem digitalen Signalprozessor (DSP) und einer Grafikkarte (GPU). Dabei erreichen wir mit unserem Algorithmus eine Bildwiederholungsrate von bis zu 75 Bildern in der Sekunde für eine Auflösung von 640×480 mit 50 Disparitätsstufen. Zur Evaluierung der Matchingqualität ver-

wenden wir die Middlebury Stereo Datenbank, in der unser Algorithmus Rang 73 von 104 Einträgen in der Haupttabelle und Rang 23, wenn Subpixel-Genauigkeit vorausgesetzt ist, erreicht. Zu guter Letzt vergleichen wir unseren Algorithmus mit dem bekannten Stereomatching Algorithmus Sum of Absolute Differences (SAD) und einer modifizierten Form des Algorithmus Semi-Global Matching, sowohl mit Middlebury Datensätzen als auch mit Aufnahmen aus dem Anwendungsfeld und zeigen eine klare Verbesserung der Ergebnisse mit unserem Ansatz.

Acknowledgements

Though this dissertation is an individual work, many people contributed in different ways, thus, it is important for me to thank them here. First of all I want to thank Markus Vincze for supervising my work. He woke my strong interest in computer vision for robotics and was a great help and scientific mentor. I also want to thank the Vision for Robotics (V4R) team around Markus Vincze especially Peter Einramhof, Walter Wohlkinger, and Sven Olufs who often helped me in fruitful discussions and constructive criticisms to observe the topic from different points of view. My honest gratitude goes to Margrit Gelautz who kindly agreed to review my work as secondary supervisor. Furthermore, I want to thank Michael Bleyer who helped me with his expertise in the topic stereo vision to find a good starting point for my work.

During my time as PhD student and before, I was employed as a research fellow at the AIT Austrian Institute of Technology. Here, I want to thank Manfred Gruber, head of the business unit Safe and Autonomous Systems, for giving me the opportunity to work for him during this time and the time after. I want to express my gratitude to Wilfried Kubinger who, first, encouraged me to write this thesis, second, supervised my work at AIT all the time, and third, was always a strong and helpful guide for me. Furthermore, I am grateful to all my colleagues at AIT who worked with me on the topic stereo vision and embedded systems. Especially, I want to thank Christian Zinner who always was and is a great inspiration. He significantly improved the quality of this work through his research expertise and engineering expert knowledge. My thanks also go to Christoph Sulzbachner, Jürgen Kogler, and Stephan Ramberger for a very nice work environment and fruitful discussions about several topics. I also want to thank Kristian Ambrosch for being a very helpful PhD study colleague, especially in publications, and Wolfgang Herzner who always gives helpful advices after paper reviews. Furthermore, I thank Daniel Hartermann, Michael Weber, and Tobias Engelke who worked with me during their diploma theses.

Most important, my highest gratuity goes to my family. Without their love and support during all my time as a student this work would not have been possible. I also want to thank my friends who always supported me and helped me to find distance when needed. Finally, my special thanks go to Felicitas Metzler. She was on my side all the time during this work and still is.

Financial support was provided by the European Union funded project robots@home^{TM} under grant FP6-2006-IST-6-045350.

Contents

List of Abbreviations

ASIC	application-specific integrated circuits
CCD	charge-coupled device
CM	confidence map
CMOS	complementary metal oxide semiconductor
CPU	central processing unit
CUDA	compute unified device architecture
DDR-RAM	double data rate random-access memory
DM	disparity map
DMA	direct memory access
DSI	disparity space image
DSL	disparity space layer
DSP	digital signal processor
ERAM	external random-access memory
FPGA	field programmable gate arrays
FPS	frames per second
GPU	graphics processing unit
GT	ground truth
IRAM	internal random-access memory
LIDAR	light detection and ranging
MDE/J	million disparity evaluations per Joule
MDE/S	million disparity evaluations per second
MiB	mebibyte (1MiB=2^{30}Byte)
MIPS	million instructions per second

NCC	normalized cross correlation
PC	personal computer
PCI	peripheral component interconnect
RMS	root mean square
ROS-DMA	resource optimized slicing-direct memory access
SAD	sum of absolute differences
SDRAM	synchronous dynamic random access memory
SGM	semi-global matching
SIMD	single instruction, multiple data
SSD	sum of squared differences
SSE	streaming SIMD extensions
TM	texture map
TOF	time-of-flight
TP	true positives
VLIW	very long instruction word
WTA	winner takes all
ZSAD	zero mean sum of absolute differences

Chapter 1

Introduction

Robotics summarizes design, development, and manufacture of robots. The task of a commercially available robot is the execution of a predefined and programmed function such as the automation of technical processes in industrial applications. Modern car manufacture is unthinkable without the use of robots, in detail robot arms, at the assembly lines. The term *robot* was invented by the Czech artist Josef Capek (1887–1945) and formed by the American author and professor of biochemistry at Boston University, Isaac Asimov (1920–1992). He also published the famous three laws of robotics [1]. The engineering science of robotics consists of electronics, mechanics, and software. Electronics and mechanics describe and realize physical sensors, actuators and manipulators which enable the interaction with the physical environment as well as movement of the robot itself. The tasks of the software, besides the control of the physical elements, are the robot's target application and the interpretation of sensor data to perceive the robot's surrounding environment and enable correct operation in there.

Besides robot arms in industrial applications, the promising research field of service robots exists. A service robot fulfills tasks which can help people in their everyday life and work. On the one hand, actually available examples for private usage are automatic household robots such as lawn-mowers or vacuum cleaners. The idea of controlling these robots is to specify a certain area, e.g. with a wire, where the robot can operate without the need of complicated path planning or mapping. On the other hand, service robots are interesting for industrial purposes as well and suitable solutions exist. Here, a service robot can fulfill tasks such as autonomous transport in office or factory buildings. Such autonomous vehicles are available as well, with the limitation of predefined paths.

A more sophisticated challenge is to realize autonomous navigation of service robots without predefined paths but with defined destinations or waypoints only. Then the robot has to deal with situations that are not a priori programmed. Besides industrial purposes, personal applications can be help for elder or handicapped people as well as household help in general.

In contrast to industrial robots, where the operation environment is well known and controlled, a personal service robot has to operate mostly in human environments. Such environments could be apartments, flats, or public places with an uncontrolled behavior. A robot always has to be able to deal with obstacles or, more important, suddenly appearing humans or animals in its operation area.

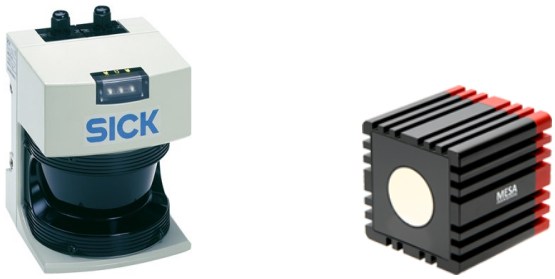


Figure 1.1: Left: LIDAR, Sick AG, LMS291, http://www.sick.de; Right: TOF, MESA Imaging AG, SwissRanger^{TM} SR4000, http://www.mesa-imaging.ch

For safe operation in such uncontrolled environments, dependable perception modules are needed for a reliable description of the surrounding of the robot. Especially 3D information is crucial for reliable operation in human environments. State-of-the-art embedded sensors for mobile robots are laser range finders or laser scanners (LIDAR, light detection and ranging) and time-of-flight (TOF) cameras. Embedded means that the sensors have to be able to operate independently of the main processing unit. This is advantageous because the processing power on a robot is limited and the application is computationally intensive anyway.

Common time-of-flight sensors illuminate the scene with modulated infrared light. They measure the time that the light needs from the sensor to an object and back to the sensor to calculate the distance of the object. In detail, a grid of infrared light emitting diodes is used for illumination of the scene and a CCD/CMOS chip detects the reflections. Laser scanners also use the round-trip time (or phase difference) of a laser beam to an object with the difference that the laser beam is used for a single point only and not for a whole area like TOF. This makes the laser more accurate with the drawback that the beam has to be moved along the scan line to cover more than one scene point.

The mentioned techniques, LIDAR and TOF, deliver accurate depth information but suffer from low resolution. Laser range finders additionally scan in one plane only, which means the field-of-view can only be horizontal or vertical depending on the mounting position. Obstacles that do not intersect with this plane can obviously not be detected. If a whole 3D scene should be scanned, the laser has to move the scanning plane which limits the robot's speed. The advantage of laser scanners is the high accuracy of a few centimeter and the robustness to lighting conditions.

Time-of-flight sensors have, as well as digital cameras, both a horizontal and vertical field-of-view. The main drawback is the low resolution of, e.g., 176×144 in comparison to industrial digital cameras where devices with an image resolution of, e.g., 640×480 are broadly available. Figure 1.1 shows two examples of commercially available range sensors. On the left, a laser range finder from the company SICK AG and, on the right, a time-of-flight camera from the company MESA Imaging AG are shown.

A promising alternative to TOF and LIDAR is stereo vision. A classic stereo sensor consists of two digital cameras which are mounted in parallel, separated by the baseline. 3D information is calculated using the correspondences between both images. The horizontal displacement of corresponding pixels is called the disparity. The stereo matching algorithm, in more detail finding the pixel correspondences of

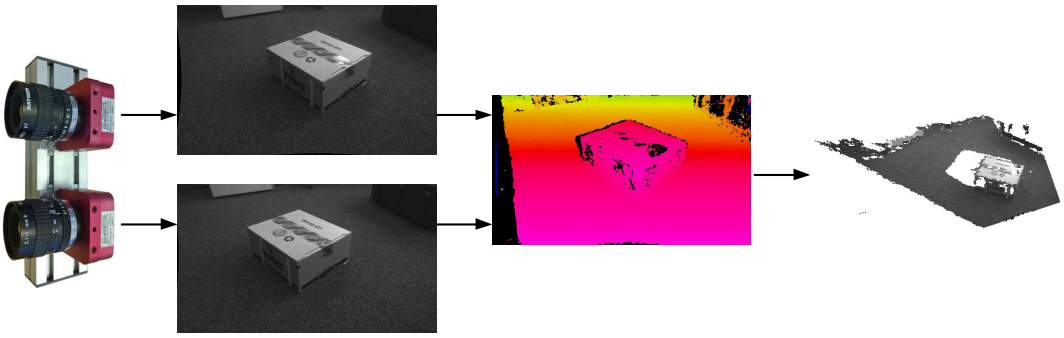


Figure 1.2: Principle of 3D perception with stereo vision; First, the images are captured with digital cameras, then corresponding pixels are searched which results in a disparity map (black areas mark unmatched pixels) and finally the 3D data of all found pixels is reconstructed.

the stereo image pair, is the core part of stereo vision systems. Figure 1.2 shows a typical stereo vision workflow for 3D data perception.

In contrast to time-of-flight and laser, stereo vision delivers 3D information and camera images of the captured environment synchronously. This makes it very well suited for robot applications because the camera images can be additionally used for other tasks such as scene classification. Each image capture of a stereo sensor delivers a 3D reconstruction of the captured scene visible in both camera images. The maximum lateral and depth resolution is physically given by the optical and geometrical properties of the digital cameras and their alignment. Stereo vision is a purely passive technology that uses, additionally to the cameras, only a processing unit to realize the sensing. The processing unit can be, e.g., a personal computer or an embedded system. The power consumption of an embedded stereo sensor is about 1 W for the cameras and about 5 W for the embedded processing unit. This is low in comparison to TOF (about 12 W) and LIDAR (about 30 W).

The mentioned properties of stereo vision, especially the higher resolution, the lower power consumption, the fact that digital images can be used for other tasks as well, and the passive 3D data perception make stereo vision very interesting for robot applications. High quality stereo matching algorithms are computationally intensive and require a large amount of hardware resources. Integrating such an algorithm into an embedded system, which is in fact limited in resources, scale, and power, is a delicate task. The real-time requirements of robot applications add additional constraints to the realization. The definition of the term real-time by Kopetz [2] which means that a task has to be finished within an a priori defined time frame is extended in this work. Additionally, we make demands on fast (at least 10 fps), constant, and scene-independent processing time.

The key to success in realizing a reliable, embedded, and real-time-capable stereo vision system is the careful design of the core algorithm and an optimized implementation on proper hardware platforms. Not all algorithms and methods are realizable in real-time up to now, so the implementation has to be part of the algorithm design as well. A suitable trade-off between computational effort, computation time, memory consumption, and quality of stereo matching must be found. The detailed analysis of possible matching approaches, the design of an adequate method to solve

Figure 1.3: Mobile robot platform "James" of the Automation and Control Institute at the Vienna University of Technology, with and without casing

the correspondence problem, the complete evaluation of the algorithm and the optimized realization from the scratch on several target platforms for robotic applications are the topics of this thesis.

1.1 Requirements

This thesis arose in close collaboration of the Automation and Control Institute (ACIN) of the Vienna University of Technology and the Safety & Security Department (DSS) of the AIT Austrian Institute of Technology. The scientific research areas of ACIN include computer vision for robotics, DSS has skills in embedded and safety critical systems for computer vision applications.

ACIN and DSS are partners in the research project robots@home[1] with the objective to provide an open mobile platform for the massive introduction of robots into the homes of everyone. Target applications could be, among others, home help, food delivery, security or elderly care. Stereo vision, as well as other sensor technologies shall be used for the main robotic challenges such as obstacle detection and navigation but also for more high-level tasks such as learning and mapping of rooms and classifying of items and furniture. The success of the project will be the learning of four homes and the heading for at least ten annotated pieces of furniture. The stereo vision system should overcome known drawbacks of actually used sensors like time-of-flight or laser; it should stay in a lower price segment and approve the valuable usage of a stereo matching systems as 3D perception sensors.

Figure 1.3 shows the research robot platform "James" of the Automation and Control Institute (ACIN) of the Vienna University of Technology equipped with laser, time-of-flight, and stereo vision 3D perception modules. It is used for realization and testing of the developed algorithms.
The following requirements for the stereo vision sensor have to be fulfilled:

[1] European Union funded project robots@home^{TM} under grant FP6-2006-IST-6-045350.

Real-time processing: The calculation of the 3D data has to be fast (at least 10 fps), constant, known and scene-independent to afford interactive behavior of the robot in a human environment.

Reliable 3D data: The resulting 3D data have to be reliable because a robot cannot trust on data for critical processing tasks which is provided but false with a high probability. Thus, uncertain 3D points have to be determined.

Accurate 3D data: The accuracy of the 3D data decreases with the increasing distance of scene points. This is systematically given, so the sensor has to be adaptable to the optimal working distance. Subpixel accuracy is also helpful to cope with the large depth steps caused by integer disparity steps in large distances.

Dense 3D data: One advantage of stereo vision is the large amount of 3D points that can be provided per image capture. Obviously, as many reliable 3D points as possible have to be found.

Embedded realization: The processing power of the onboard computer of the robot is limited, so an additional calculation unit has to be used for the stereo matching. On a robot, electric power as well as mounting space are limited resources. Embedded systems have a small form factor and low power consumption, and thus, are well suited for robot applications.

Scalable design: The target platform cannot be strictly defined, so the design has to be scalable in terms of resource usage, image resolution, and processing speed.

Memory awareness: On embedded platforms, a frequent constraint is the fast on-chip memory; thus, the algorithm has to be memory-aware.

Low price: Service robots for common homes have to be much cheaper than robots in production lines or other industrial applications. The stereo sensor is only a small part of it and, thus, has to cost a fraction of the whole system. Due to their high price, embedded systems using field programmable gate arrays (FPGA) or application specific integrated circuits (ASIC) are not favorable in this work.

Passive: The stereo sensor is optimized for home robot application. This includes that it should influence its environment as little as possible. Contrary to active sensors, such as laser scanners or time-of-flight cameras, stereo is purely passive and does not disturb the surrounding environment with light beams or other radiance.

Power consumption: Due to the usage of the sensor on robotic platforms where power is a critical resources this requirement is essential.

1.2 Contributions

As part of computer vision, the research in stereo matching algorithms is driven by the motivation of ideally calculating a correct and perfectly dense disparity map.

Processing time is only of secondary priority. Many well-matching algorithms exist but cannot be used in robot practice because of their high processing time and high memory consumption. The contribution of this thesis is the design of a stereo matching algorithm and the integration into a stereo vision system that overcomes these problems by fulfilling all requirements listed above. Thus, the algorithm is a well-balanced trade-off between resource consumption and results quality. It 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 reliable 3D data. An adaption of the Census transform makes this possible, which halves the processing time with nearly unchanged matching quality. This allows the use of large Census mask sizes while still enabling real-time performance. An advantage is also the good scalability which makes it applicable to robotics because it can be adapted according to the needs of the autonomous platform. It is well suitable for embedded systems because it can be processed in small pieces sequentially as well as for computational powerful platforms such as graphics processing units (GPU) where high image resolutions can also be processed in real-time because of the possibility of parallel implementation. The highly optimized implementation on several hardware platforms is an important contribution of this work as well. We give a detailed performance comparison and analysis of the completely different processing platforms. As a pure software solution, it reaches a performance level that was possible on expensive hardware only up to now. To make this possible, we use memory and processing time-efficient approaches for calculating and storing the matching costs before a local optimization technique is used to find the optimal disparity. Due to the fact that the depth resolution decreases with the growing distance, subpixel refinement is used to increase the accuracy of the matches.

The algorithm can be adapted according to different camera characteristics by using a confidence and a texture metric. The confidence estimates the probability of the uniqueness of each match and the texture adjusts the algorithms' sensitivity to the cameras' noise. We evaluate the results by the use of the Middlebury stereo evaluation website as well as with images with overlayed synthetic noise and with scenes under real-world conditions. In the Middlebury main ranking, the algorithm is well-ranked in comparison to all other real-time algorithms and outperforms all higher ranked algorithms in processing time clearly. The algorithm is also the only purely embedded solution in the ranking up to now.

1.3 Organization

The remainder of this thesis is organized as follows. Chapter 2 introduces the fundamentals of stereo vision, the epipolar geometry, the correspondence problem, the 3D reconstruction and the camera calibration. Chapter 3 describes the state-of-the-art in stereo matching algorithms, introduces real-time stereo vision systems and algorithm evaluation techniques. Chapter 4 gives a description of the proposed real-time stereo engine, where each step, from image acquisition to 3D reconstruction is described in detail. Afterwards, the matching quality of the algorithm is evaluated in Chapter 5 with a wide number of parameter configurations. Chapter 6 shows the reference implementations on a personal computer (PC), a digital signal processor (DSP), and a graphics processing unit (GPU) with a comprehensive performance

comparison. Finally, Chapter 7 concludes the thesis and gives an outlook to future research.

1.4 Resulting Publications

The work presented in this thesis has appeared in following journal and conference papers:

1.4.1 Journal Paper

Martin Humenberger, Christian Zinner, Michael Weber, Wilfried Kubinger, and Markus Vincze. *A Fast Stereo Matching Algorithm Suitable for Embedded Real-Time Systems.*
Journal on Computer Vision and Image Understanding, 2010, http://dx.doi.org/10.1016/j.cviu.2010.03.012, Elsevier.

1.4.2 Conference and Workshop Papers

Martin Humenberger, Tobias Engelke, and Wilfried Kubinger. *A Census-Based Stereo Vision Algorithm Using Modified Semi-Global Matching and Plane-Fitting to Improve Matching Quality.*
In *Proceedings of the Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, 6th Workshop on Embedded Computer Vision*, 2010.

Martin Humenberger, Christian Zinner, and Wilfried Kubinger. *Performance Evaluation of a Census-Based Stereo Matching Algorithm on Embedded and Multi-Core Hardware.*
In *Proceedings of the 6th International Symposium on Image and Signal Processing and Analysis*, 2009.

Martin Humenberger, Daniel Hartermann and Wilfried Kubinger. *Evaluation of Stereo Matching Systems for Real World Applications Using Structured Light for Ground Truth Estimation.*
In *Proceedings of IAPR Conference on Machine Vision Applications*, 2007.

Martin Humenberger and Wilfried Kubinger. *A Stereo Matching Development Platform with Ground Truth Evaluation and Algorithm Taxonomy for Embedded Systems.*
In *Proceedings of the 18th International DAAAM Symposium*, 2007.

Chapter 2

Fundamentals of Stereo Vision

The research area around stereoscopy started long ago in 1840 as Sir Charles Wheatstone first used this method to create a three dimensional illusion of a scene by showing the eyes two different images, captured side by side. The brain matches the image pair and creates the 3D illusion. This technique was first used in entertainment and experimental setups only, but found function in optical distance measurement later on as well. From the invention of the digital camera, this concept was picked up by computer vision researchers to use it for 3D sensing. They tried to do the matching, done by the human brain so far, with computer vision algorithms. Many approaches came up but the so called correspondence problem is still not completely solved.

Classic stereo vision uses a stereo camera setup, which is built up of two cameras, called stereo camera head, mounted in parallel. It captures a synchronized stereo pair consisting of the left camera's image and the right camera's image. A typical stereo head is shown in Fig. 2.1; the distance between both cameras is called the baseline. The main challenge of stereo vision is the reconstruction of 3D information of a scene captured from two different points of view. In this chapter, a summary of stereo vision fundamentals is given. Additional information can be found in, e.g., Sonka et al. [3], Faugeras [4], Gonzalez and Woods [5], Hartley and Zisserman [6], and Davies [7].

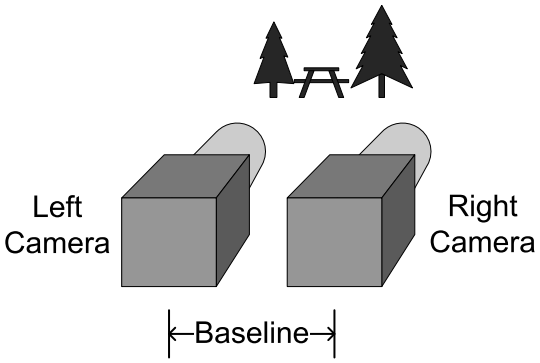


Figure 2.1: Typical stereo camera head

First, we describe in Section 2.1 the geometry of a stereo setup. Then we explain the difficulties of finding the correct correspondences in Section 2.2 followed by the 3D reconstruction of the captured scene in Section 2.3. Finally, in Section 2.4 we

describe how stereo cameras need to be calibrated to calculate the exact stereo camera geometry.

2.1 Epipolar Geometry

The pinhole camera model is used to explain the stereo camera geometry. Figure 2.2 illustrates the projection of a scene point P onto the image planes, π_l and π_r, of two cameras. The cameras are separated by the baseline b, and have the optical centers at O_l and O_r. Of course, real lenses usually cause distortion in the images which can be corrected, as explained later on in Section 2.4. The scene point $P = (x, y, z)$ is given in world coordinates (meters) and its projection onto the image planes, $p_l = (u, v)$ and $p_r = (u, v)$, in image coordinates (pixels). The points p_l and p_r represent the same scene point and thus correspond to each other.

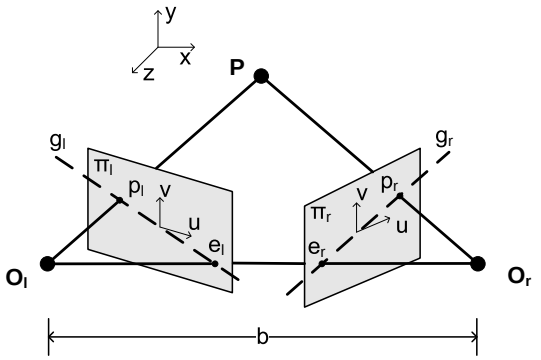


Figure 2.2: Arbitrary aligned stereo vision geometry as pinhole camera model

The points p_l and p_r lie on their epipolar lines g_l and g_r which are defined by the intersection of the plane spanned by (P, O_l, O_r) and the image planes π_l and π_r. The intersection of the baseline b and the image planes are called epipoles e_l and e_r. When searching for the correspondence of a pixel in the one image, each possible matching candidate lies on the according epipolar line in the other image. The search for the best match is therefore restricted to a search along the corresponding epipolar line instead of the whole image This reduces the processing effort significantly.

The epipolar line g_r corresponds to the pixel p_l and g_l to p_r. The corresponding epipolar lines can only be calculated if the geometry between both cameras is known. The computation of this epipolar geometry is part of stereo calibration, which is explained in Section 2.4.

Once the geometry is known, the images can be rectified. In rectified images, the epipolar lines are horizontal, the epipoles are mapped to infinity and corresponding pixels have the same v-coordinate. The epipolar lines correspond to the horizontal scanlines, which makes the search for matches much easier, because only one image coordinate changes. The arbitrary aligned stereo camera setup in Fig. 2.2 changes after rectification to the setup in Fig. 2.3.

An important restriction also reduces the search range for the corresponding pixel along the epipolar lines. Figure 2.4 shows that if the scene point P is assumed to be within the distance $z_{c,range}$ with respect to the right camera coordinate system, the possible positions of the corresponding point p_l lies within the disparity search

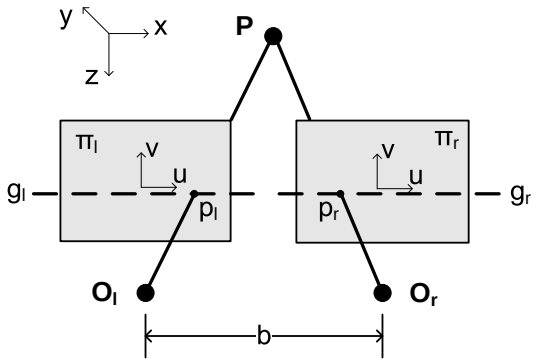


Figure 2.3: Rectified stereo vision geometry as pinhole camera model

range d_{range}. Low disparities mean high distance and vice versa. The disparity range defines the depth range of the sensor and is freely configurable. A common usable search range starts at infinity (disparity 0) and searches until a defined minimum detectable distance (maximum disparity). The processing effort increases for high search ranges, so it should be well chosen for the target application.

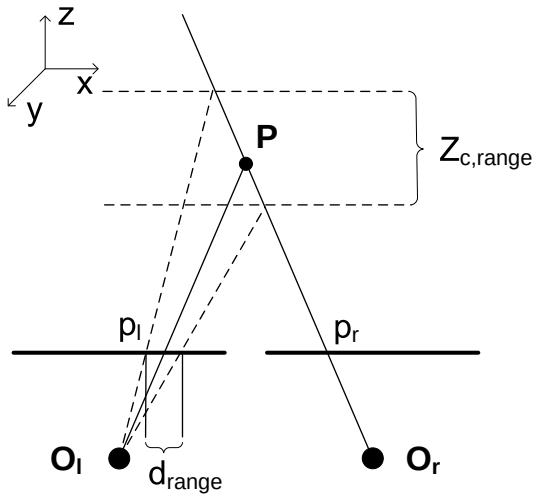


Figure 2.4: Disparity range along the epipolar line for rectified images

2.2 The Correspondence Problem

As a reminder, the correspondence problem for projected scene points is solved by a stereo matching algorithm and the result is a disparity map. This is an image of the same size like the stereo pair images containing the disparity for each pixel instead of the intensity value.

As a consequence of the stereo geometry, not all areas within the field of view are visible in both cameras. Especially at depth discontinuities, so called occlusions appear. Areas that are visible in one camera only are called half-occluded and areas visible in none of the cameras, occluded. Figure 2.5 shows an example for both.

Each scene point visible in both images has exactly one representing pixel per

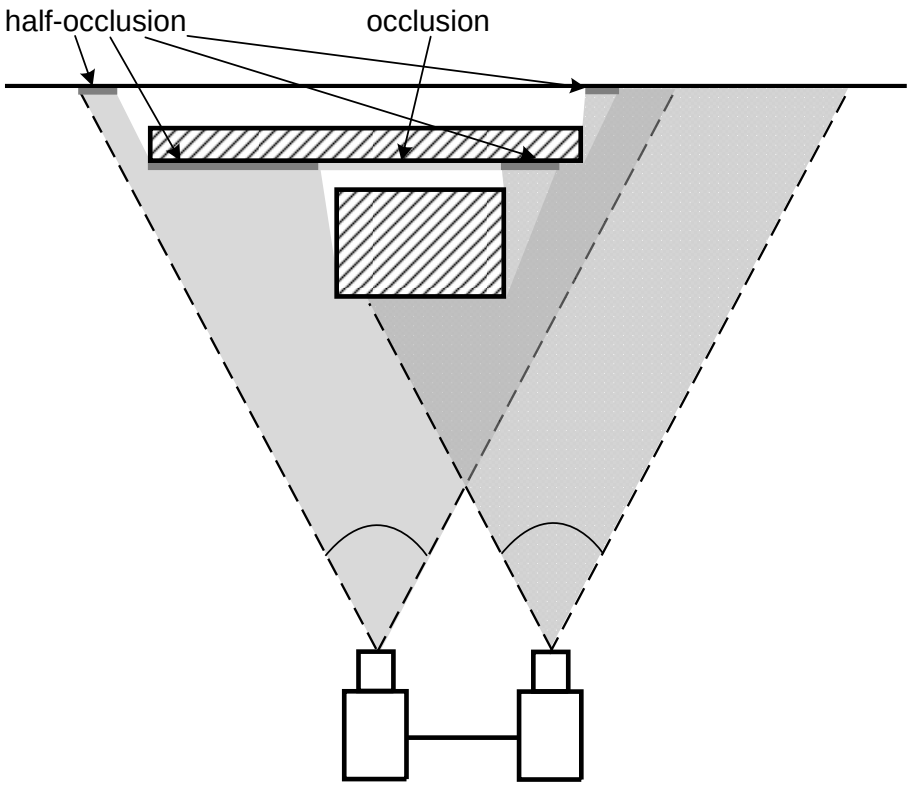


Figure 2.5: Occluded and half-occluded areas in a scene captured with a stereo camera head

image, and these pixels have to be determined uniquely. In practice, this is not so easy, due to the vast number of similar scenery points in different distances which cause a pixel in, say, the left image, to be mapped to a series of similar pixels in the right image. The problems stereo matching algorithms have to face are:

Occlusions: These areas cannot be matched because they are not visible in both images.

Reflections: The camera cannot distinguish between the reflecting surface and the reflected object. Thus, the distance of the reflected object is calculated instead of the surface which would be the correct distance.

Transparency: Perfect transparent objects and surfaces are impossible to match because they cannot be identified by the cameras. Additionally, transparent surfaces such as glass often also cause reflections and violate the uniqueness constraint as described in Section 3.1.2 later on.

Textureless areas: On textureless areas, all pixels have nearly the same color. Thus, a pixel of one image has a huge number of equal matching candidates in the other image. This decreases the probability of choosing the correct pixel.

Repetitive textures: As well as textureless areas, repetitive textures offer lots of equal matching candidates and thus are difficult to match.

Thin objects: If a neighborhood of pixels is used to increase the matching quality, pixels of thin objects are hard to find if the neighborhood is broader than the object.

Cameras: All known problems occurring by the use of cameras exist in stereo vision as well. Especially in robotics, a big problem is the limited dynamic range of the cameras. If a robot moves from a dark room into a very bright room, a good auto shutter is needed. The real problem comes up if one area of the image is very dark and an other area is very bright because common image sensors and optics can only be adjusted for the whole image.

2.3 3D Reconstruction

Once the correspondences between the stereo images are found, the disparity (the horizontal displacement of corresponding pixels) is defined as shown in Fig. 2.6.

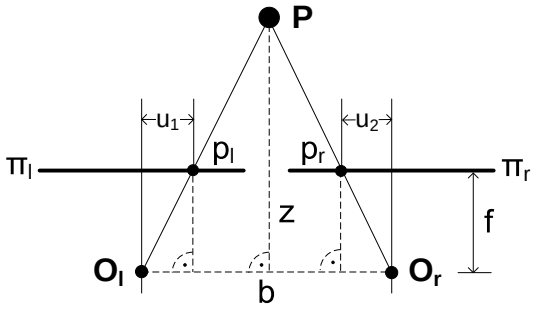


Figure 2.6: Corresponding pixels with disparity $d = u_2 - u_1$

For 3D reconstruction, the used coordinate systems have to be known; Fig. 2.7 shows how they are defined. O is the optical center of the camera, f the focal length of the lens, (u, v) are the image coordinates on the image plane, (x_c, y_c, z_c) are the camera coordinates of the projected points in space with the camera's optical center as origin and (X, Y, Z) are the world coordinates with the application-specific origin.

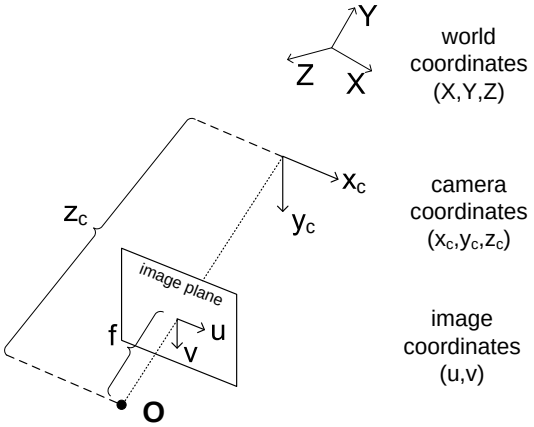


Figure 2.7: Camera, image, and world coordinate system

For depth calculation, as shown in Fig. 2.6, triangulation is used for each pixel to obtain

$$z = \frac{b \cdot f}{d}, \tag{2.1}$$

where z is the distance between the camera's optical centers and the projected scene point P, b is the baseline, d the disparity and f is the focal length of the camera. This relation can be derived from the similar right angle triangles in the model as explained in Gonzalez and Woods [5]. The depth of a pixel is therefore indirectly proportional to its disparity.

The complete 3D point cloud in camera coordinates can be calculated with

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = K^{-1} \begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \end{pmatrix} \tag{2.2}$$

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 Equ. (2.1). K and f have to be determined by the camera calibration explained in the next section. If the mounting position of the stereo sensor is known, the point cloud can be transformed to any world coordinate system by a translation and rotation.

2.4 Camera Calibration

Camera calibration is an essential part of stereo vision because it is used to determine the exact camera and stereo parameters needed for 3D reconstruction and optimized stereo matching. On the one hand, both cameras have to be calibrated separately to obtain the intrinsic and extrinsic camera parameters for lens undistortion and 3D reconstruction. On the other hand, a stereo calibration has to be done to determine the geometry between both cameras which will be further used to rectify the stereo images. The calibration method used in this work is well chosen with respect to the needs of embedded stereo vision. The calibration process is based on capturing images of known calibration pattern, e.g. chessboards. The correspondences between camera and world coordinate systems are used for single camera calibration and the correspondences between the stereo image pair for stereo calibration. Methods for calculating the calibration parameters can be found in Zhang [8], Sonka et al. [3], Fusiello et al. [9] and Bradski and Kaehler [10]. Implementations of most of the described functions can be found in the Caltech Calibration Toolbox, Bouguet [11], and in the OpenCV library [12].

2.4.1 Single Camera

The intrinsic camera parameters are given by the camera calibration matrix K, Equ. (2.3), and the distortion coefficients described later on. The calibration matrix contains the focal length f which is given in horizontal pixels (f_x) and in vertical pixels (f_y). If the pixels are exact quadratic, f_x and f_y are equal. It also contains the principal point $C(c_x, c_y)$ which is the true center of the image plane in pixels.

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

The calibration matrix is used to transform the camera coordinates (x_c, y_c, z_c) to image coordinates (u, v) with

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \frac{1}{z_c} \begin{pmatrix} f_x x_c \\ f_y y_c \end{pmatrix}. \tag{2.4}$$

If homogeneous coordinates are used, the coordinate transform is the matrix multiply

$$\begin{pmatrix} uz_c \\ vz_c \\ z_c \end{pmatrix} = K \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}. \tag{2.5}$$

The inversion of Equ. (2.4) is

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = z_c \begin{pmatrix} \frac{u-c_x}{f_x} \\ \frac{v-c_y}{f_y} \\ 1 \end{pmatrix}. \tag{2.6}$$

Needless to say that it is only unique if z_c is known. If z_c is calculated with Equ. (2.1) and (u, v) is one of the corresponding pixels, Equ. (2.6) can be used to reconstruct the 3D data given in camera coordinates of the projected scene point. Of course homogenous coordinates can be used as well with

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = K^{-1} \begin{pmatrix} uz_c \\ vz_c \\ z_c \end{pmatrix}. \tag{2.7}$$

The assumption of ideal central projection usually fails by the use of real lenses because of lens distortion. To overcome this problem, the camera images have to be undistorted using a radial and tangential distortion model described with the distortion coefficients d_{r1}, d_{r2} and d_{r3} for radial and d_{t1}, d_{t2} for tangential distortion. Let (u_d, v_d) be the image coordinates in the distorted image. For undistortion, every pixel in the undistorted image (dst) is given by the pixel (u_d, v_d) in the distorted image (src).

$$dst(u, v) = src(u_d, v_d) \tag{2.8}$$

formulates this backward transform. If this is done for the whole image, the generated map can be used for every image captured by the calibrated camera. The benefit of this backward transform is that, unlike to forward transform, it is ensured that all pixels of the destination image get filled. Subpixel coordinates are expected, so the exact pixel value is determined by bilinear interpolation.

The calculation of (u_d, v_d) works backwards as well. First, the ideal image coordinates (u, v) of the undistorted image have to be transformed in camera coordinates (x_c, y_c) with Equ. (2.6) where $z_c = 1$. After that, the distortion model is applied to (x_c, y_c) with

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \end{pmatrix} (1 + d_{r1} r^2 + d_{r2} r^4 + d_{r3} r^6) + \begin{pmatrix} d_{t1} 2 x_c y_c + d_{t2}(r^2 + 2x_c^2) \\ d_{t1}(r^2 + 2y_c^2) + d_{t2} 2 x_c y_c \end{pmatrix} \tag{2.9}$$

where

$$r = \sqrt{x_c^2 + y_c^2}. \tag{2.10}$$

The final distorted image coordinates are transformed back from the distorted camera coordinates (x_d, y_d) with Equ. (2.4) where $x_c = x_d$, $y_c = y_d$ and $z_c = 1$. The result is the location (u_d, v_d) of the undistorted pixel (u, v) in the original image.

The single camera calibration is mainly the calculation of the intrinsic and extrinsic camera parameters. It is then used for correcting the distortion of the camera images and after the stereo matching process, for the 3D reconstruction of the projected scene points with respect to one of the cameras' coordinate systems.

2.4.2 Stereo Camera

A stereo camera head built up of two identical cameras placed side by side perfectly in parallel is called a rectilinear stereo rig. Even if this perfect assumption is valid only theoretically, the cameras should be set up as rectilinearly as possible. The missing accuracy can be obtained with stereo camera calibration which makes it essential for stereo vision systems. It determines the epipolar geometry of both cameras which offers the possibility of stereo rectification.

The epipolar geometry is described by the so called fundamental matrix F with the core characteristic

$$p_r^T F p_l = 0 \tag{2.11}$$

where p_l and p_r represent corresponding pixels in the left and in the right image. The epipolar line e_l of a pixel p_r in the right image can be determined with

$$e_l = F^T p_r \tag{2.12}$$

and the epipolar line e_r of a pixel p_l in the left image with

$$e_r = F p_l. \tag{2.13}$$

One possibility of stereo matching is to compute the corresponding epipolar line for each pixel to avoid the need of stereo rectification. The advantage of this approach is that the input images do not suffer from transforms, which means more accuracy in 3D reconstruction. The drawback is that memory accesses during costs aggregation are much more complex, which causes more processing time. As described later on in Chapter 4, the costs aggregation is a linear filter with a quadratic kernel. The epipolar lines of unrectified images are skewed so a processing time optimized memory access is not realizable. So, in this work, the input images are rectified before the stereo matching is applied. For rectification, basically two techniques came up where both calculate, as well as for lens distortion correction, a backward transform which follows Equ. (2.8).

The first technique was introduced by Hartley [13] where no knowledge about the intrinsic and extrinsic camera parameters is needed and is therefore called uncalibrated method. It can be used if undistorted images are available. The only requirements are accurate point correspondences between the stereo images which can be first used to calculate the fundamental matrix and afterwards to calculate the homography matrices $H_{r,l}$ which are used to rectify the camera images. Alternatively, the homographies can be transformed in 3D rotation matrixes for the camera coordinate system following

$$R_{r,l} = K_{r,l}^{-1} H_{r,l} K_{r,l}. \tag{2.14}$$

The advantage of the uncalibrated method is that it enables online calibration due to the fact that only point correspondences are needed. The drawback is that the intrinsic camera parameters are unknown, thus the 3D reconstruction is possible up to a projective transform. This means that objects with different sizes could seem to be the same.

The second technique was introduced by Tsai [14], Zhang [8], and Zhang [15] and uses knowledge about both camera geometries for calculation of the transform matrixes and is therefore called calibrated method and will be used in this work. Jean-Yves Bouguet implemented this method in his well-known camera calibration toolbox for Matlab [11]. The geometry of a camera is represented by its perspective projection matrix $P = K(R|t)$ which can be used to transform camera to world coordinates. R and t are the extrinsic camera parameters, determined by single camera calibration, where R is the rotation and t the translation between camera and world coordinates. The results of this technique are two 3×3 rotation matrixes $R_{r,l}$ which rotate the camera coordinate systems until the epipolar lines are horizontal, the epipoles are at infinity and corresponding pixels share the same v coordinate. The camera matrices also change to rectified camera matrixes

$$K'_{r,l} = \begin{pmatrix} f'_x & 0 & c'_x \\ 0 & f'_y & c'_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.15}$$

The rectification process itself is also a backward transform which calculates the coordinates (u_r, v_r) in the original image for each pixel (u, v) in the rectified image. First, (u, v) is transformed to camera coordinates with Equ. (2.7) where $z_c = 1$ and $K = K'_{r,l}$. After that, the rectification is inverted with

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = R_{l,r}^{-1} \begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} \tag{2.16}$$

and

$$x' = \frac{x}{w} \qquad y' = \frac{y}{w}. \tag{2.17}$$

Finally, the unrectified camera coordinates (x', y') are transformed to the final unrectified image coordinates (u_r, v_r) with Equ. (2.5) where $x_c = x'$, $y_c = y'$ and $z_c = 1$. The pixel value is determined by bilinear interpolation. The generated maps can be further used for rectification of all images captured by the calibrated stereo head.

The quality Q of the calculated stereo calibration can be determined by the use of the characteristics of the fundamental matrix in Equ. (2.12) and Equ. (2.13). They are used to determine the epipolar lines for the corresponding pixel pairs used for stereo calibration. If the calibration is perfect, p_l lies on its corresponding epipolar line e_r and p_r on e_l. If $ax + by + c = 0$ represents $e_{l,r}$, $p_{l,r} = (x, y)$ has to fulfil this equation. The quality is the root mean square (RMS) error over all corresponding pixel pairs, calculated with

$$Q = \sqrt{\frac{1}{N} \sum_{i=0}^{N} (e_{r,a} p_{l,x} + e_{r,b} p_{l,y} + e_{r,c})^2 + (e_{l,a} p_{r,x} + e_{l,b} p_{r,y} + e_{l,c})^2} \tag{2.18}$$

where N is the total number of pixel pairs.

2.4.3 Rectification

The previous section described how to calculate the transform maps for correction of the lens distortion and rectification of the stereo camera images. These maps can be combined to a single transform per image. On embedded systems, the camera calibration parameters, consisting of the camera matrices, the distortion parameters, the rectification matrices, and the projection matrices are used to calculate the maps at start-up. As a reminder, once the combined transform maps, from now on called rectification maps, are calculated, they can be used to rectify (lens distortion correction is from now on included in this term) the input images at runtime.

For this, also a backward transform is used to calculate the image coordinates (u, v) of the rectified image out of the unrectified image coordinates (u_{lr}, v_{lr}). The process of calculating the rectification maps is explained for the left camera. The same process can be used for the right camera. First, (u, v) is transformed to camera coordinates (x_c, y_c, z_c) where $z_c = 1$ and $K = K'_l$ with Equ. (2.7). Then the rectification is reversed with Equ. (2.17), the distortion model is applied with Equ. (2.10), and finally the camera coordinates are transformed to image coordinates with Equ. (2.5) where $z_c = 1$. The resulting rectification maps,

$$mapx(x, y) = u_{lr} \qquad mapy(x, y) = v_{lr}, \tag{2.19}$$

can be used to rectify the stereo image pair with

$$dst(u_l, v_l) = src(mapx(u_l, v_l), mapy(u_l, v_l)). \tag{2.20}$$

Figure 2.8 illustrates the backward transform of rectification.

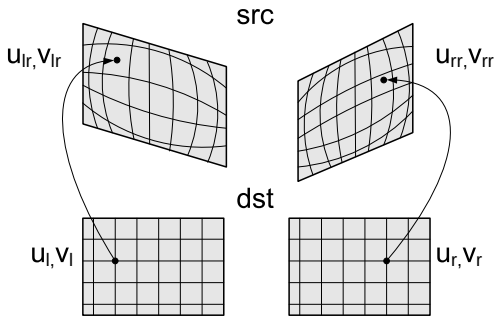


Figure 2.8: Backward transform of rectification

2.5 Summary

In this chapter we presented the fundamentals of stereo vision. A stereo vision sensor consists of two digital cameras mounted in parallel. The cameras capture the scene and a matching algorithm searches for the pixel correspondences to finally reconstruct the 3D information for each matched pixel. The horizontal displacement is called disparity and is indirectly proportional to the depth of the projected scene point. The disparity range is used to define the target depth range, e.g. from infinity (disparity 0) and the minimum distance (maximum disparity). The alignment of the cameras is described with the epipolar geometry. Corresponding pixels always

lie on their epipolar lines. Once the epipolar geometry is known, the images can be rectified. The corresponding pixel search effort reduces a lot because the rectified epipolar lines are aligned in parallel to the image rows and corresponding pixels have the same v-coordinate. The camera parameters and the epipolar geometry can be calculated with a camera calibration and can be used to reconstruct the 3D points in respect to the left or right camera coordinate system.

In the next chapter, we will discuss state-of-the art stereo matching approaches and analyze them according to the given requirements.

Chapter 3

Related Work

Stereo matching algorithms try to solve the correspondence problem between the two pixels representing the projection of a single scene point on the left and right image planes. To do this, all possible matching candidates have to be analyzed and the best matches have to be chosen. First, we explain in Section 3.1 the possibilities of calculating the matching probabilities, how the best match can be chosen out of all candidates, and how the best matching disparities can be optimized. Then we give possibilities how the results of stereo matching algorithms can be evaluated and compared with others in Section 3.2, followed by a compilation of real-time stereo systems in Section 3.3. Finally, we analyze the system requirements in terms of stereo matching algorithms and reason the algorithm decision for the proposed stereo vision sensor in Section 3.4.

3.1 Stereo Matching

There are two main groups of stereo matching algorithms: feature-based and area-based algorithms. The first try to find proper features, such as corners or edges, in the images and match them. The second try to match each pixel independently of the image content. Feature-based algorithms result in a sparse disparity map because they only calculate 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.

The techniques described here are restricted to area-based algorithms because this work attempts to obtain a dense disparity map. A good summary of stereo matching algorithms can be found in the work of Brown et al. [16] and Scharstein and Szeliski [17].

Basically, an area-based stereo matching algorithm is built up as follows: First, preprocessing functions are applied, e.g. a noise filter. This step is not mandatory because modern digital cameras have good noise characteristics. Second, the matching costs for each pixel at each disparity level within the disparity range are calculated. The matching costs determine the probability of a correct match. Mathematically, it is a similarity measurement between two pixels. Third, the matching costs for all disparity levels are aggregated within a certain neighborhood window (block). Fourth, the disparities are optimized to find the best matches.

Aggregating costs increases the uniqueness of possible matches using the assumption that pixels within the window share the same disparity level. This assumption fails if blocks overlap disparity discontinuities. In such cases the borders become broader because the costs of the foreground object have more impact than the half occluded areas of the background. However, the bigger the block size, the higher the chance for a correct match at difficult areas. Beside the drawback of quality loss at disparity discontinuities, large blocks also decrease the accuracy for small objects. Small blocks increase the quality at object borders and the localizing of matches is more accurate but they can cause more false matches at difficult areas. Chapter 5 gives more details about the impact of different aggregation block sizes. To overcome this problem, Hirschmueller [18], Hirschmueller et al. [19], Fusiello et al. [20] introduced multiple windowing aggregation strategies. The basic idea is to aggregate multiple blocks of different sizes and shapes around the target pixel and then use the blocks with the lowest costs only. Other strategies are to select an appropriate block for each pixel by variation of intensity and disparity [21] or to first estimate an initial disparity and optimize it afterwards [22]. In this approach, the block size and shape is selected by including boundary information. Yoon and Kweon [23] introduced an adaptive support weight approach. The support weight for each pixel is adaptively determined by the use of color similarity and geometric relationship to the reference pixel. With the use of integral images (Veksler [24]), the processing time is independent of the window size, so an adaptive aggregation can also be realized with a low processing time. A good comparison and evaluation of different costs aggregation strategies is given by Tombari et al. [25].

3.1.1 Costs Calculation

In the following, popular costs calculation metrics for the pixel $I_1(u, v)$ in the reference image and pixel $I_2(u + d, v)$ in the corresponding image are discussed. The disparity is defined as d and the resulting costs value as $c(u, v, d)$. Also an $n \times m$ costs aggregation

$$\sum_{i=n}\sum_{j=m} = \sum_{i=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \sum_{j=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \tag{3.1}$$

is included in the metrics.

A very popular method is the sum of absolute differences (SAD) and is calculated with

$$c(u, v, d) = \sum_{n}\sum_{m} |I_1(u + i, v + j) - I_2(u + d + i, v + j)|. \tag{3.2}$$

As can be seen, the costs are the absolute differences of the pixels' intensities. It is simple and can be used for very fast implementation due to the usage of additions and subtractions only. The drawback is that additive brightness differences have a strong influence on the matching quality.

Based on SAD, the metric sum of squared differences (SSD) with the costs calculation is

$$c(u, v, d) = \sum_{n}\sum_{m} (I_1(u + i, v + j) - I_2(u + d + i, v + j))^2. \tag{3.3}$$

Here, the similarity is defined by the squared intensity difference of two pixels. The exponent of 2 applies a higher weight to large errors than to small errors in contrary to SAD where the costs are linear. This makes the SSD more sensitive to outliers. The computational effort increases because of the high number of multiplications.

A metric that makes the costs invariant to additive or multiplicative intensity differences caused by different shutter times, lighting conditions or apertures of the cameras is the normalized cross correlation (NCC) calculated with

$$c(u,v,d) = \frac{\sum_n \sum_m I_1(u+i,v+j) I_2(u+d+i,v+j)}{\sqrt{\sum_n \sum_m I_1(u+i,v+j)^2 \sum_n \sum_m I_2(u+d+i,v+j)^2}}. \tag{3.4}$$

Another similarity measure is the zero mean sum of absolute differences (ZSAD) with the costs

$$c(u,v,d) = \sum_n \sum_m |(I_1(u+i,v+j) - \overline{I_1}) - (I_2(u+d+i,v+j) - \overline{I_2})| \tag{3.5}$$

where

$$\overline{I} = \frac{1}{nm} \sum_n \sum_m (I(u+i,v+j)) \tag{3.6}$$

is the mean intensity of image I. The advantage of this metric is that the offset, caused by different apertures, is reduced by subtraction of the mean value of the images.

A different matching strategy is to first apply a non-parametric local transform to the images. This means that the transform does not rely on the intensity value of the image but on their ordering. In statistics, non-parametric models are models whose structure cannot be defined a priori; thus, it is determined by the data. Such transforms are the Census and the Rank transform introduced by Zabih and Woodfill [26]. Both transforms are based on local intensity relations between the actual pixel and the pixels within a certain window. This relation is given with

$$\xi(p_1,p_2) = \begin{cases} 0, & p_1 \leq p_2 \\ 1, & p_1 > p_2 \end{cases} \tag{3.7}$$

where p_1 and p_2 are pixels in the image. The Census transform uses Equ. (3.7) to create a bit string for each pixel in the image I, as shown in Equ. (3.8) where the operator $\bigotimes$ denotes a bit-wise concatenation and $n \times m$ the window size.

$$I_{census}(u,v) = \bigotimes_{i=n} \bigotimes_{j=m} (\xi(I(u,v), I(u+i,v+j))) \tag{3.8}$$

The costs calculation for Census-transformed pixels is the calculation of the Hamming distance between the two bit strings with

$$c(u,v,d) = \sum_{i=n} \sum_{j=m} \text{Hamming}(I_1(u+i,v+j), I_2(u+d+i,v+j)) \tag{3.9}$$

including an $n \times m$ aggregation. The Hamming distance is the number of different bits of two numbers (bit strings) and is defined as the logical operation

$$\text{Hamming} = ||x \otimes y|| \tag{3.10}$$

where $||x||$ denotes the number of set bits of x. The processing time of Census-based matching strongly depends on the window size of the Census transform.

The Rank transform changes each pixel to the sum of all pixels within a certain window whose intensities are less than the actual pixel's intensity. A Rank transformed pixel is calculated with

$$I_{rank}(u,v) = \sum_{i=n} \sum_{j=m} (\xi(I(u,v), I(u+i, v+j))). \tag{3.11}$$

For costs calculation, an intensity-difference-based metric applied on the transformed pixels can be used.

Zabih [27] mentioned an 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 Equ. (3.7)). The use of certain neighborhoods allows the avoidance of double calculations and reduces the total number of comparisons. The mask configuration in his work obtains 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.

A detailed evaluation of the mentioned costs functions can be found in the work of Hirschmueller and Scharstein [28].

3.1.2 Constraints

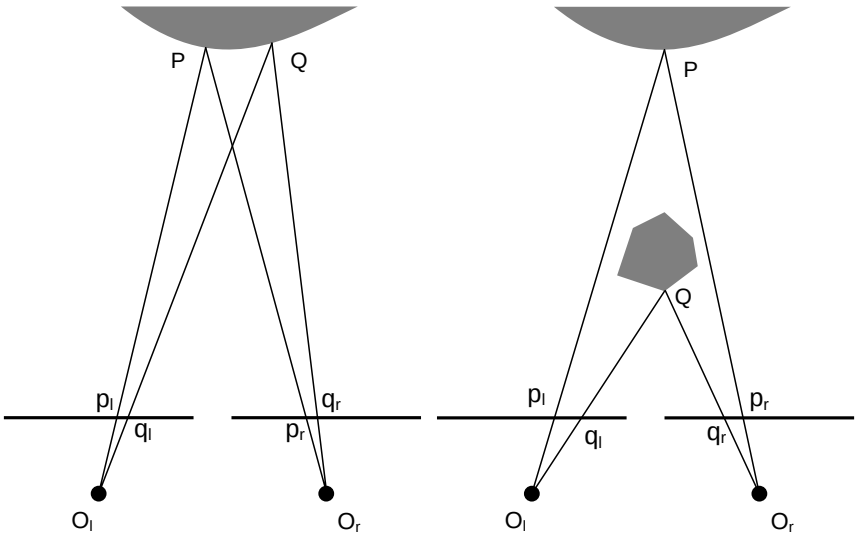
The matching costs define how likely two pixels correspond to each other. However, due to the fact that all possible pixels within the disparity range are analyzed, costs for systematical impossible matches are calculated as well. To reduce the chance for a mismatch, following constraints and assumptions can be used to exclude such matching candidates from the search for the correct correspondence.

Uniqueness

The first assumption is that each pixel in the left image corresponds to exactly one pixel in the right image, as long as it is not occluded. This assumption is violated for objects behind transparent surfaces. A point of an object, e.g. seen through a window glass in the left image, is on the one hand represented by its projection in the right image and on the other hand by the proper point of the window glass. However, transparent surfaces cannot be reliably detected with stereo vision anyway. Pixels which have assigned more than one correspondence can thus be assumed as invalid and eliminated.

Ordering

The second constraint defines the ordering of the matches. It assumes that the ordering of the pixels in one scanline stays unchanged in the corresponding scanline. Let us say that pixel p_l of the left image corresponds to pixel p_r of the right image and q_l to q_r. It is assumed that if p_l is on the left of q_l then p_r is also on the left of q_r. This assumption fails on thin objects in the foreground. Figure 3.1 shows an example of a valid ordering (3.1(a)) and an example of a violation (3.1(b)).



(a) Valid (b) Invalid

Figure 3.1: Examples of the ordering constraint

Smoothness

The third assumption concerns the disparity steps in the near neighborhood of each pixel. Depth changes on surfaces of natural objects tend to have a smooth characteristic. This leads to the constraint that disparity changes have to be smooth as well in the neighborhood of the projected pixels. This assumption obviously does not apply to so called disparity discontinuities at abrupt changes from foreground objects to background or vice versa. The smoothness constraint can be used if disparity discontinuities can be determined and, thus, excluded from the smoothness analysis.

Consistency

The last constraint breaks the ranks in comparison to the others but is the most important in this work. Contrary to the others, the consistency constraint is used to eliminate found disparities and not matching candidates. As shown in Fig. 3.2, the costs calculation can be done from the right to the left and from the left to the right. The consistency constraint says that only disparities with the same value (within a threshold of e.g. 1 disparity step) for both directions are accepted. It is also called left/right consistency check and is a good method to eliminate uncertain and in particular occluded matches.

3.1.3 Disparity Optimization

Once the matching costs of all matching candidates are calculated and false matching candidates are eliminated due to the previously described constraints, the final step is to find the best match out of all candidates for each pixel. At this point, we have to differentiate between local and global approaches. Local methods select the match with the lowest costs (or energy) independent of the other pixels aside

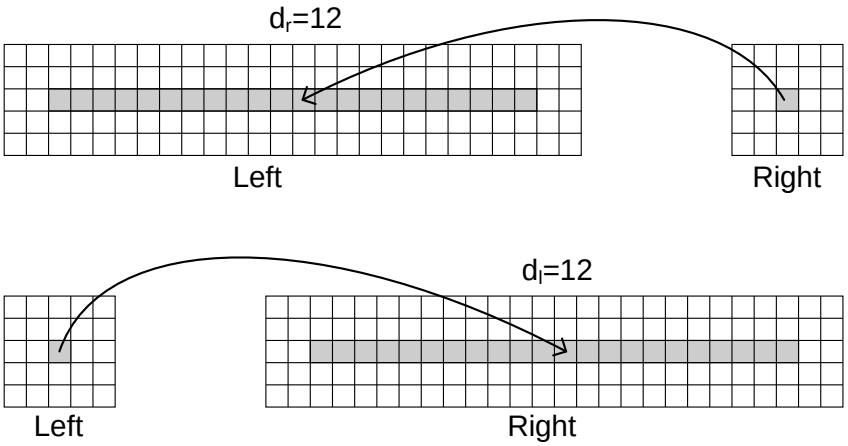


Figure 3.2: Illustration of the consistency constraint; the match is valid because $d_l = d_r$

from the nearest neighbors (because of aggregation). The most common method is a winner-takes-all (WTA) minimum or maximum search over all possible matching costs. Global methods try to minimize the energy of the actual scanline or the whole image to assign a disparity value to each pixel. Beside the matching costs, the energy consists of additional metrics. Commonly, a smoothness term which represents the smoothness of the disparity transitions between neighboring pixels is defined. Such an energy function can be formulated as

$$E(dm) = E_{data}(dm) + E_{smooth}(dm) \tag{3.12}$$

where E_{data} is the function representing e.g. the matching costs and E_{smooth} the function representing the smoothness constraint. The following sections describe global optimization strategies used by many published approaches.

Dynamic Programming

The strategy of dynamic programming (Ohta and Kanade [29], Birchfield and Tomasi [30], Gonzalez et al. [31], Forstmann et al. [32]) 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 pixels in the matching image, specifies the possible neighbors of each matching candidate. The goal is now to find the path with the lowest energy for each scanline. The energy is defined by the matching costs and a smoothness term along each scanline as formulated in Equ. 3.12. If occlusions are determined in the path, a penalty can be added to the energy of the path. It is also possible to mark occlusions from both directions as invalid matches and so imply a left/right consistency check. Due to the optimization of a whole line, the resulting disparity maps suffer from horizontal streaks. An advantage of dynamic programming is that it is computationally efficient in comparison to the other global optimization techniques. There are implementations of dynamic programming with very low processing time, e.g. from Birchfield and Tomasi [30] in the OpenCV library [12].

Graph Cuts

The aforementioned drawback of dynamic programming is that it only considers horizontal smoothness constraints. An approach that overcomes this is graph cuts (Boykov et al. [33], Kolmogorov and Zabih [34]) where vertical smoothness is also taken into consideration. Finding stereo correspondence with graph cuts formulates the correspondence problem as the search for the maximum flow of a weighted graph. This graph has two special vertices, the source and the sink. Between them, nodes are connected with weighted edges. Each node represents a pixel at a disparity level and is associated with the according matching costs. Each edge has an associated flow capacity 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 comparable to the optimal path along a scanline in dynamic programming, with the difference that it is consistent in two dimensions. The computation of the maximum flow is very intensive, so it cannot be used for real-time applications. An implementation can also be found in the OpenCV library [12].

Belief Propagation

Another global disparity optimization approach is belief propagation (Sun et al. [35]). Described as a labeling problem, this iterative strategy uses rectangular Markov random fields to assign the best matching disparities to the pixels. Each node is assigned to a disparity level and holds its matching costs. The belief (probability) that this disparity is the optimum arises from the matching costs and the belief values from the neighboring pixels which represent the joint probability. At each iteration, all nodes send their belief values to the 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 each disparity level. Finally, the best match is the one with the lowest belief values defined by the sum over all four directions. A real-time implementation on a graphics processing unit can be found in Yang et al. [36].

Semi-Global Matching

A rather modern global optimization approach is semi-global matching ([37, 38]). Like dynamic programming, this technique also tries to minimize the global energy with the extension that not only the two horizontal directions are optimized but also vertical and diagonal directions as well. Hereby, an eight or sixteen neighborhood can be used. The costs-path $L_r(p, d_p)$ of the pixel $p := (u, v)$ at disparity d_p in direction r is calculated recursively with

$$\begin{aligned} L_r(p, d_p) := C(p, d_p) + \min(&L_r(p - r, d_p), \\ &L_r(p - r, d_p - 1) + P_1, \\ &L_r(p - r, d_p + 1) + P_1, \\ &\min_{k \in \mathcal{D}} L_r(p - r, k) + P_2) \ , \end{aligned} \tag{3.13}$$

where P_1 is a penalty which is added if the disparities differ by one and the penalty P_2 is added if the disparities differ by more than one ($P_1 < P_2$). $\mathcal{D}$ is the set of

all possible disparities. Afterwards, the costs S are summed up over all paths in all directions r

$$S(p, d_p) := \sum_r L_r(p, d_p) \ . \tag{3.14}$$

A real-time implementation on an FPGA is presented by Gehrig et al. [39] and a fast implementation on a GPU by Ernst and Hirschmueller [40].

Segmentation-Based Matching

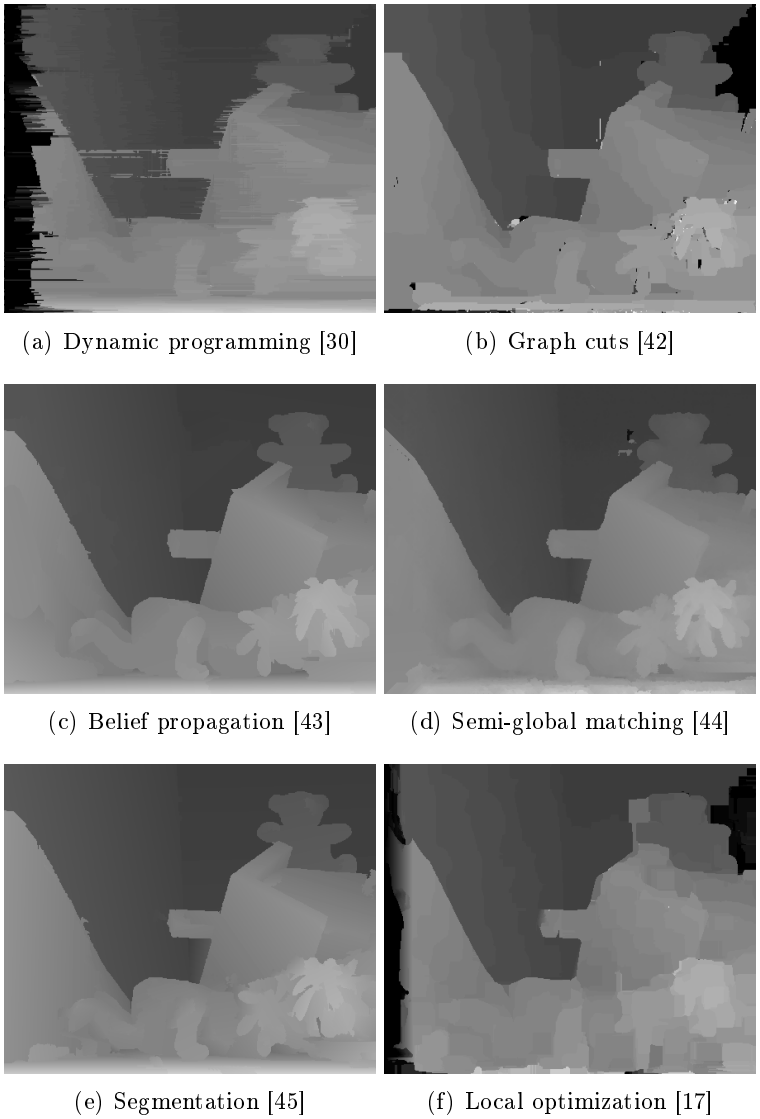
The described techniques up to this point have in common that they try to optimize calculated costs in terms of data and smoothness. Once a disparity map is determined, it can also be treated as initial state for further optimizations such as image segmentation. In this approach, one or both images of the stereo pair are divided into non-overlapping segments. As segmentation criteria e.g. color, intensity, or texture can be used. Mean shift [41] is a commonly used segmentation technique that uses color to create the segments. After assigning each pixel to a segment, the initial disparities are used to fit a proper model (e.g. planar) onto the segment. This model is then used to optimize the initial disparities for each segment. A color segmentation implies, on the one hand, the assumption that disparities of pixels within a segment follow smoothly the chosen model. On the other hand, disparity discontinuities are not expected to be inside a segment. A big advantage of segmentation-based matching is that it increases the matching quality at textureless areas. It also enables the possibility of assigning disparities to half occluded areas as long as these areas are part of non-occluded segments. Of course, the quality of the fitted model strongly depends on the initial disparities. A plane, e.g., cannot be fitted correctly if the data mainly consist of outliers. A disadvantage is that the implied assumptions often fail in real-world environments, such as fitting a plane onto a ball. Another drawback is that the processing time strongly increases, especially when complex models are used. The segmentation makes the processing time scene-depended as well what controverts our definition of real-time capability in stereo vision.

Figure 3.3 gives an example of the matching quality for each stereo matching algorithm mentioned above. The next section introduces how to evaluate the different approaches.

3.2 Evaluation of Stereo Matching Algorithms

The most meaningful method of evaluating stereo matching algorithms is to compare the resulting disparities with their true values (ground truth), which results in a statistical analysis of the true and false matches. To do this, test datasets consisting of the stereo image pair and the appropriate ground truth image are used.

The advantage of this method is that it exactly evaluates the matching quality for the used datasets. The drawback is that the datasets are created under very controlled conditions with high-quality digital cameras, which cannot be found in real-world applications. To realize a better approximation of cameras used in real-world applications, noise and radiometric distortion can be added. The following two sections introduce different groups of datasets for statistical matching quality evaluation.



(a) Dynamic programming [30] (b) Graph cuts [42]

(c) Belief propagation [43] (d) Semi-global matching [44]

(e) Segmentation [45] (f) Local optimization [17]

Figure 3.3: Example disparity maps for different matching strategies

3.2.1 Middlebury Ranking

Scharstein and Szeliski [17] have developed an online evaluation platform, the Middlebury Stereo Evaluation[1] [46], which provides about 40 stereo image datasets. The main feature is an online comparison of submitted area-based stereo matching algorithms. To keep the effort for the developers small, just four of those, shown in Fig. 3.4, are used for the online evaluation. All available datasets are created by illuminating a scene with a series of light pattern to encode the pixels uniquely and make matching trivial. Details of the method used can be found in Scharstein and Szeliski [47].

To evaluate an algorithm on this website, disparity maps of all four datasets have to be generated and uploaded. The disparity maps have to relate to the left stereo image and the disparities have to be scaled by a factor depending on the dataset (Teddy 4, Cones 4, Venus 8, Tsukuba 16). The evaluation engine calculates the percentage of badly matched pixels (false positives) within an error threshold ($\delta = 0.5, 0.75, 1, 1.5 or 2$) 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 - there are over 90 entries up to now - use this platform for evaluation. This gives a good relation on how the developed algorithm performs in comparison to others. The platform is up-to-date and constantly growing. Unfortunately, processing time is not taken into consideration. Of course, this is difficult to realize if a fair comparison should be guaranteed because an independent processing time measurement would be needed.

The main table of the ranking is sorted by the average rank of all twelve resulting values for an error threshold of 1. As mentioned above, the error threshold can be set to 0.5, 0.75, or 1.5 to enable subpixel accuracy as well; unfortunately, only a limited number of algorithms in the ranking support subpixel accuracy. To overcome this problem, Yang et al. [48] have introduced a postprocessing step to enhance the resolution of range images to subpixel accuracy. They have proved that it is suitable for all algorithms of the Middlebury ranking on publication date and published the results on their website[2]. The drawback is that the website is out-of-date at the creation time of this work, so only 25 algorithms are listed. Using the subpixel error thresholds in the Middlebury ranking shows how less actual algorithms refine their results in the subpixel domain during calculation and not as a postprocessing step.

Table 3.1 shows the Middlebury ranking valid during the elaboration of this work. The order of the algorithm accords with the main ranking. For each algorithm in Table 3.1, the matching algorithm class and the processing time is listed. Important is that the algorithm class just gives an idea of the basic matching strategy used. The different approaches vary considerably when they are analyzed in detail. If an algorithm does not fit into a class described above it is marked as *others*. The processing time is taken from the reference papers, as long as it is mentioned.

[1] http://vision.middlebury.edu/stereo/

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

Table 3.1: Middlebury main ranking with algorithm class and processing time. If entries are missing, no paper was available or the information was not provided.

Reference	Algorithm	Av. % of bad pixels	Proc. time
CoopRegion [49]	Seg, PF	4.41	20s
AdaptingBP [43]	BP, Seg	4.23	14s - 25s
DoubleBP [50]	Seg, PF, BP	4.19	> 20s
OutlierConf [51]	Seg	4.60	
SubPixDoubleBP [48]	other	4.39	
SurfaceStereo [52]	Soft Seg	4.06	about 1h
WarpMat [53]	Seg	4.98	10mins
Undr+OvrSeg (ano.)	Seg	5.39	
GC+SegmBorder (sub.)	Seg	4.52	
AdaptOvrSegBP [54]	Seg	5.59	90s
GeoSup [55]	Seg, local, adapt. support	5.80	
PlaneFitBP [56]	Seg, PF, BP	5.78	1fps
SymBP+occ [57]	BP	5.92	45s
AdaptDispCalib [58]	Rank, adapt. window	6.10	
Segm+visib [45]	Seg, PF	5.40	> 20s
C-SemiGlob [38]	SGM	5.76	a few secs.
MultiResGC (sub.)	GC	6.04	
SO+borders [59]	SO	6.03	some mins.
DistinctSM [60]	other	6.14	
OverSegmBP [61]	Seg, BP	6.11	50s
MVSegBP [62]	Seg, BP	6.34	
CurveletSupWgt [63]	other	5.75	
SegmentSupport [64]	Seg, adapt. window	6.44	
LocallyConsist [65]	Adapt. aggr.	6.33	13s
CostAggr+occ [66]	local, other	6.20	49s
RegionTreeDP [67]	DP	6.56	10s
EnhancedBP [68]	BP	6.69	
PUTv3 [69]	other	6.64	
GradAdaptWgt (sub.)	Adpt. support	6.55	
AdaptWeight [23]	Adpt. support	6.67	about 1min
SegTreeDP [70]	Seg, DP	6.82	60 - 70ms
MultiCue (sub.)	local	6.89	
InteriorPtLP [71]	other	7.26	9mins
ImproveSubPix [72]	other	6.90	7s
SemiGlob [44]	SGM	7.50	1,3s
BP+DirectedDiff [73]	other	7.29	
FastBilateral [74]	other	7.31	14s
RealTimeABW		7.90	
CostRelaxAW [75]	other	7.66	11s
BPcompressed [76]	BP	7.53	
RealtimeBP [36]	BP	7.69	16fps
RealtimeBFV [77]	BFV	7.65	57fps
VariableCross [78]	Adapt. support	7.60	60s
2OP+occ [79]	other	7.75	
CCH+SegAggr [80]	Seg	8.07	
VarMSOH (sub.)	other	8.17	
FastAggreg [81]	Color seg. aggr.	8.24	0.2s
GC+occ [34]	GC	8.26	83s
MultiCamGC [42]	GC	8.31	369s
Unsupervised [82]	other	9.45	
SNCC		9.41	
Layered [83]	other	8.24	

ESAW [84]	Adapt. support	8.21	10 - 43.4ms
StereoSONN [85]	other	8.89	100s
RealtimeVar [86]	other	7.85	0.6 to 3fps
AdaptPolygon [87]	other	8.32	
OptimizedDP [88]	DP	8.83	0.2s
ConvexTV [89]	other	9.30	15s
GenModel [90]	other	9.50	
TensorVoting [91]	other	9.25	2.5mins
RealTimeGPU [92]	Adapt. aggr., DP	9.82	0.054s
CostRelax [93]	other	10.6	
ReliabilityDP [94]	DP	10.7	16.6fps
DOUS_Refine		10.6	
TreeDP [95]	DP	11.7	about 1s
GC [17]	GC	11.4	23.6s
CSBP [96]	BP	11.4	0.67fps
BioPsyASW [97]	Adapt. support	11.2	
DCBGrid [98]	Adapt. support	10.9	16fps
BP+MLH [99]	BP	11.1	
H-Cut [100]	GC	11.7	
SAD-IGMCT (sub.)	Census	12.5	
DPVI (ano.)	other	13.3	
DP [17]	DP	14.2	1s
Bipartite		15.4	
PhaseBased [101]	other	15.3	
RegionalSup (sub.)	other	17.0	
IMCT (techn. rep.)	other	16.3	
SSD+MF [17]	SSD	15.7	1.1s
SO [17]	SO	16.6	1.1s
MI-nonpara		15.4	
STICA (Expo Mexico)	other	19.7	
PhaseDiff [102]	BP	18.8	15mins
Rank+ASW		18.4	
LCDM+AdaptWgt [103]	other	19.5	
Infection [104]	other	20.7	

It can be seen clearly that the top performing algorithms all include segmentation-based matching. Most of them segment the input image using color as criterion. Looking at the processing time, the first considerable fast algorithm can be found on rank 11 even if it reaches 1 fps only. All other algorithms supposed to be capable for real-time are located between rank 30 and 61. The average percentage of bad matches gives an indication how close the algorithms lie to each other. The main ordering criterion is the average rank over three evaluations. The top ten performing algorithms vary only by 1.57 percent of bad matches and the real-time approaches between rank 30 and 61 by 4.88. It must be said that the disparity maps used for the evaluation have to be completely dense. That means that matches marked as unsure or occluded have to be extrapolated, which strongly influences the performance in the ranking.

3.2.2 Real-World Scenes

The Middlebury evaluation website offers a good possibility to compare the matching quality of an algorithm to others, but it has its limitations as well. Especially the selection of the datasets is not always meaningful for real-world environments. To overcome this problem, we used the technique of Scharstein and Szeliski [47] in a

previous work [105] to create our own datasets. We selected the scenes according to our opinion of indoor stereo matching difficulties and used standard industrial digital cameras to keep the dataset as realistic as possible. The creation of the datasets works as follows.

First, we illuminate the scene with a series of light patterns from different positions and capture each with the cameras. The light patterns project a code on the scene, thus, the matching becomes trivial because each pixel of the scene has its unique code value (gray code). The resulting disparity map is a first 3D reconstruction of the scene and we use it to calculate the projection matrices between the two cameras and each illumination source position. With these matrices, we calculate the illumination disparities for all pixels. The last step is the combination of all available sets of disparities which results in the final disparity map. Figure 3.5 shows the datasets we created in our office. Unfortunately, due to the lower image quality and resolution of the used cameras, the resulting disparity maps are of less quality than the datasets from Middlebury.

3.3 Real-Time Stereo Vision Systems

This work is related to embedded real-time stereo vision systems, so a list of available systems (not only pure embedded solutions) is given in Table 3.2 and Table 3.3. Due to their high price, only a few embedded solution using an FPGA or an ASIC are compared here. Nevertheless, the selected hardware systems are well chosen because of their importance in the target research field. Ambrosch [106] gives a detailed comparison of hardware-based FPGA stereo systems in his dissertation. Table 3.2 compares local optimizing systems and Table 3.3 global optimizing systems in terms of the stereo matching algorithm, the processing platform and of course the processing speed. The processing speed of the different systems is described with the frame rate in frames per second (fps) on the one hand and, more meaningful, in million disparity evaluations per second with

$$\mathrm{Mde}/s = \mathrm{width} \times \mathrm{height} \times \mathrm{disps} \times \mathrm{fps} \tag{3.15}$$

on the other hand. The advantage of using Mde/s is that it is a processing time metric which includes image resolution and disparity range. For content independent and non-iterative algorithms, Mde/s stays constant for different image resolutions. The processing time evaluation in Section 5 shows that it also depends on the processing platform. Obviously, on platforms which can process a large amount of data more efficiently, the Mde/s increases for higher image resolutions. Not all steps of stereo matching algorithms are influenced by the number of disparities. Thus, in such cases the Mde/s varies for different disparity ranges. Nevertheless, with these facts in mind, Mde/s is a useful performance measure metric for system comparison because it summarizes image resolution and disparity range in one value. In general, a high Mde/s means a high performance of the system. All performance data in the table are directly taken from the authors' papers; the Mde/s is self-calculated with Equ. 3.15. Detailed information about certain pre- or postprocessing as well as optimization steps can be found in the literature. A few of the algorithms can be also found in the Middlebury stereo evaluation ranking, as will be described in Section 5.

Table 3.2: A selection of published real-time stereo vision systems using local optimization techniques, sorted by Mde/s

Reference	Mde/s	fps	Algorithm	Platform
Mobile Robots[107]	996.24	30	SAD	FPGA
VidereDesign [108]	589.824	30	SAD	FPGA
Miyajima and Maruyama [109]	491.52	20	SAD	FPGA + PC
Yang et al. [110]	283.268	11.5	SAD	GPU
Poi [111]	203.98	83	SAD	CPU
Woodfill et al. [112]	187.20	30	Census	ASIC
OpenCV [10]	117.97	66.67	SAD	CPU
Chang et al. [113]	88.473	50	SAD	DSP
Woodfill and Herzen [114]	77.414	42	Census	16 FPGA
Wang et al. [92]	52.838	43	SAD	GPU
Kanade et al. [115]	38.4	30	SSAD	8 DSP
Kimura et al. [116]	38.4	20	SSAD	FPGA
Khaleghi et al. [117]	11.5	20	Census	DSP, MCU
Tombari et al. [81]	8.84	5	segm. aggr.	CPU
Faugeras et al. [118]	7.489	3.6	cross correlation	FPGA
Kosov et al. [86]	0.353	2.15	SSD	CPU

Table 3.3: A selection of published real-time stereo vision systems using global optimization, sorted by Mde/s

Reference	Mde/s	fps	Algorithm	Platform
Gehrig et al. [39]	870.40	25	SGM	FPGA
Forstmann et al. [32]	188.928	12.3	DP	CPU
Ernst and Hirschmueller [40]	165.15	4.2	SGM	GPU
Zhang et al. [77]	100.859	57	BFV	GPU
Gong and Yang [94]	42.11	23.8	DP	GPU
Yang et al. [36]	19.66	16	BP	GPU
Salmen et al. [88]	3.804	5	DP	CPU

Very interesting are the commercially available stereo vision products. A very fast local optimizing stereo vision system is the Mobile Ranger from Mobile Robots Inc. [107]. It uses SAD for stereo processing on a PCI board equipped with an FPGA. Another system is the well known Small Vision System (SVS) from Videre Design, developed by Konolige [108]. It is a fully embedded system with integrated cameras and an FPGA for stereo processing (Stereo on a Chip - STOC). The system benefits from the high processing speed and the small form factor. Konolige also published his algorithm in the OpenCV library [12] where it processes noticeably more slowly due to the fact that it runs on a PC instead of the high parallel hardware. Also a software stereo system is provided by Point Grey Research Inc. [111]. The system includes a stereo head and the processing is done on a target PC. The last commercially available sensor is the only Census-based system and developed by Tyzx Inc. [112]. It is also fully embedded and the stereo processing is done on a dedicated stereo processor chip which does a fast 7×7 Census correlation. The drawback of this system is its high price. Woodfill et al. [114] published an earlier Census-correlation on an FPGA array, as well.

Additionally to the commercially available system, other published systems are

listed in Table 3.2. Khaleghi et al. [117] implemented another Census-based matching algorithm on a DSP and emphasized 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. The other systems use SAD or SSD as correlation criterion.

Most global optimization algorithms are implemented on powerful processing platforms, such as GPU or FPGA, to reach real-time processing rates. This can be seen in Table 3.3 where the highly optimized FPGA implementation from Gehrig et al. [39] of semi-global matching performs best by far. All algorithms are based on dynamic programming, belief propagation or semi-global matching, beside the approach from Zhang et al. [77] which uses bitwise fast voting (BFV). This technique optimizes the disparity map after WTA by voting for the best disparity in support regions around each pixel where it is assumed that pixels within a support region have the same disparity.

In addition to the processing speed, obviously the matching quality is essential. A fair comparison of the stereo matching quality of all the systems listed in Table 3.2 and 3.3 is hard to realize because not all of them use a consistent results comparison, such as the mentioned Middlebury database, or are available for evaluation.
Summarizing, following lessons can be learnt from the systems comparison:

- There are only local optimization algorithms commercially available.
- All except one of them use SAD for correlation.
- The Census-based sensor uses dedicated hardware for real-time stereo calculation.
- Nearly all fast implementations are hardware solutions using an FPGA or a dedicated ASIC.
- Only a very small number of DSP solutions exist.
- Computationally intensive global optimization approaches, such as SGM or BP, need powerful processing platforms such as GPUs or expensive hardware platforms such as FPGAs or ASICs for high speed processing.

3.4 Analyzing the Requirements

In Section 1.1, the requirements for an embedded real-time stereo sensor are stated. The requirements concerning the stereo matching algorithms are summarized as suitability for real-time processing, reliability of the 3D data, possibility of embedded realization and scalability of the design. The analysis of the state-of-the art stereo matching algorithms and real-time stereo systems in Section 3.2 showed that global optimizing techniques promise to deliver high quality 3D data and local optimizing techniques enable high speed realization even on an embedded hardware.

Table 3.4 compares the most promising algorithms in terms of fulfilling the four requirements. Semi-global matching is capable for real-time as long as powerful processing platforms such as an FPGA or a GPU are used. The FPGA platform fails

Table 3.4: Analysis of the requirements

Algorithm	Real-time processing	Reliable 3D data	Embedded realization	Scalable design
SAD (local)	yes	no	yes	yes
Census (local)	yes	no	yes	yes
SGM (global)	no	yes	no	no
Segmentation (global)	no	yes	no	no
Belief prop. (global)	no	yes	no	no

due to the low-costs requirement and the GPU cannot be seen as purely embedded because it needs a PC for operation. For semi-global matching, the paths to each pixel from up to 16 directions has to be saved. So the memory consumption is strongly dependent on the input image dimensions, and, thus it is not scalable.

Segmentation-based matching fails in all algorithm requirements. Due to the model fitting, it is computationally very intensive, which makes real-time processing impossible up to now. As a reminder, in this work, real-time stands for processing speed of 10 fps and above, known, and scene independent processing time. However, the Middlebury evaluation showed that the 3D data of segmentation-based matching is of high quality, using the ranking as criterion, because the model assumption is fulfilled in many datasets. A drawback is that this assumption often fails in real-world environments. This makes the 3D data unreliable for robot applications. The segmentation part also makes the processing time and the memory consumption strongly scene-dependent. Due to that, embedded realization and scalable design suffer considerably.

Belief propagation also proved real-time capability on GPUs only. This challenge could be tackled but the large amount of memory consumption makes it not suitable for embedded realization.

The two remaining local approaches only fail in terms of reliable 3D data. In a previous work [119], we showed that Census-based matching promises better results than SAD. Banks and Corke [120] showed that Census is more robust to radiometric distortions and occlusions. Due to the computational effort of calculating the Hamming distance instead of the absolute difference, it is computationally more intensive than SAD. The state-of-the-art analysis also showed that parallelization is needed for reaching real-time performance with a Census-based approach. The current Census-based approaches also use windows of up to 7×7 only. Larger windows promise to increase the matching quality, as shown later on in Chapter 5. Obviously, larger windows also increase the challenge of realizing real-time performance.

As a consequence of the analysis summarized in Table 3.4, Census-based matching promises to be a good compromise between high-quality stereo matching and real-time performance on embedded systems.

3.5 Summary

In this chapter, we presented the important related work on stereo matching and introduced published real-time stereo systems. There are two main groups of stereo matching algorithms. Feature-based approaches search for certain features in the images and match them. This results in depth information for the found features

only. Area-based methods try to solve the correspondence problem for each pixel in the images, and, thus result in very dense disparity maps.

In area-based algorithms, the matching costs for each possible match are calculated. To increase the probability of correct matches, the costs are aggregated within a block neighborhood. The best matching candidate is found either by local or by global optimization techniques. Local techniques chose the candidate with the highest matching probability. This can be realized at low processing time. Global techniques optimize a global energy function to find the best match. This is computationally very intensive, has a high memory consumption and is, therefore, not suited for embedded real-time systems.

The evaluation of the matching quality of different algorithms can be done with proper stereo datasets. These datasets consist of the stereo image pair and a ground truth image which is a disparity map that holds the true disparities for all pixels. Scharstein and Szeliski have developed an online stereo evaluation platform (Middlebury) which provides several stereo datasets and a ranking to compare the algorithms. It is shown that the top performing approaches are all based on image segmentation, which is, however, not suited for embedded realization. The drawback of the Middlebury evaluation is that the datasets are not that meaningful for real-world environments because the scenes are well chosen to avoid many stereo matching problems such as large textureless areas of unknown geometry. To overcome this, we have created additional datasets. The scenes are taken from an office and show a more realistic environment for robot applications. The drawback is the lower quality of the input images and the ground truth disparity map because the datasets are captured with low cost cameras at uncontrolled conditions.

The analysis of the requirements and the related work showed that a local area-based stereo matching algorithm fulfills the needs, defined in Section 1.1 of an embedded real-time capable stereo sensor. In our opinion, the Census transform is the right choice because it promises to be a well compromise between high quality results and real-time realization.

In the next chapter we will describe our Census-based stereo matching algorithm and the adaptations we have made to make it capable for real-time implementation.

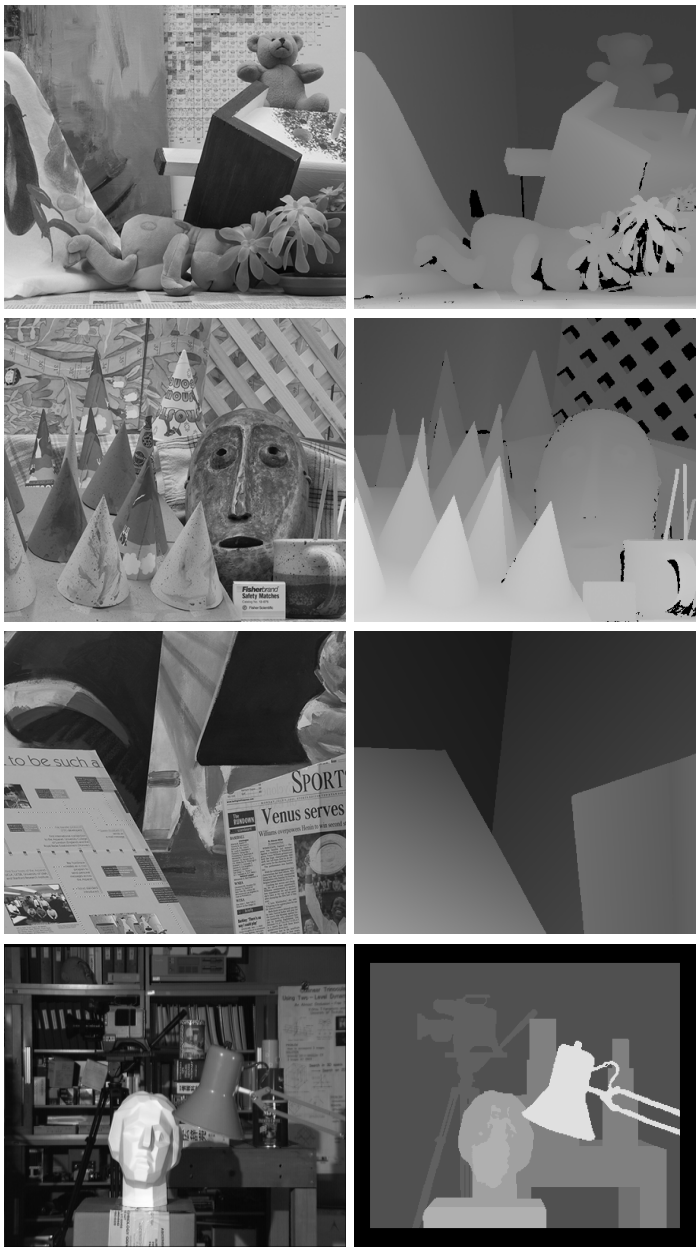


Figure 3.4: Middlebury evaluation stereo datasets [46]; Left: Left stereo images; Right column: Ground truth images; From top to bottom: Teddy, Cones, Venus, Tsukuba

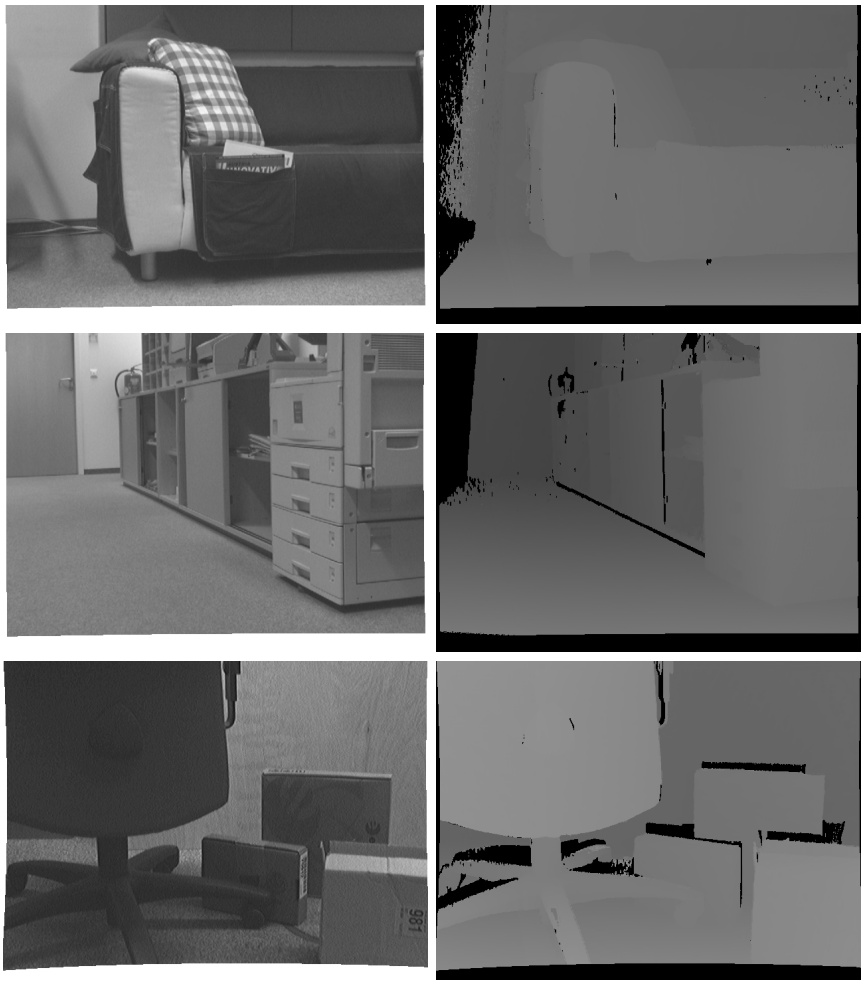


Figure 3.5: Real-world stereo datasets

Chapter 4

Real-Time Census-Based Stereo Matching

The design of a real-time capable Census-based stereo matching algorithm with still high matching quality is the challenge of this chapter. We present the workflow of the algorithm in Section 4.1, where first the stereo image pair is acquired and rectified, described in Section 4.2. Then the images are Census transformed in Section 4.3. The matching costs are then calculated in Section 4.4 and aggregated in Section 4.5. The best matching candidate is searched and a subpixel refinement is done in Section 4.6. In the next step, false matches and occlusions are identified and eliminated. To detect occlusions, we do left/right consistency check in Section 4.7.1 and, to eliminate unreliable matches, we use a confidence and texture threshold in Section 4.7.2. Finally, the 3D reconstruction in Section 4.8 closes the workflow.

4.1 Workflow

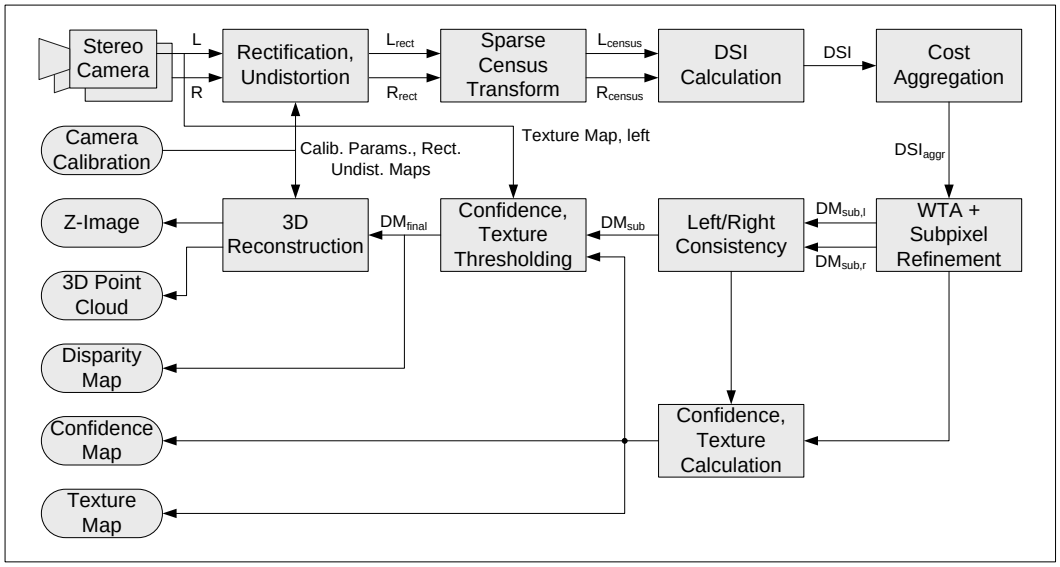


Figure 4.1: Stereo matching block diagram.

Figure 4.1 shows the principle workflow of the algorithm. 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.

4.2 Image Acquisition and Rectification

Step 1 of the workflow in Fig. 4.1 is the image acquisition with the stereo head. The input stereo images are delivered by two digital cameras mounted in parallel with minimal variation. Before a continuous stereo matching can be done, the stereo camera head has to be calibrated offline. We used the calibration method described in Section 2.4. Here, the undistortion and rectification maps for the stereo camera head that hold the image coordinates of the undistorted and rectified images are calculated.

The proposed algorithm uses monochrome input images, so it would be advantageous 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, step 2, undistortion and rectification, follows with

$$L_{rect}(u,v) = L(mapx_l(u,v), mapy_l(u,v)) \tag{4.1}$$

and

$$R_{rect}(u,v) = R(mapx_r(u,v), mapy_r(u,v)). \tag{4.2}$$

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.

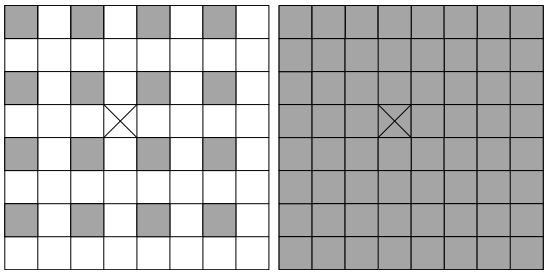
With the calibrated stereo images, the stereo matching can begin. Thus, step 3 of the workflow is the Census transform.

4.3 Sparse Census Transform

Based on the balanced tradeoff between quality loss and performance gain, as will be shown in Section 5.1, a modified Census transform, hereinafter referred to as sparse Census transform, is used. Our approach keeps the mask size large and symmetric and uses only every second pixel and every second row of the mask for the Census transform, as shown in Fig. 4.2 for an 8×8 mask. The filled squares are the pixels used for the Census and the sparse Census transform. Avoiding the double comparisons, like Zabih's approach, is not our key to minimize the processing time. Furthermore, we show that large sparse Census masks perform better than small normal (dense) Census masks with the same weight of the resulting bit strings. Thus, 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. Section 5.1 presents a detailed analysis and an experimental proof of this claim. Our 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 quality improvement for this kind of neighborhood.



(a) sparse (b) normal

Figure 4.2: Census masks

After evaluating different mask sizes, as will be explained in Section 5.1, a mask size of 16×16 was chosen for the 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 4.2 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 Equ. (3.8) and performed with

$$L_{census}(u,v) = \bigotimes_{n \in N} \bigotimes_{m \in M} \xi(L_{rect}(u,v), L_{rect}(u+n, v+m)) \tag{4.3}$$

and

$$R_{census}(u,v) = \bigotimes_{n \in N} \bigotimes_{m \in M} \xi(R_{rect}(u,v), R_{rect}(u+n, v+m)) \tag{4.4}$$

where

$$\xi(p_1, p_2) = \begin{cases} 0, & p_1 \leq p_2 \\ 1, & p_1 > p_2 \end{cases} \tag{4.5}$$

and

$$N = M = \{-7, -5, -3, -1, 1, 3, 5, 7\}. \tag{4.6}$$

The transformed images can now be used to calculate the probabilities of correct correspondences between the source image and the matching candidates in step 4. This probabilities are defined by the matching costs; in case of Census, the Hamming distance has to be used.

4.4 Disparity Space Image Calculation

The calculated costs are stored in the so-called disparity space image (DSI), which is a three-dimensional data structure of size $disps \times width \times height$ and is further used to search for the best matching candidate. For each disparity level there exists a slice of a 3D matrix as shown in Fig. 4.3.

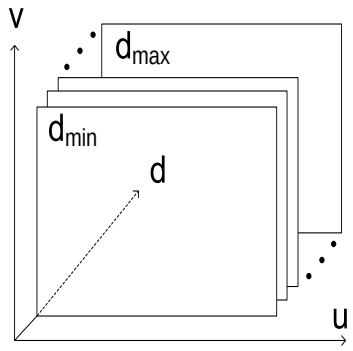


Figure 4.3: Disparity space image

Not all entries of the DSI have to be filled because per disparity level and per image row, 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 no matching candidates, as can be seen in Fig. 4.4. The amount of these pixels increases with the disparity level.

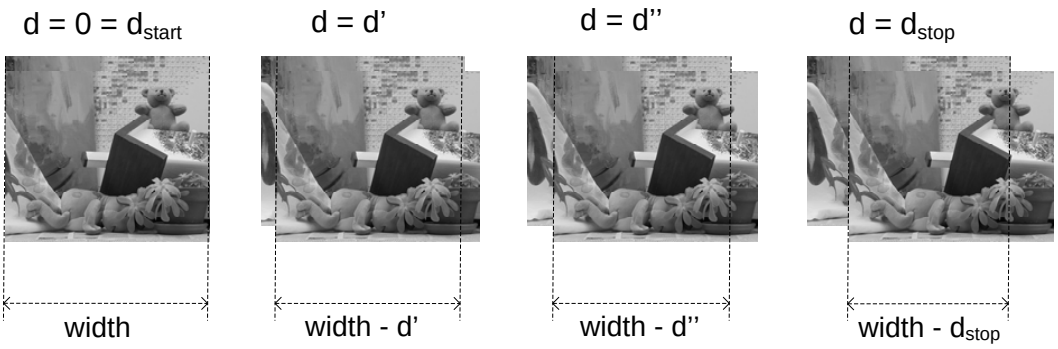


Figure 4.4: The widths of the DSI levels shrink with the disparity.

Particularly with the use of the classic Census transform, the calculation of the DSI is computationally very intensive 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

$$\forall d \in d_{start}, ..., d_{stop} : DSI_d(u,v) = \text{Hamming}(R_{census}(u,v), L_{census}(u+d,v)). \quad (4.7)$$

Figure 4.5 illustrates the calculation of the DSI with Equ. (4.7). The arrows indicate the calculation of the Hamming distance.

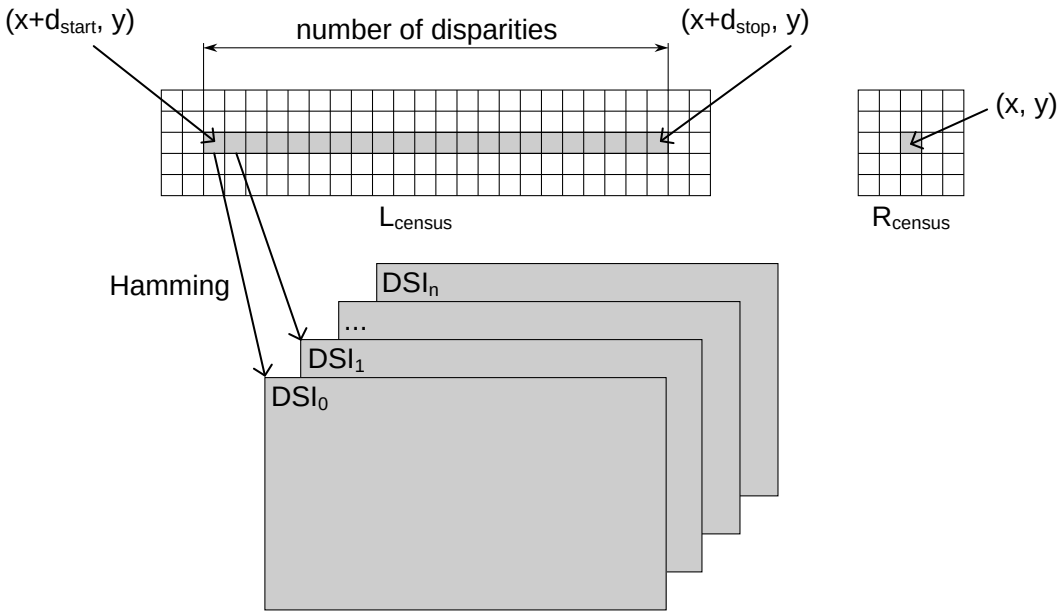


Figure 4.5: Illustration of the disparity space image calculation

4.5 Costs Aggregation

It is assumed that neighboring pixels, except at disparity discontinuities, have a similar disparity level, so a costs aggregation in step 5 increases the uniqueness of matching candidates. 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 5.1). 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 size (convolution) and is calculated with

$$\forall d \in d_{start}, ..., d_{stop} : DSI_{d,aggr}(u, v) = \sum_{n \in N} \sum_{m \in M} DSI_d(u + n, v + m). \tag{4.8}$$

4.6 Subpixel Refinement

After calculating all possible matches, the best matching candidate has to be found in step 6. As explained above, the best match is the one with the lowest costs and it is calculated for a pixel at image coordinates (u, v) with

$$d_{min}(u, v) = \min(DSI_{d_{start},aggr}(u, v), ..., DSI_{d_{stop},aggr}(u, v)). \tag{4.9}$$

Figure 4.6 shows a typical costs function. The 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. 4.6, and its minimum gives the disparity in subpixel accuracy. From now on, $y(d)$ means the cost of a

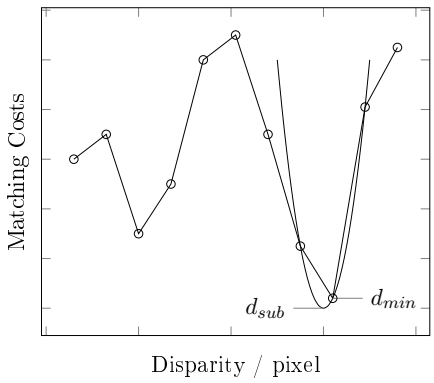


Figure 4.6: Example of a costs function including subpixel refinement.

match at disparity d for a certain pixel (u, v). The subpixel disparity for one pixel is calculated with

$$d_{sub} = d_{min} + \frac{y(d_{min} + 1) - y(d_{min} - 1)}{2(2y(d_{min}) - y(d_{min} - 1) - y(d_{min} + 1))}. \tag{4.10}$$

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

$$DM_{sub,l}(u, v) = d_{sub,l}(u, v) \tag{4.11}$$

and

$$DM_{sub,r}(u, v) = d_{sub,r}(u, v). \tag{4.12}$$

4.7 Disparity Validity

Because of the problems described in Section 2.2 stereo matching algorithms have to face, a not ignorable amount of uncertain matches exists in the disparity map at this point. As mentioned in Section 3.1, disparity discontinuities are a challenge for any kind of block matching approach because the blocks overlay the discontinuities and broadens them in the disparity map. The algorithm described up to now completely ignores this fact. The existence of occlusions is known as well but not properly treated. Repetitive pattern and textureless areas also cause false matches. These are the reasons why the left/right consistency, the confidence, and the texture check are introduced in step 7. The following sections describe these techniques to filter occlusions and eliminate uncertain and unreliable matches. Examples of costs functions with applied validity checks are given as well.

4.7.1 Left/Right Consistency Check

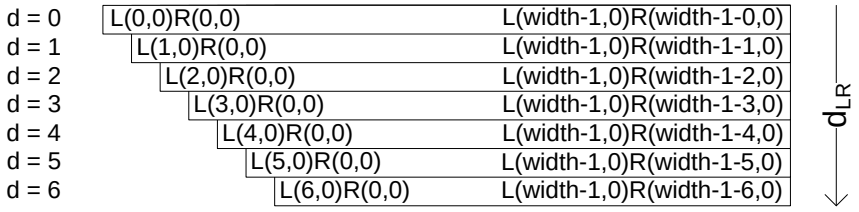
To filter false matches mainly caused by occlusions, a left/right consistency check is applied to $DM_{sub,l}$ and $DM_{sub,r}$ with

$$a = DM_{sub,l}(u, v) \qquad b = DM_{sub,r}(u - a, v) \tag{4.13}$$

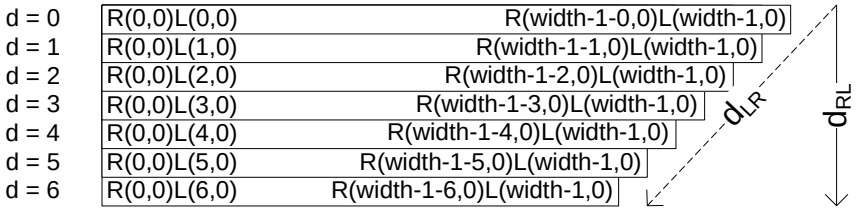
and

$$DM_{sub}(u,v) = \begin{cases} |\frac{a+b}{2}|, & |a-b| \leq 1 \\ 0, & \text{else} \end{cases} . \tag{4.14}$$

It is based on the consistency constraint, mentioned in Section 3.1.2, which says that a match is valid only if it results in the same disparity (within a threshold of 1) if searched form right to left and vice versa. As an implementation detail, the costs calculation is not really done twice. If all possible matching costs are calculated, one straight and one slanted search through all candidates result in both matching directions.



(a) left to right



(b) right to left

Figure 4.7: Disparity space image entries for one line at 6 disparity levels with the WTA search directions

Figure 4.7 shows the DSI entries for a single line in both matching directions. A matching pair of pixels in the left and right image is given with L(x,y)R(x',y') and vice versa depending on the matching direction. Matching a line from left to right works by starting comparing the rightmost pixels of the left and right images. As shown in Fig. 4.7(a), to increase disparity, the coordinates of the pixel in the left image remain unchanged while the x-coordinate of the pixel in the right image decreases. WTA is then performed straight through the possible disparities for each pixel in the left image. Matching from right to left works vice versa by starting the comparisons with the leftmost pixel of the left and right images. Here, the pixel of the right image stays constant and the pixel of the left image moves along the line for the disparity range (Fig. 4.7(b)). As can be seen, the pixel comparisons for both directions are the same but in a different order. The DSI from left to right is a slanted version of the DSI from right to left and vice versa, so a slanted WTA can be performed to use one DSI only.

4.7.2 Confidence and Texture Threshold

A large part of the remaining uncertain matches is eliminated with a confidence and texture threshold. The confidence of a match is defined by the relation of the absolute

costs difference between the best two matches, as denoted with dy in Fig 4.8, 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 with

$$CM(u,v) = min\left(255, 1024\frac{dy(u,v)}{y_{max}}\right). \tag{4.15}$$

For a better visualization, the confidence value is scaled by the factor 1024 and saturated at 255. Figure 4.8 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.

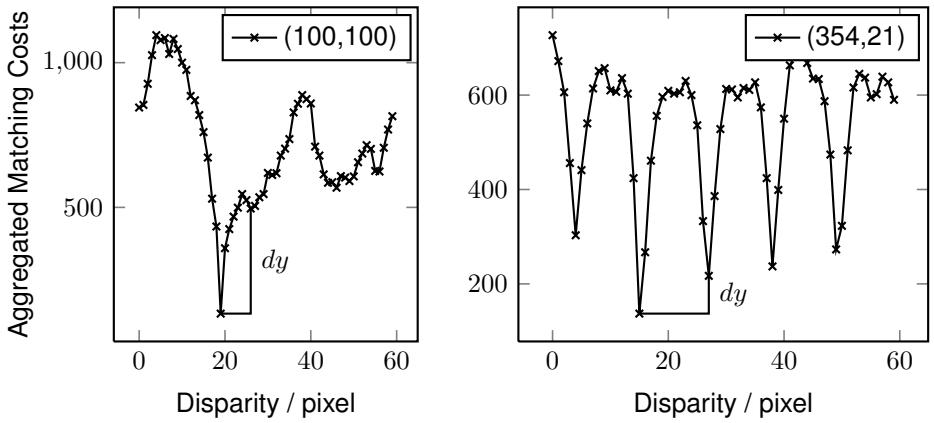


Figure 4.8: Cost functions for two pixels with different confidence values

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 reliable matches, so textureless areas should be masked out. A texture map is calculated for this purpose with

$$TM(u,v) = \frac{1}{nm}\sum_{i=n}\sum_{j=m} L(u+i,v+j)^2 - \frac{1}{nm}\left(\sum_{i=n}\sum_{j=m} L(u+i,v+j)\right)^2, \tag{4.16}$$

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

So in the final step, the confidence and texture maps are applied on the left/right-checked disparity map by a thresholding with

$$DM_{final}(u,v) = \begin{cases} DM_{sub}(u,v), & CM(u,v) \geq \gamma \\ 0, & \text{else} \end{cases} \tag{4.17}$$

and

$$DM_{final}(u,v) = \begin{cases} DM_{sub}(u,v), & TM(u,v) \geq \tau \\ 0, & \text{else} \end{cases}. \tag{4.18}$$

The thresholds are γ for confidence and τ for texture.

4.7.3 Example Costs Functions

The texture check uses the left input image to determine textureless areas. In contrast, left/right consistency and confidence check analyze the costs function to rate the reliability of a match. In the following, four examples of costs functions are presented to clarify the applied checks. The effect of broaden borders will be shown as last point in this section with another two costs functions.

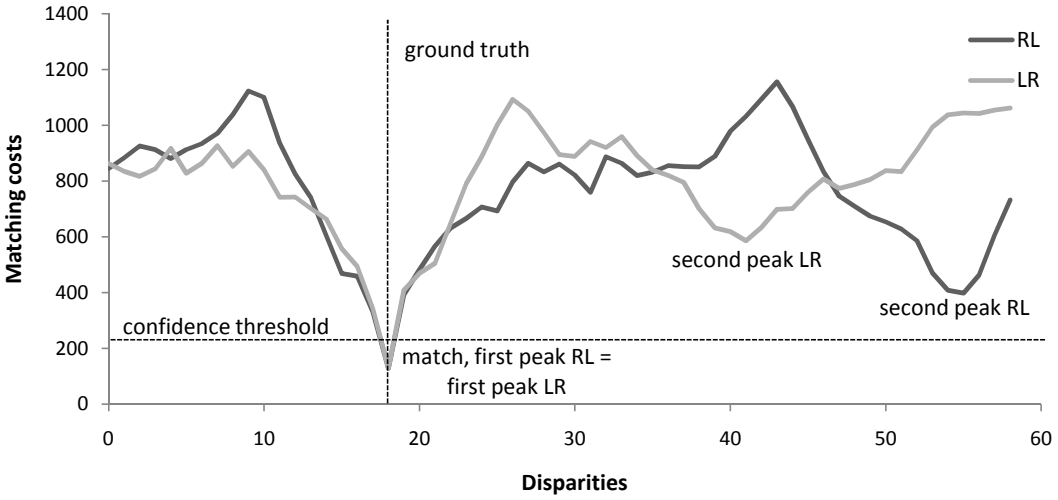


Figure 4.9: Left/right and confidence check are valid

Figure 4.9 shows an example of a costs function for a valid left/right and confidence check. It can be seen that the first peak in the functions of both matching directions is at the same disparity level and, thus, the match is set as valid. The ground truth indicator (vertical slashed line) shows that this match is correct. The horizontal slashed line marks the confidence threshold. If the second peak of a costs function is below this border, the confidence check fails and the match is eliminated. Here, both second peaks are above so the confidence check is valid as well.

The fact that the left/right check alone is not sufficient for reliable matching is shown in Fig. 4.10. Here, the first peak of both matching directions results in the same disparity. The ground truth shows that this decision is wrong. The second peak of both directions is beneath the confidence threshold, so the wrong match is eliminated correctly.

Figure 4.11 shows an example of a costs function where both checks failed. None of the peaks represents the true disparity; thus, the decision is correct.

The confidence check is successfully used (as will be shown in Chapter 5) to eliminate false positives. Contrarily, Fig. 4.12 shows an example where a true positive is eliminated because of a low confidence. The functions consist of three considerable peaks where the one with the lowest costs correspond with the ground truth. The second peaks are very close to the first peak, so the algorithm classified the match as uncertain and, thus, as invalid.

The last two costs functions are examples for false matches at disparity discontinuities. Figure 4.13 shows a costs function for a foreground pixel at an object border. It has a clear peak, the left/right check as well as the confidence check declared it as valid and the ground truth indicator shows the correctness of this decision.

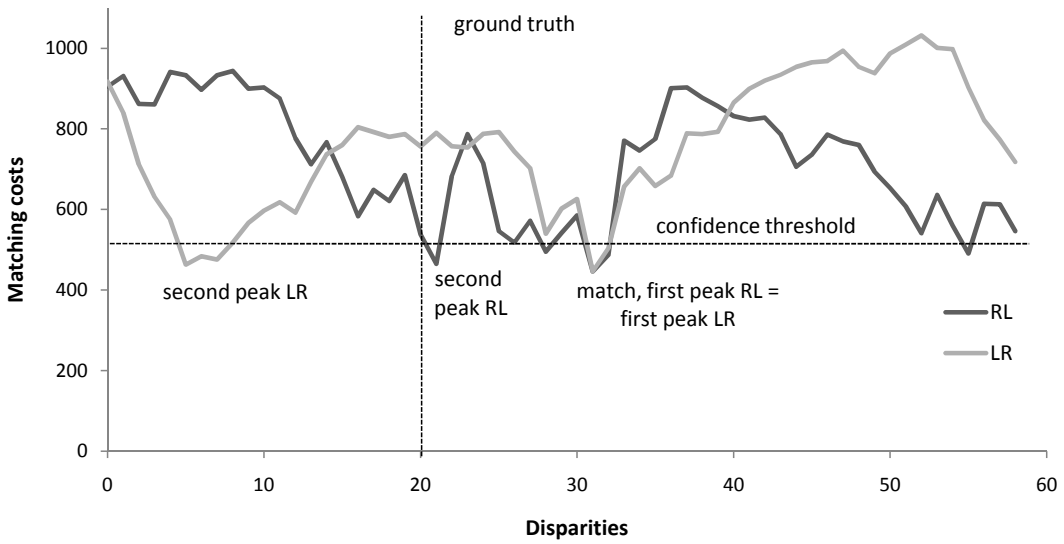


Figure 4.10: Left/right check valid and confidence check failed

Analyzing the costs function of a pixel near this object border at the background in Fig. 4.14 shows that, due to the Census window and the aggregation block, the foreground object still has a strong influence on background pixels. This influence causes the peak at the same disparity level as the pixel on the foreground object. The ground truth shows that the correct disparity is smaller; the pixel belongs to the background. Both checks declared this pixel as correct and, thus, this decision is an example for a false positive which is not eliminated.

4.8 3D Reconstruction

Finally, in step 7, the z-image using (2.1) and the 3D point cloud with respect to the left camera's coordinate system using (2.7) are calculated. If the 3D reconstruction has to be done for an arbitrarily aligned world coordinate system, the proper rotation and translation have to be calculated.

4.9 Summary

In this chapter, we introduced our Census-based real-time stereo matching engine. The decision for this algorithm was made because it has no systematically given constraints to fulfill the real-time capability. It is computationally intensive but delivers high matching quality. The challenge was to design the algorithm in a way to benefit from the matching quality but still enabling fast realization. This is the first part of realizing a fast real-time capable stereo matching algorithm. The optimized implementation follows in Chapter 6.

The algorithm is built up as follows: First, the images are captured by an offline calibrated stereo camera head. The lens distortion is removed and the image pair is rectified. The next step is the sparse Census transform where only every second row and every second column is used. This enables high matching quality with only half

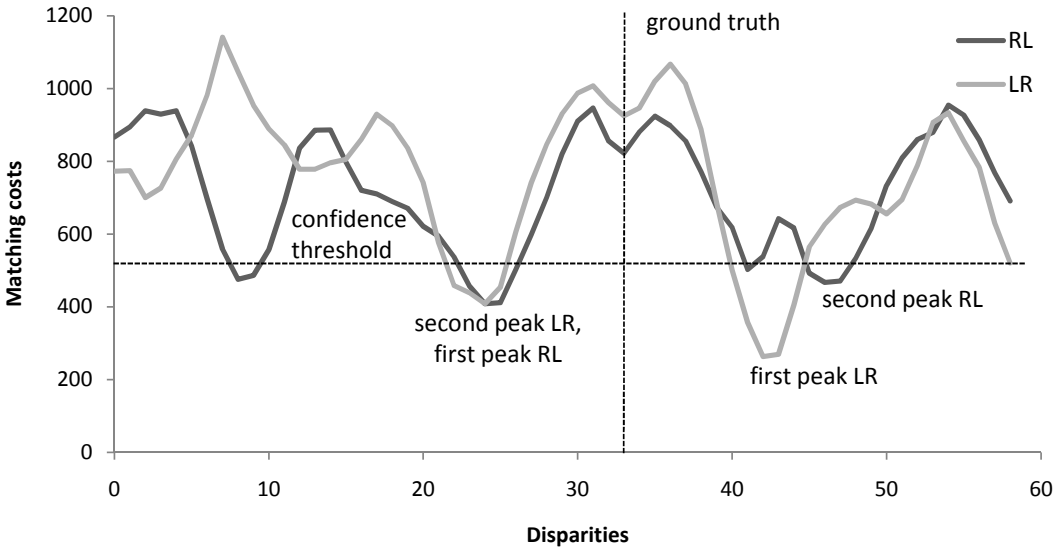


Figure 4.11: Left/right and confidence check failed

of the computational effort. The matching cost of two Census transformed pixels is the Hamming distance. It is calculated for all possible disparity levels. To increase the uniqueness of the correct matching candidate, the assumption that pixels share the same disparity level in their nearest neighborhood is used and, thus, the costs are aggregated within a neighborhood block. A minimum search (winner takes all) is used to pick the best matching candidate out of the whole disparity range. To increase the reliability of the matches, a left/right consistency, a confidence, and texture check is applied. The left/right check can only pass those matches which are consistent over both matching directions. This is a fast and proper way of occlusion detection. The confidence is calculated out of the relation between the costs difference between the two best matching candidates and the maximum possible matching costs. Textureless areas and areas with repetitive texture are difficult to match and, thus, result in unreliable matches. These areas have a low confidence value and can be eliminated in this way. Additionally, a texture filter is used to eliminate pixels at completely textureless areas. These post-matching steps are depicted with six examples of typical costs functions at the end of this chapter.

The algorithm has certain parameters, most important the sparse Census mask and the aggregation block size, to adjust. In the next chapter, we will evaluate all important parameter settings in terms of processing time and matching quality. Furthermore, we will evaluate our algorithm using the Middlebury stereo database and two publicly available stereo matching algorithms.

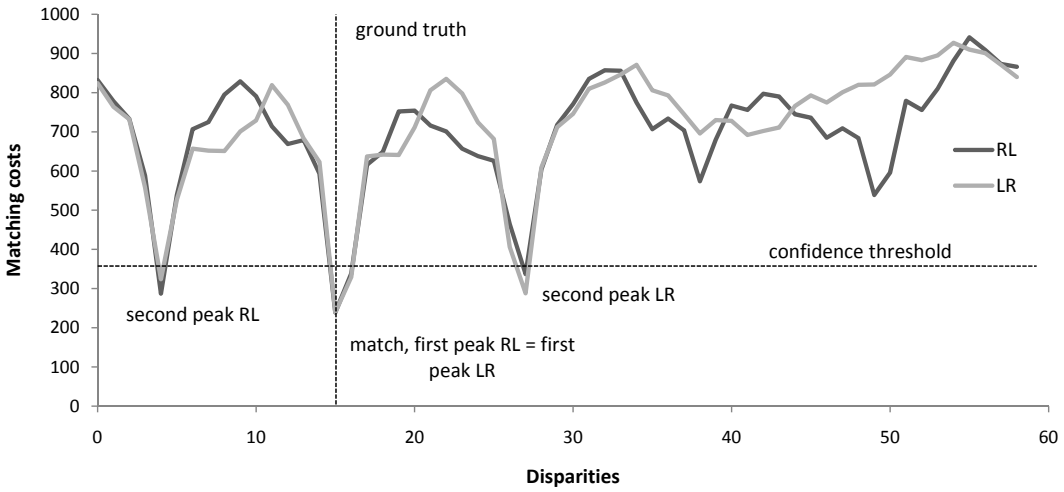


Figure 4.12: Left/right valid and confidence check failed

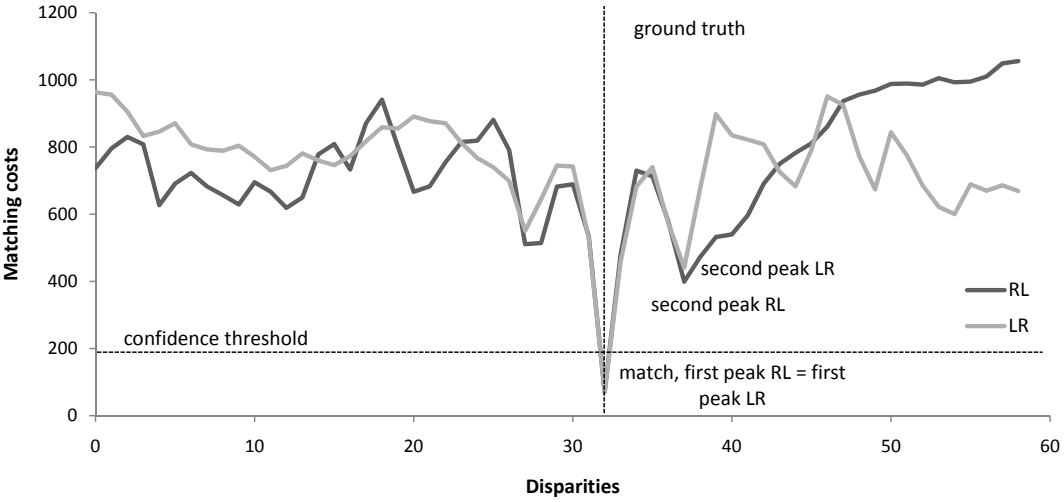


Figure 4.13: True match at an object border

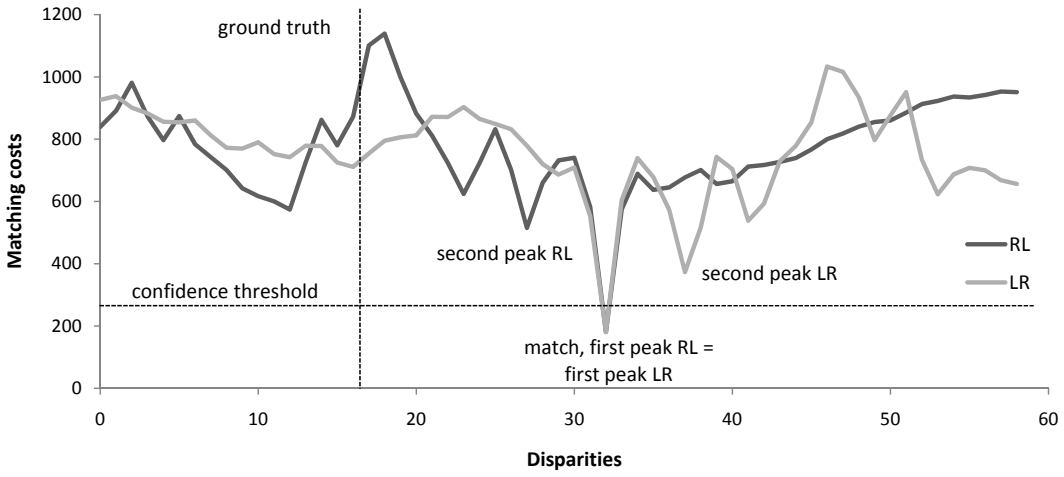


Figure 4.14: False match at an object border

Chapter 5

Evaluation

In the former chapters, we declared the proposed algorithm to be a good compromise between matching quality and processing time. To prove this claim, we now introduce a detailed evaluation of the matching quality by ground truth comparison with two state-of-the-art algorithms and the approaches from the Middlebury database. Additionally, we provide a comparison with other algorithms in terms of processing time.

The proposed algorithm uses a non-parametric transform which, means that it does not rely on the intensity values but on their ordering inside a certain window. Non-parametric does not mean that the transform has no parameters. It solely means that the structure of the model is not defined a priori. In the case of Census, this means that the structure of the bit string relies on the ordering of the pixels and not on their intensity values.

However, there are several parameters in the proposed algorithm, such as the sparse Census window size and the aggregation block size, which have to be set properly. They have an important impact on the performance in terms of matching quality and processing time of the algorithm and have to be treated with care. In this chapter, we first explain the parameter impact on the matching quality in Section 5.1, then we compare the proposed algorithm with state-of-the-art approaches using the Middlebury datasets and real-world scenes in Section 5.2.

5.1 Parameter Impact and Matching Quality

This section analyzes the impact of the algorithm parameters described in Chapter 4 in detail. The goal is to find a proper Census mask size, aggregation block size and suitable confidence and texture thresholds for the target applications. Additionally, we will present the advantages of the use of a sparse Census transform. To find the most suitable parameters, the matching quality but also the processing time play the key role. As reference for the matching quality, we use 31 datasets from the Middlebury stereo evaluation website ([17, 47, 121, 122]). These datasets are captured with low-noise and high-resolution cameras that allow the high quality of the ground truth images. Unfortunately, this is not close to the expected environment in the target application. To overcome this, we additionally overlay the datasets with random noise. Thus, for the proposed experiments, we analyze the matching quality always twice, once with the original and once with the noisy datasets. Figure 5.1

shows one example dataset with its original left image, the noisy left image and its ground truth image.

As evaluation criterion for the matching quality, we use the average percentage of the true positives ([tp]=%) over all 31 datasets on the one hand with

$$tp = \frac{100}{P_{nz}} \sum_{u,v} (|DM(x,y) - GT(x,y)| \leq \delta), \tag{5.1}$$

where δ is the error threshold, DM is the final disparity map, GT is the ground truth image, and P_{nz} is the number of pixels that are non zero in the disparity map and the ground truth image. The latter means that they have to be found by the algorithm and a proper ground truth value has to exist. The true positives rate the accuracy of the results. On the other hand, we use the average percentage of the correct matched pixels ([total]=%) in relation to the total number of pixels P (with available ground truth values) in the image with

$$total = \frac{100}{P} \sum_{u,v} (|DM(x,y) - GT(x,y)| \leq \delta). \tag{5.2}$$

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, we use an error threshold of $\delta = 0.5$ pixels. In this work, we use a squared window with the anchor point in the middle for the Census mask as well as for the aggregation block. In the following charts, the sizes give the side length of the square.

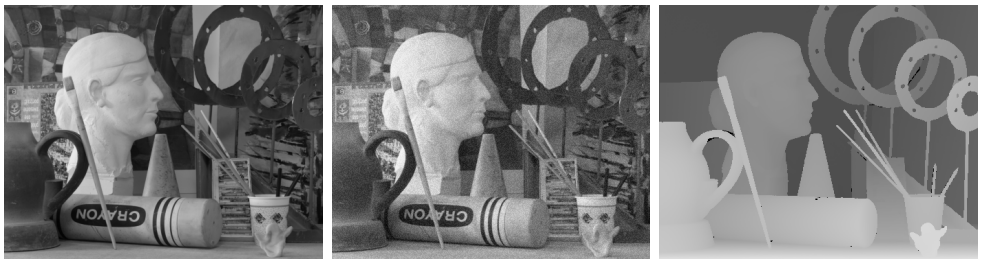
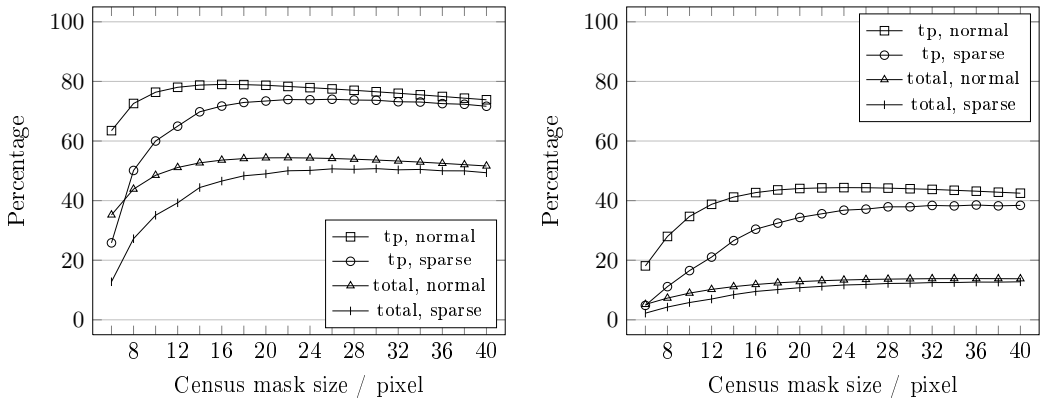


Figure 5.1: Middlebury dataset Art, from Scharstein and Pal [121]; Left image, noisy left image, and ground truth image

5.1.1 Census Mask and Aggregation Block Size

The first parameter to analyze is the Census mask size. Figure 5.2 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 the original datasets results in a maximum at a size of 16×16. The noisy datasets 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. 5.3 where the processing times



(a) original (b) noisy

Figure 5.2: Matching quality for different Census mask sizes without costs aggregation

of normal and sparse Census transforms are compared. We used a plain software solution for this comparison.

After finding a proper Census mask, we analyze the aggregation block size. A well-known problem concerning aggregation block sizes are disparity discontinuities as explained in Section 3.1 and shown in Section 4.7.3. Figure 5.5 shows the matching quality for increasing block sizes at a Census mask size of 16×16. 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 important consequence of costs aggregation is that it closes the matching quality gap (Fig. 5.2) between normal and sparse Census transform.

Figure 5.4 shows the true positives for the noisy datasets of sparse and normal Census masks. We chose the mask sizes in a way 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, we use a block size of 5×5.

After analyzing Census mask and aggregation block size separately, we give a summarizing evaluation of more combinations of both. As can also be seen in Fig. 5.2, the noisy datasets in Fig. 5.6 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 sizes beyond 10×10. In the real-world application, rather good image quality but difficult scenes are expected, so a compromise of a large Census mask, 16×16, and a relative small aggregation size, 5×5, is used.

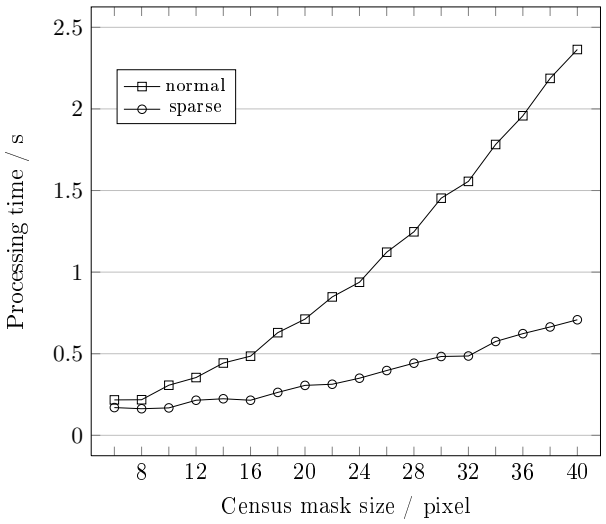


Figure 5.3: Processing time for different normal and sparse Census mask sizes; The times are measured using the plain software version with optimized Hamming distance calculation.

5.1.2 Disparity Discontinuities

Finally, we analyze the matching quality on disparity discontinuities and object borders for different Census mask and aggregation block sizes. As reference, the Middlebury database provides four datasets with disparity discontinuity masks. Thus, only pixels on disparity discontinuities are evaluated. These datasets are also used for the online ranking described later in Section 5.2.5. We set the error threshold to $\delta = 1$ 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. Figure 5.7 shows that 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 again a good compromise.

5.1.3 Confidence and Texture Thresholds

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 5.8 shows the effect on the matching quality with respect to increasing confidence values for a 16×16 sparse Census mask with a 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. Figure 5.8 shows that 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 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.

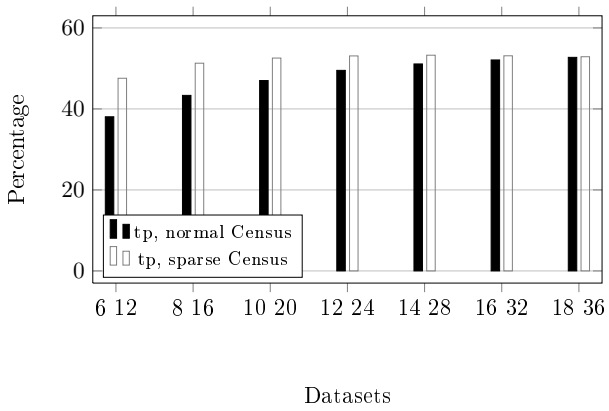
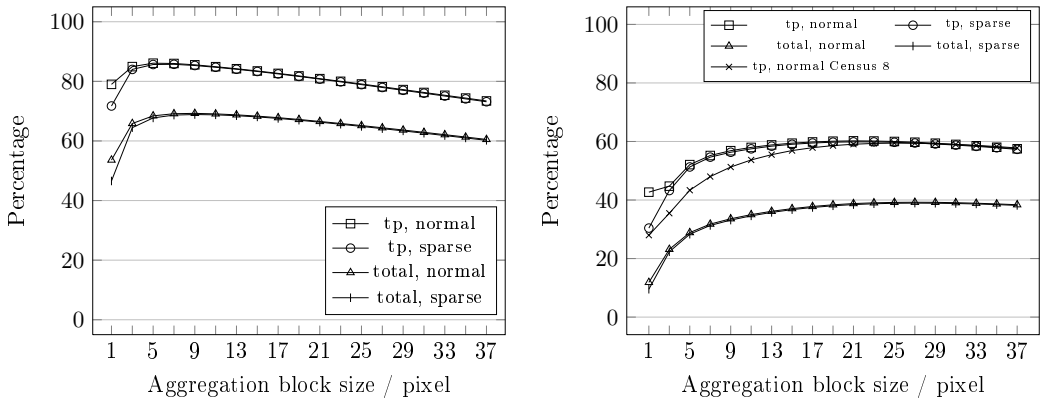


Figure 5.4: Matching quality of sparse Census versus normal Census masks with the same bit string weights

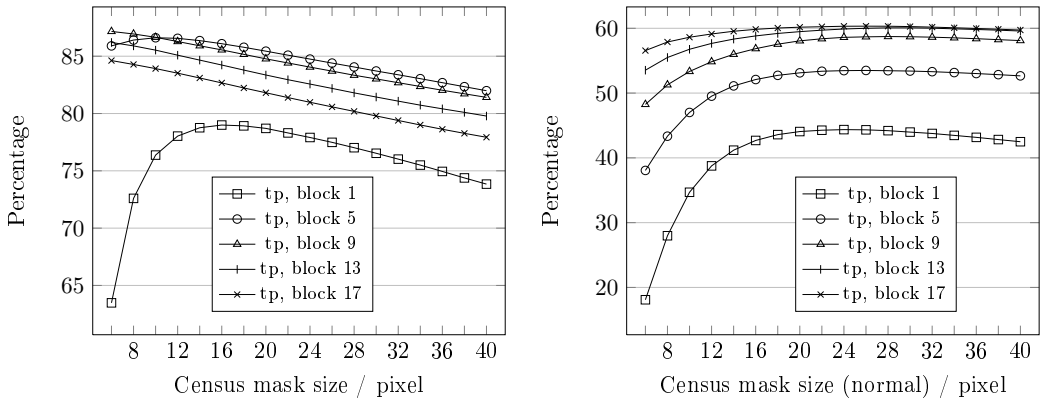


(a) original (b) noisy

Figure 5.5: Matching quality for different aggregation block sizes and a 16×16 sparse Census transform

Finally, the texture threshold has to be adjusted. As explained in Section 4.7.2, it filters textureless areas where correlation-based, locally optimizing matching algorithms fail. 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. Figure 5.9 shows that 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.

As a summary of the parameter impact, we use a sparse Census mask size of 16×16 and an aggregation block size of 5×5 for the optimized reference implementations in Chapter 6. To adjust the algorithm to the specific application, we set the texture and confidence threshold on runtime.



(a) original (b) noisy

Figure 5.6: Matching quality for different Census mask and aggregation block sizes

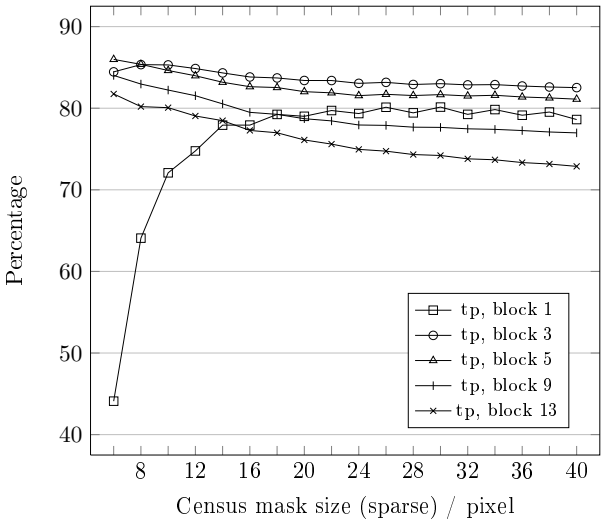
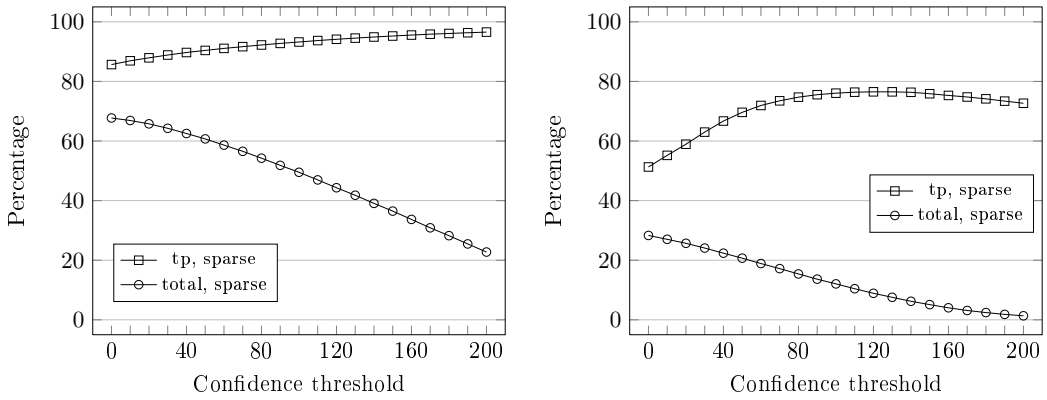


Figure 5.7: Matching quality on disparity discontinuities for different Census mask and aggregation block sizes

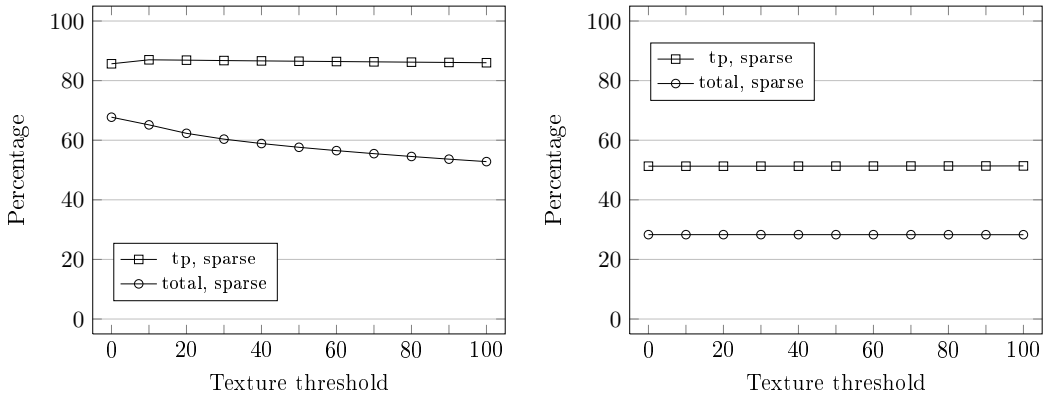
5.2 Algorithm Comparison

We chose two well-known stereo matching algorithms for detailed results evaluation. The first is the SAD block matching algorithm from Konolige [123] available in the OpenCV library [10]. The second is the semi-global Matching algorithm from Hirschmueller [37] also, in a modified version, available in the OpenCV library. The main difference between this version and the original is that the Birchfield-Tomasi [30] metric is used for costs calculation instead of mutual information. Thus, the costs can also be aggregated, what we further do for our evaluations. However, if the aggregation block size is set to 1, the aggregation is turned off. We chose these two algorithms for three reasons: First, both are well-known, and second, both are meaningful representatives of their algorithm class. The SAD for fast local area-based matching and the SGM for fast (semi) global area-based matching. Both are also implemented embedded, SAD on a DSP [123] and SGM on an FPGA [39]. The



(a) original (b) noisy

Figure 5.8: Matching quality for different confidence threshold values with 16×16 Census transform and 5×5 aggregation



(a) original (b) noisy

Figure 5.9: Matching quality for different texture threshold values with 16×16 sparse Census transform and 5×5 aggregation

third reason is that implementations of both algorithms are available in the OpenCV.

For ground truth comparison, we also use the 31 datasets from Middlebury, in the original and the noisy version. Unfortunately, the SAD and SGM algorithms produce a black border of the width of the maximum disparity. Additionally, the SAD produces a black border of the width of the half aggregation block size around the disparity map. One strength of our algorithm is the use of the maximum possible matching range even at the image borders. Nevertheless, to provide a completely fair comparison, the black borders produced by SAD and SGM are artificially added in the results of the Census algorithm.

Our algorithm, as well as both from the OpenCV keep all parameters constant over all datasets. Table 5.1 lists the experimentally chosen values. The uniqueness parameter of SAD and SGM is similar to the confidence parameter of our sparse Census algorithm. It eliminates all matches where the difference between the first and the second minimal costs value are less than this threshold. The higher the threshold, the more unique the residual matches. Additionally, the SGM uses a

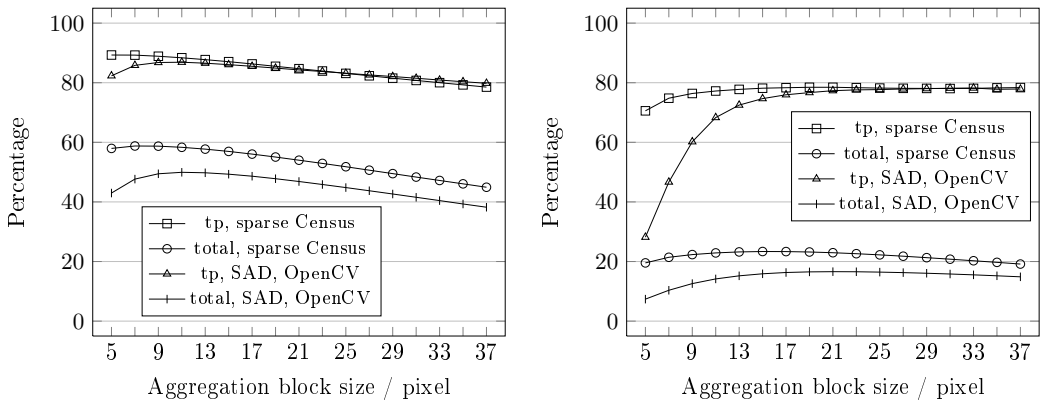
speckle filter for postprocessing. This means that only disparities within the filter kernel which differ maximum for the given difference are set as valid. The penalties of SGM, as described in Section 3.1.3, are set to $P_1 = 7$ and $P_2 = 21$. We always chose the parameters in a way that the true positives are maximized without losing too many correct matches.

Table 5.1: Parameters for the algorithm comparisons

	Postprocessing	
Datasets	Original	Noise
Sparse Census		
Sparse Census mask	16	16
Confidence	35	50
Texture	0	0
SAD		
Uniqueness	40	20
Texture	0	0
SGM		
Uniqueness	50	50
Speckle (Size/Difference)	16/10	16/10

5.2.1 Overall Quality

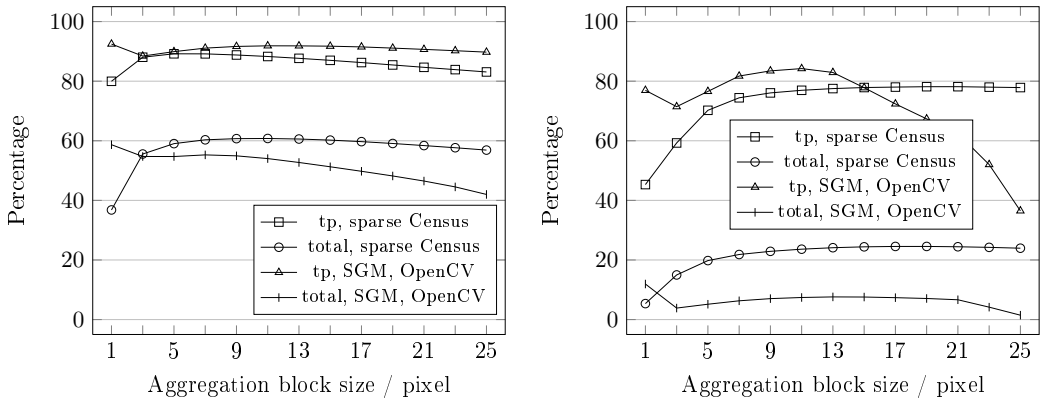
Figure 5.10 shows the true positives (tp) and the total matches (total) of the proposed algorithm in comparison to the SAD with an error threshold of $\delta = 0.5$ for different aggregation block sizes. Both charts of Fig. 5.10 show that the proposed algorithm performs significantly better for small aggregation block sizes and nearly equal for large sizes in terms of true positives. The real improvement is the significant higher percentage of total matches, which means a higher density of the disparity maps.



(a) original (b) noisy

Figure 5.10: Matching quality comparison between the 16×16 sparse Census transform and the SAD algorithm for different aggregation block sizes

Figure 5.11 shows the true positives (tp) and the total matches (total) of the proposed algorithm in comparison to the SGM with an error threshold of $\delta = 0.5$ for different aggregation block sizes. For the original datasets, SGM performs best without costs aggregation, as originally published, but stays nearly constant up to a block size of 31×31. Our algorithm is only slightly below in terms of true positives but delivers a higher number of correct matches (total). For the noisy datasets, SGM performs best with a block size around 11×11 and strongly decreases the performance for large blocks.



(a) original (b) noisy

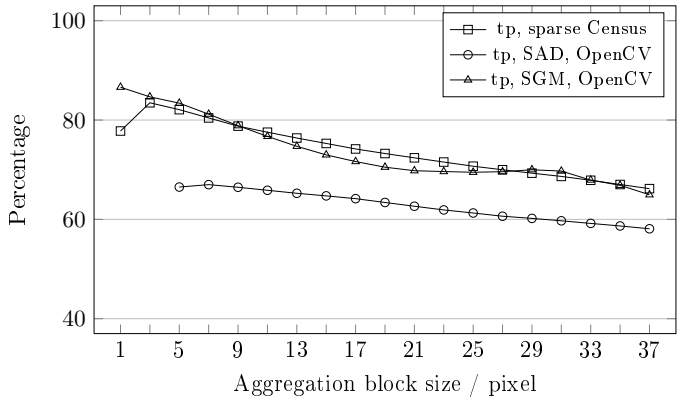
Figure 5.11: Matching quality comparison between the 16×16 sparse Census transform and the SGM algorithm for different aggregation block sizes

As a summary of the overall matching quality, our proposed algorithm performs significantly better than SAD for the original as well as for the noise datasets. The best matching configuration of SGM and sparse Census differs only a little in terms of true positives with a benefit of SGM. Sparse Census delivers for the original as well as for the noise datasets more correct matches in relation to the total number of pixels.

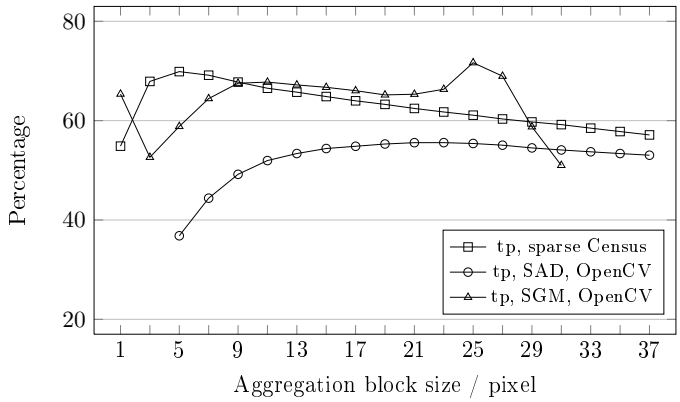
5.2.2 Disparity Discontinuities

An advantage of the Census transform is the robustness on disparity discontinuities because of a good outlier tolerance as described by Woodfill et al. [124]. Figure 5.12 shows that the matching quality at object borders is clearly better than using SAD. It also shows that it performs only slightly worse than the SGM approach which has one of its strengths in good performance on object borders. This is due to the fact that not local neighborhoods only are taken into consideration but also global connections in form of scanlines in 8 directions. Thus, the impact of the local matching window, which causes the wrong matches, decreases. The experiments with the original images show that 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 noticeable

and the number of true positives begin to decrease. SGM performs for the noisy datasets at object borders again best with a costs aggregation of about 11×11. As done before in Section 5.1, we set the error threshold is to $\delta = 1$ for this analysis and abandon postprocessing.



(a) original



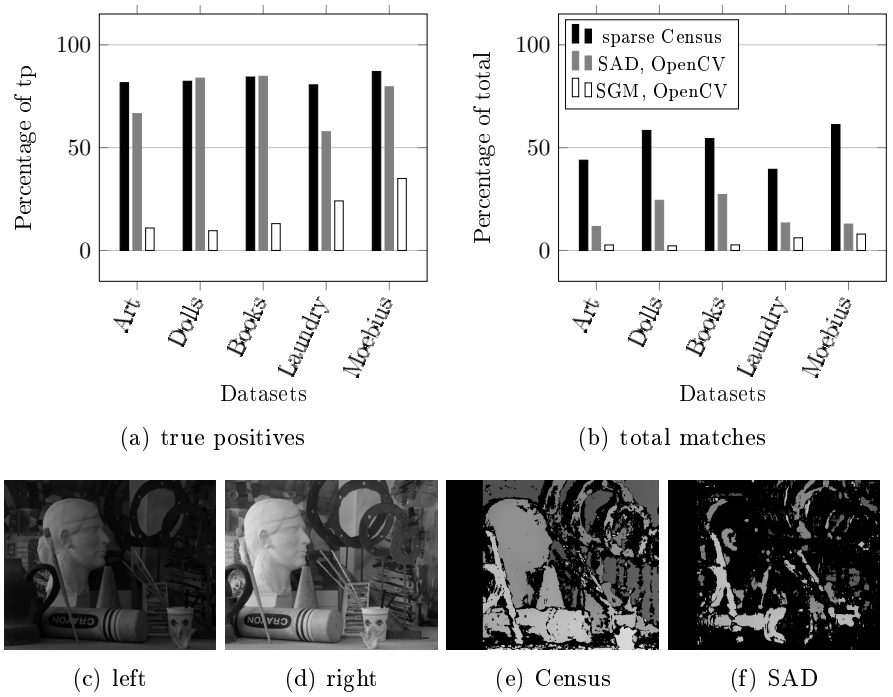
(b) noisy

Figure 5.12: Matching quality comparison between the 16×16 sparse Census transform, the SAD, and the SGM on disparity discontinuities

5.2.3 Brightness Differences

In real-world environments, it can easily happen that the left and the right camera images suffer from different illuminations. A reason for this can be, e.g., independently operating autoshutters. Hirschmueller and Scharstein [122] evaluated many costs functions in terms of different illumination in their work and 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 are independent of constant gain and brightness differences in the stereo pair, similar results for the Census transform are expected. Figure 5.13 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 outperforms the SAD. The SGM approach deals worse with these datasets as can be seen in the charts.



(a) true positives (b) total matches

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

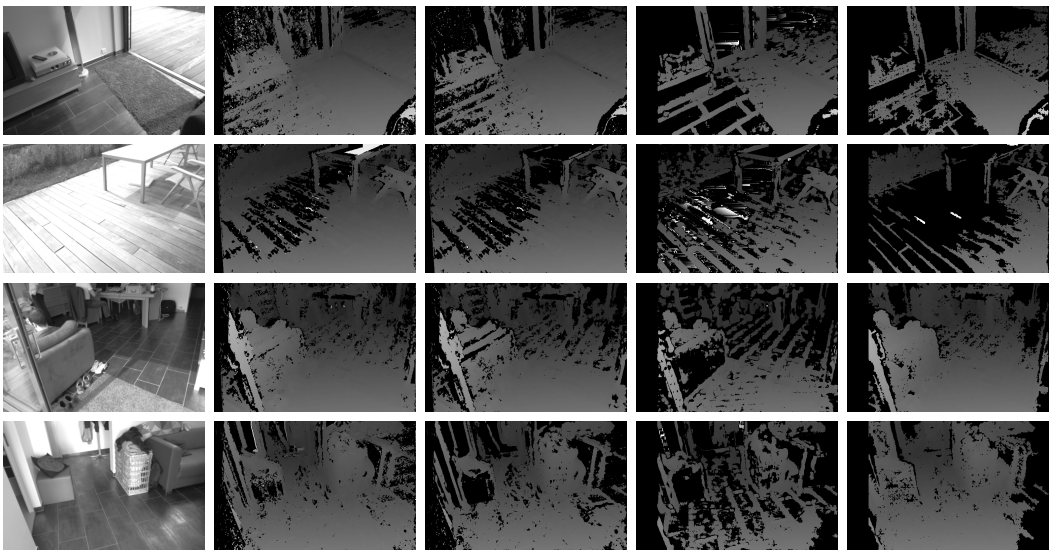
Figure 5.13: Matching quality comparison between the 16×16 sparse Census transform with 5×5 aggregation and the 11×11 SAD for scenes with illumination difference between the stereo pair

5.2.4 Real-World Scenes

For visual comparison of the matching quality, Fig. 5.14 shows four real-world scenes. We performed the aggregation block size selection with the charts in Fig. 5.10 and Fig. 5.11. For each algorithm, we chose the best matching configuration for the original scenes. The proposed algorithm has a 5×5 aggregation block size, the SAD a block size of 11×11, and the SGM is done without costs aggregation. The residual parameters can be found in Tab. 5.1. Additionally, we show the disparity maps of the sparse Census algorithm without confidence and texture thresholds. It can be seen that false matched pixels are filtered out well and apparent true matches are kept valid.

5.2.5 Middlebury Ranking

As explained earlier in Section 3.2.1, the Middlebury stereo database [46] provides a huge number of stereo image datasets consisting of the stereo image pair and the appropriate ground truth image. Four of these datasets (Fig. 3.4) are used to evaluate stereo matching algorithms and to compare the results with many others



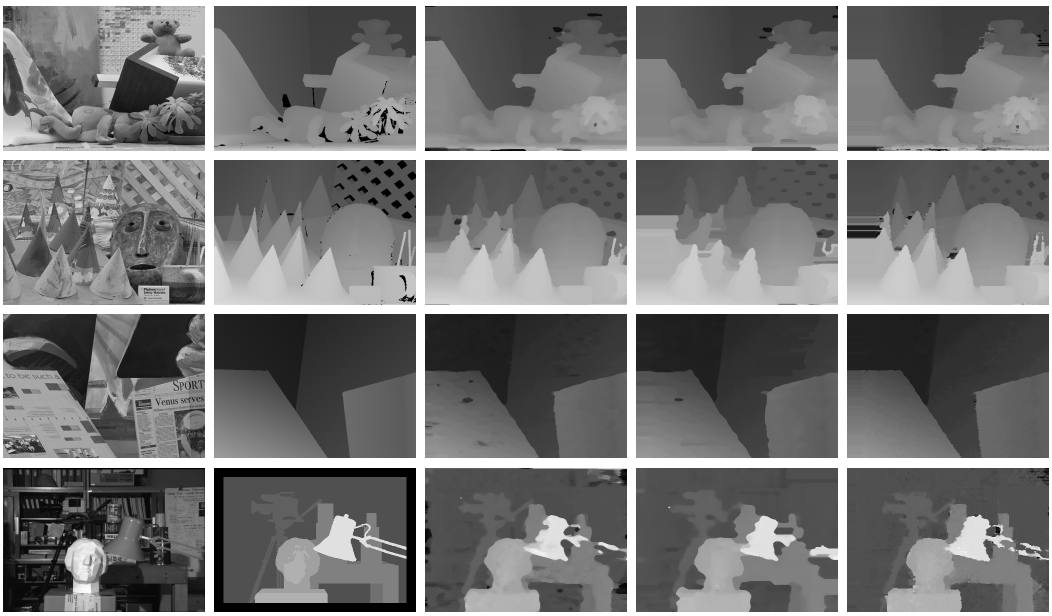
(a) Left image (b) Census 16 (c) Census 16t (d) SAD 11t (e) SGM

Figure 5.14: Real-world scenes; From left to right: Left stereo image, Census 16 (16×16 sparse Census mask, 5×5 aggregation, without thresholds), Census 16t (16×16 sparse Census mask, 5×5 aggregation, confidence 40 and texture 0), SAD 11t (SAD with 11×11 block size and uniqueness threshold 40), SGM (without aggregation and uniqueness threshold of 50)

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

Figure 5.15 shows the four evaluation datasets and the resulting disparity maps of sparse Census and the SAD as well as the SGM from Section 5.2. A sparse Census mask of 16×16 is the main configuration of the algorithm but another configuration, with a Census mask of 10×10, is used for the Middlebury evaluation. Another 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. 5.6 in Section 5.1 shows that smaller Census masks are well suited for high quality input images. For SAD, an aggregation of 11×11 is used and SGM is done without aggregation and postprocessing. Additionally, a 9×9 median filter is applied as a postprocessing step for Census 10 and SAD. We chose these parameters in such a way as to achieve the best possible rank on this 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 postprocessing, the processing time is not negatively influenced.

As a reminder, Table 3.1 lists the actual ranking. 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 the evaluation, we mention only algorithms declared capable of real-time or near real-time, or at least faster than one second. As Table 5.3 will show later on, the meaning of real-time and the corresponding



(a) Left image (b) GT (c) Census 10t (d) SAD 11t (e) SGM

Figure 5.15: Middlebury stereo datasets [17, 47]; From left to right: Left stereo image, ground truth, Census 10t (10×10 sparse Census mask, 3×3 aggregation, confidence 40 and texture 0), SAD 11t (11×11 block size, uniqueness 40 and texture 0), SGM (without aggregation and postprocessing)

processing time is interpreted differently by the authors of the algorithms. We also evaluate the SAD and SGM algorithms from Section 5.2 even if they are not in the permanent table. Of course, due to the fact that they produce the black border on the left side of the image, they suffer more from extrapolating. The processing times for published algorithms are taken from the literature, the SAD and SGM algorithms and the proposed algorithm are measured by ourselves.

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 $\delta = 1$ and is shown in Table 5.2. Figure 5.16 and Fig. 5.17 give a visual comparison of the disparity maps. As can be seen, only one of the real-time algorithms ranks among the top 15. The others, including the proposed algorithm, rank in the middle and bottom positions. In addition, we give the average percentage of bad pixels over all twelve columns. This value is meaningful because shows how close together the algorithms lie. The distance between the best real-time algorithm and the proposed algorithm is only 3.95 percent.

Interesting is the comparison of the processing times in Table 5.3. We 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 only slightly worse in terms of 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.

In Section 5.1 and Section 5.2 the error threshold is set to $\delta = 0.5$ because the

Table 5.2: Middlebury main ranking (error threshold 1) with real-time algorithms only; The SAD and SGM are not included in the online table, that is why we printed the absolute instead of the average rank. While writing this thesis, the main ranking consists of a total of 90 (SAD and SGM 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** all	**Venus** all	**Teddy** all	**Cones** all	**Avg.** (%)
PlaneFitBP	12	1.83 19	0.51 15	12.1 22	10.7 35	5.78
SegTreeDP	31	2.76 39	0.60 20	15.2 56	7.86 5	6.82
RealtimeBP	42	3.40 43	1.90 49	13.2 29	11.6 45	7.69
RealtimeBFV	43	2.22 32	0.87 29	15.0 48	12.3 48	7.65
FastAggreg	48	2.11 30	4.75 66	15.2 50	12.6 51	8.24
ESAW	54	2.45 36	1.65 47	14.2 43	12.7 58	8.21
RealtimeVar	57	5.48 60	2.35 52	7.25 5	6.59 3	7.85
OptimizedDP	58	3.78 48	4.74 65	13.9 38	13.7 57	8.83
Prop. Alg.	62	6.25 66	2.42 53	13.8 35	9.54 24	9.73
RTimeGPU	63	4.22 52	2.98 56	14.4 45	13.7 56	9.82
ReliaDP	65	3.39 42	3.48 62	16.9 59	19.9 71	10.7
SGM	66	4.87 66	1.74 50	18.8 71	12.8 60	10.0
TreeDP	67	2.84 39	2.10 50	23.9 69	18.3 66	11.7
CSBP	68	4.17 55	3.11 64	20.2 73	16.5 72	11.4
DCBGrid	72	7.26 84	1.91 56	17.2 66	11.9 51	10.9
SAD	82	7.91 73	3.57 63	22.6 65	17.9 64	17.2

algorithm delivers disparities in subpixel accuracy. The Middlebury ranking also supports subpixel error thresholds but lots of algorithms in the database calculate integer disparities only. Thus, a direct comparison would be unfair. To overcome this problem, we used the postprocessing step, as explained in Section 3.2.1, to enhance the resolution of range images to subpixel accuracy by Yang et al. [48]. The final evaluation step of the introduced algorithm is a subpixel comparison 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%.

However, if we use an error threshold of $\delta = 0.5$ in the Middlebury ranking, our algorithm is ranked on an overall position of 20 followed by the first real-time algorithm on position 40. If additionally only non-occluded areas are taken under consideration, the rank of our algorithm increases to position 12 of all algorithms in the database followed by the first real-time algorithm on position 33.

5.3 Summary

In this chapter, we presented an evaluation of the matching quality of the proposed algorithm and gave a comparison with other algorithms. We provided a detailed analysis of the algorithm's parameters and their influence. We illustrated that espe-

Table 5.3: 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 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 [56]	12	1	512×384, 48	GeForce 8800 GTX
SegTreeDP [70]	31	6.13	384×288, 16	PC Dual Intel Xeron 2.4 GHz
RealtimeBP [36]	42	16	320×240, 16	GeForce 7900 GTX
RealtimeBFV [77]	43	57	384×288, 16	GeForce 8800 GTX
FastAggreg [81]	48	5	384×288, 16	Intel Core Duo 2.14 GHz
ESAW [84]	54	100	384×288, 16	GeForce 8800 GTX
RealtimeVar [86]	57	2.15	384×288, 16	PC 2.83 GHz
OptimizedDP [88]	58	5	384×288, 16	PC 1.8 GHz
Proposed Alg.	**62**	**573.7**	**320×240, 15**	**GeForce GTX 280**
Proposed Alg.	**62**	**169**	**320×240, 15**	**PC Intel Core2 Quad 2.5 GHz**
Proposed Alg.	**62**	**26.4**	**320×240, 15**	**DSP 1 GHz TI TMS320C6416**
RealTimeGPU [92]	63	43.48	320×240, 16	Radeon XL1800
ReliabilityDP [94]	65	23.8	384×288, 16	Radeon 9800 XT
SGM[1]	66	11.11	384×288, 16	PC Intel Core2 Quad 2.5 GHz
TreeDP [95]	68	1-5	Middlebury[2]	n/a
CSBP [96]	68	0.67	800×600, 300	GeForce 8800 GTX
DCBGrid [98]	72	16	480×270, 40	Quadro FX 4800
SAD [10]	82	66.67	384×288, 16	PC 3 GHz

[1] Modified version of [37].
[2] "It runs in a fraction of a second for the Middlebury images."

cially the confidence and texture threshold are strongly dependant on the application and camera. Large Census masks as well as large aggregation blocks increase the matching quality in textureless areas and noisy images. The drawback is that localizing of the matches decreases, especially at disparity discontinuities such as object borders. Thus, we discovered a Census mask size of 16×16 and an aggregation block size of 5×5 as a good compromise between processing time and high quality matching. These parameters are then used for the optimized implementations introduced in the next chapter. We also proved that the use of a sparse Census size significantly decreases the processing time by nearly constant matching quality. Mentionable is that the aggregation step closes the gap between sparse and normal Census masks.

We did a detailed comparison of our algorithm with a state-of-the-art SAD block matching and a globally optimizing SGM algorithm. It showed that the sparse Census transform performs significantly better than the SAD in all measures. Our algorithm performs slightly worse than the SGM in terms of true positives but delivers more correct matches in the original as well as in the noisy datasets. We also achieve nearly the same matching quality at disparity discontinuities as SGM and outperform SAD.

Further, we evaluated the proposed algorithm with the Middlebury database. Taking only approaches into account which declare to be real-time capable, our approach is ranked in the middle using matching quality as criterion but outperforms all others in terms of precessing speed.

In the last chapter, we will introduce four highly optimized reference implementations on different platforms to show the real-time and embedded realization capability of our algorithm.

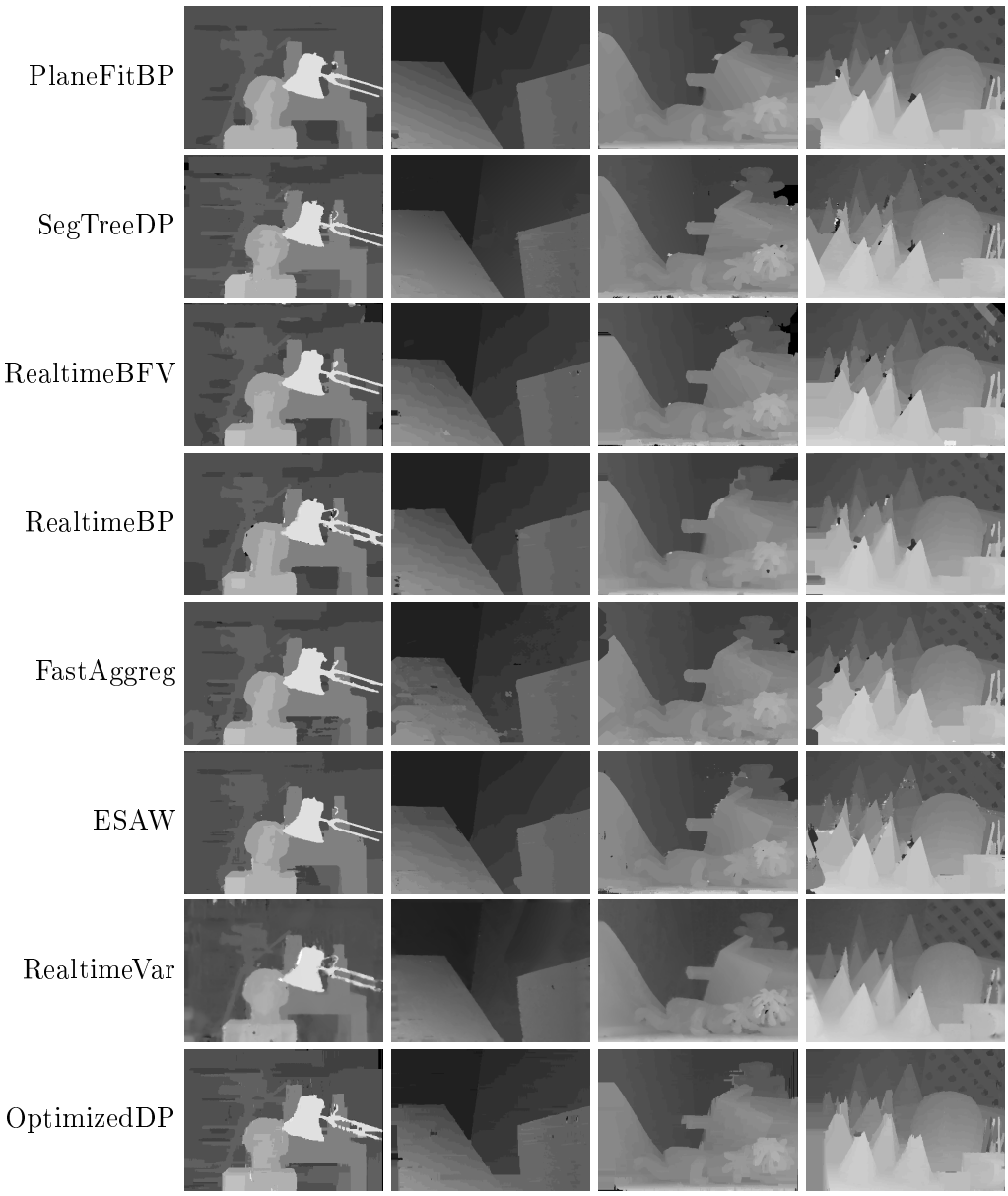


Figure 5.16: Visual comparison of the real-time algorithms on the Middlebury website, sorted by the main ranking in Tab. 5.2, part 1

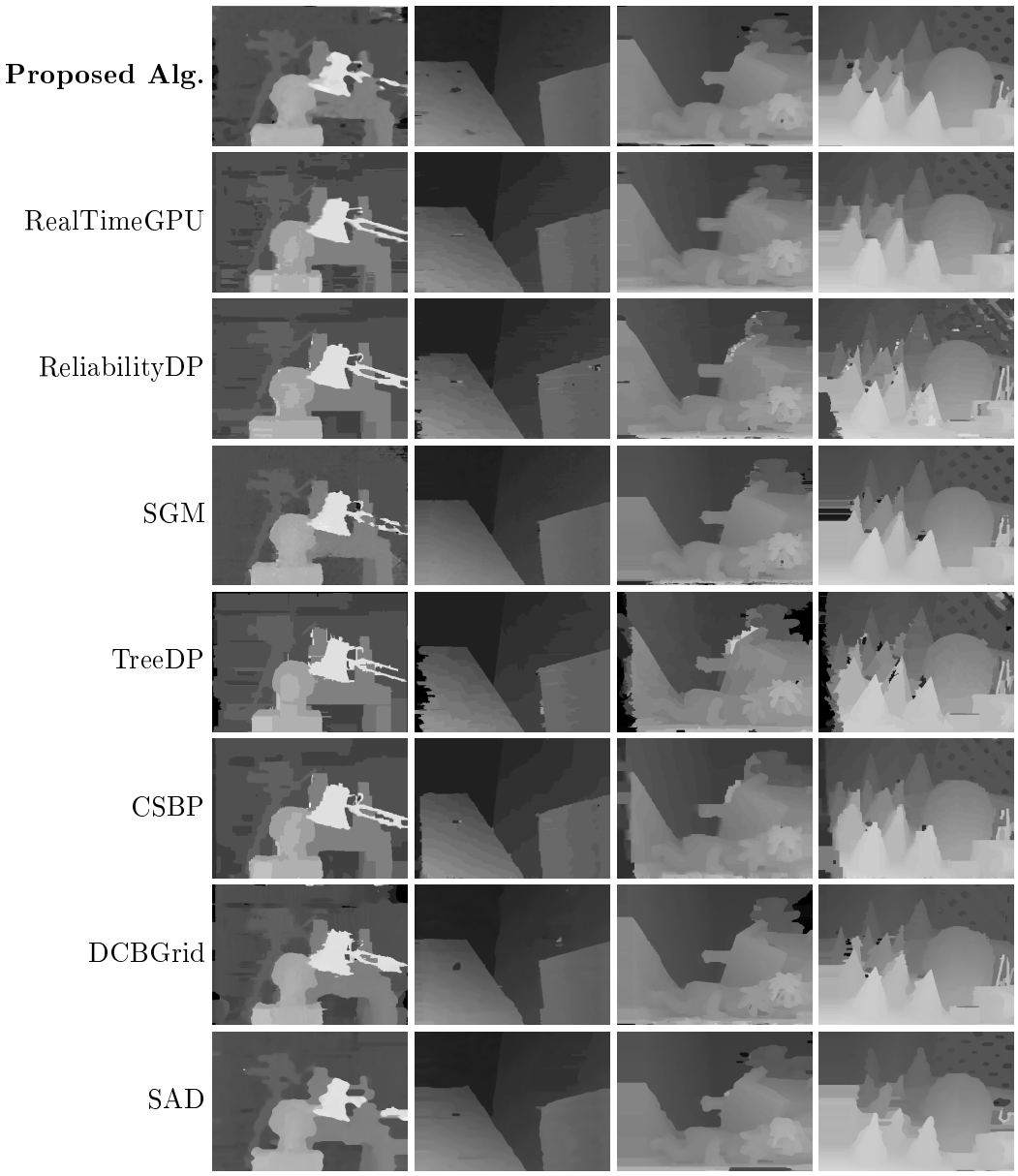


Figure 5.17: Visual comparison of the real-time algorithms on the Middlebury website, sorted by the main ranking in Tab. 5.2, part 2.

Chapter 6

Reference Implementations

A highly optimized real-time implementation of an image processing algorithm is not a trivial task. First, the huge amount of data being processed have to be managed efficiently. Especially on light weight platforms such as digital signal processors, memory is a rare resource. Second, the processing time of the individual pixels has to be low because it is done, e.g., about 6 million times for images with a resolution of 640×480 assuming a frame rate of 20 fps. In case of stereo matching, each pixel is compared with a number of pixels equal the disparity range which can strongly vary depending on the operation distance and range. As explained in Chapter 4, the proposed algorithm uses the Hamming distance of two 64-bit words for costs calculation. For the example resolution above and a disparity range of 60, the Hamming distance has to be calculated about 368 million times per second (Mde/s). This is just the costs calculation; it has to be kept in mind that the costs have to be evaluated and optimized as well to get the final results.

In this chapter, we introduce four reference implementations of the proposed stereo matching algorithm. First, we present a plain and hardly optimized software solution for common CPUs in Section 6.1. Then, we present an optimized software version for multi-core CPUs in Section 6.2, followed by an implementation on three NVIDIA GPUs in Section 6.3. In Section 6.4, we present a version for Texas Instruments DSPs which shows the possibility of real resource-aware and purely embedded implementation. All the implementations have in common that the calibration is performed with the OpenCV library [12] and undistortion and rectification are realized by calculating the transform maps offline and applying a remapping to the images online as described in Section 2.4. Also, all implementations are flexible in terms of image dimensions and disparity levels. Finally, we give a performance and power consumption comparison in Section 6.5.

6.1 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 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 for AMD64 Processors* (AMD [125]). We used this implementation also for the evaluations in Section 5.1.

6.2 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.

6.2.1 Target Platform

The main target platform during the performance-optimized implementation is an Intel Mobile Core2 Duo processor (model T7200) clocked at 2 GHz (Int [126]). This CPU model is commonly used not only in notebook PCs, but also in industrial and so-called "embedded" PCs. We intentionally avoided to optimize our software only for high-end PCs with respect to a broader field of application in the industrial / mobile domain. Beside that, this implementation is applicable for a freely configurable number of processor cores. We will give the detailed performance evaluation of the algorithm on multi-core architectures in Section 6.5.

On the software side, we implemented the proposed algorithm in C with the requirement to keep it portable among MS Windows and Linux. We used Microsoft Visual Studio Compiler 8 on the Windows side and under Linux the GCC 4.3 compiler came to service. Both compilers can deal with basic OpenMP[1] directives which we used to realize the multi-core capability.

6.2.2 Overall Strategies

We use the PfeLib [127] as the back-end for the performance critical functions, which in turn uses parts of the Intel Performance Primitives for Image Processing (IPP) [128] whenever possible. Much of the performance optimizations described in the following sections were actually done in functions of the PfeLib. The library has an open architecture for adding new functions, migrating them to other platforms, and optimizing them for each platform. It also provides components for verification and high resolution performance timing on various platforms.

6.2.3 Algorithm Performance Optimization

A common guideline of our software design process is to consequently encapsulate all computation intensive image processing functions into the dedicated performance primitives libraries (PfeLib and IPP). The remaining part of the program is usually not performance critical and, thus, can be coded in a portable manner. The following sections describe in detail the optimization strategies of the core parts of the proposed algorithm.

[1] http://openmp.org/

Table 6.1: Hamming distance calculation scheme [130].

Pseudocode	Data bits								Comment
a	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	operand a
b	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	operand b
c=xor(a,b)	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	get differing bits
d=and(c, 01010101_2)	0	c_6	0	c_4	0	c_2	0	c_0	1-bit comb mask
e=and(shr(c,1),01010101_2)	0	c_7	0	c_5	0	c_3	0	c_1	shift and mask
f=add(d,e)	$c_6 + c_7$		$c_4 + c_5$		$c_2 + c_3$		$c_0 + c_1$		add bits
g=and(f,00110011_2)	0	0	$c_4 + c_5$		0	0	$c_0 + c_1$		2-bit comb mask
h=and(shr(f,2),00110011_2)	0	0	$c_6 + c_7$		0	0	$c_2 + c_3$		shift and mask
i=add(g,h)	$c_4 + c_5 + c_6 + c_7$				$c_0 + c_1 + c_2 + c_3$				add
j=and(i,00001111_2)	0	0	0	0	$c_0 + c_1 + c_2 + c_3$				4-bit comb mask
k=and(shr(i,4),00001111_2)	0	0	0	0	$c_4 + c_5 + c_6 + c_7$				shift and mask
l=add(j,k)	$c_0 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7$								hamming weight

Sparse Census Transform

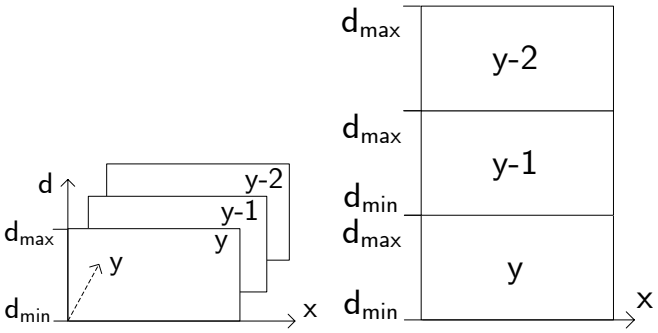
To optimize the Census transform, the SSE provides the `_mm_cmplt_epi8` instruction, which compares 16 pairs of signed 8-bit values at once. The drawback is that it cannot be used directly because the image pixel intensities are unsigned numbers. Pixel values greater than 127 would be interpreted as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

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 by the BitMagic library (Kuznetsov [129]), which is similar to the already mentioned method 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. Table 6.1 explains the optimized Hamming distance calculation for two 8-bit values a and b using pseudocode. We extended the procedure for 128-bit registers achieving now 64 cycles for the Hamming distance calculation.

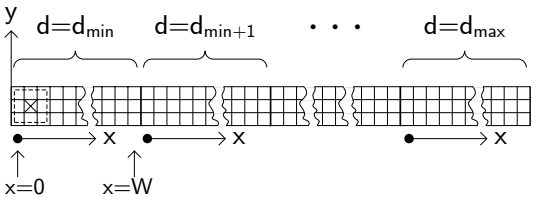
DSI and Costs Aggregation

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 in a way that it can also be used for embedded realization. The chosen method is to process the stereo image pair line-by-line. The standard DSI (Fig. 4.3) is transformed in such a way that one layer of the three dimensional structure 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. Figure 6.1(a) shows the transformed DSI and we call such a slice through the DSI a disparity space layer (DSL). A DSL comprises the costs values for all disparities of an image line; thus, it is a 2D-data structure, which in turn can be treated



(a) Conventional method: storing separate DSLs for each into a common frame buffer

(b) Storing the last n DSLs into a common frame buffer line



(c) The same frame buffer with tweaked image description parameters: common filtering functions are now applicable

Figure 6.1: Memory layout for efficient cost aggregation (example with a 3×3 mask) [130].

with image processing functions. The problem is that the y-neighborhood of costs values is actually spread among different images, which makes it hard to use common optimized filter functions. We faced this by storing the last n DSLs into one single image frame buffer of n-fold height as shown in Fig. 6.1(b). Now, it would be fine to have costs values with equal disparities, but from adjacent y-coordinates, on top of each other. This can be achieved by tweaking the image parameters describing width, height, and the number of bytes per line. We finally get a result as shown in Fig. 6.1(c), which is an image with only n pixels in height, but with D times the width.

We notably achieved this without any movement of pixel data in memory. Now, aggregation means not more than applying a linear filter with all filter mask coefficients set to one and the common divisor also set to one. The filter kernel is indicated in its start position in Fig. 6.1(c). The result of the filtering can be transformed into an "ordinary" DSL reversely. The same method is used in the DSP implementation in Section 6.4 because of its memory awareness.

6.3 Graphics Processing Unit

A graphics processing unit (GPU) is a dedicated graphics rendering device which is used in personal computers, workstations, or game consoles. Modern GPUs are very efficient at manipulating and displaying computer graphics, and their highly parallel

architecture makes them very effective for complex algorithms. An important part for achieving high performance on GPUs is the parallelizing of the algorithm. The better the algorithm can be subdivided into parallel tasks, the higher performance gain can be achieved. Thus, this is the main challenge in this reference implementation.

The GPU can be programmed 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 [131].

For this work, we chose CUDA for the implementation of the optimized GPU software. 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.

6.3.1 Target Platform

We used three different graphic card generations in this work. The first is the Quadro FX 570 and it is based on the G84GL chip generation, the second is the GeForce 9800 GT (NVIDIA [132]), which is based on the G92 chip generation, and the third is the GeForce GTX 280 (NVIDIA [133]) based on the chip generation GT200. Since the CUDA Toolkit only supports NVIDIA graphic cards, we only used these and compared them in Table 6.2.

Table 6.2: Graphic cards used

Specifications	**FX 570**	**9800 GT**	**GTX 280**
Multiprocessors	4	14	30
Processor Cores[1]	32	112	240
Processor Clock (MHz)	460	1500	1296
Registers per core	8192	8192	16384
Peak GFLOPS	44.1	504	933
Memory (MB)	256	1024	1024
Memory Clock (MHz)	800	900	1107
Memory Interface Width (Bit)	128	256	512
Peak Memory Bandwidth (GB/s)	12.8	57.6	141.7
Compute Capability	v1.1	v1.1	v1.3

[1]Each multiprocessor contains 8 cores.

The Quadro FX 570 is an entry level graphics card and is therefore the slowest, in processor as well as in memory clock. It has also the least number of multiprocessors which means that fewer threads can be launched in parallel. 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 overcompensated by more than double the number of processors on the card as well as a 2.4 times larger memory bandwidth. We present a detailed performance comparison between the three graphics cards in Section 6.5.1.

6.3.2 Overall Strategies

The bandwidth between the host and the graphics card via the PCI-Express bus (PCI-SIG [134]) is quite low compared to the graphic cards memory bandwidth as can be seen in Fig. 6.2. Hence, data transfer between GPU and CPU is a major bottleneck.

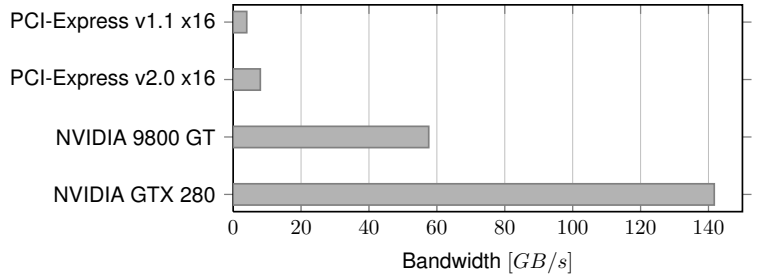


Figure 6.2: GPU memory bandwidth versus PCI-Express bus bandwidth

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 6.3 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 global memory. Access to global memory can also be very fast as long as adjacent memory addresses are accessed. 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 6.3: 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 required, they are also transferred into shared memory as shown in Fig. 6.3. 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.

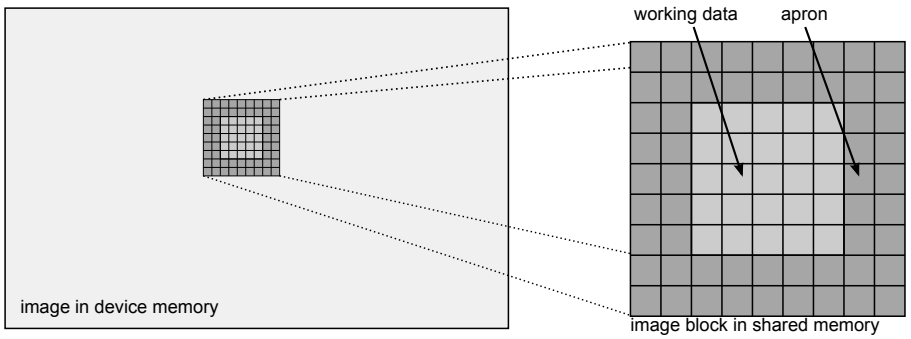


Figure 6.3: Image data are loaded block-wise with a surrounding apron

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.

6.3.3 Algorithm Performance Optimization

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 which is needed because of subpixel coordinates in the maps.

Sparse Census Transform

Even though the image pairs are stored in the texture memory and memory access is cached, to speed up subsequent reads, the data are loaded block-wise, including an

apron, into the shared memory as can be seen in Fig. 6.3. 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 [125]). 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 $\lceil \frac{width}{128} \rceil * 128 * height$ threads. If 128 is not an integer divisor of the image width, the GPU starts a few dummy threads. The fast shared memory can be used efficiently only with this drawback. Each thread evaluates Equ. (4.7) for its pixel.

Costs Aggregation

Costs aggregation can be implemented using convolution. However, 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, we examined three different strategies. In Gong et al. [135], 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 [24]. Figure 6.4 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.

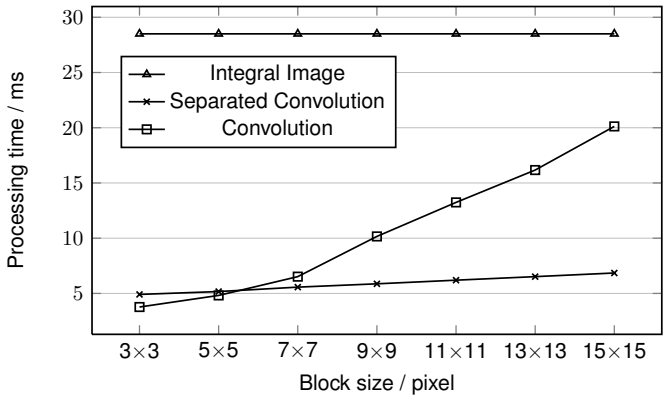


Figure 6.4: The computation time needed for the aggregation with different mask sizes using convolution, separated convolution, and integral images. The runtime was measured on a GTX 280 graphics card. For each computation 50 disparities with an 512×512 image were used.

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 for subpixel refinement.

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.

6.4 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.

6.4.1 Target Platform

The primary DSP platform for the reference implementation is a C6416 fixed-point DSP clocked at 1GHz; its details can be found in Tex [136]. 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 (Tex [137]) 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.

6.4.2 Overall Strategies

The TMS320C64x has no floating point unit. Floating point operations have to be emulated in software. Thus, 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 memory hierarchy of the TMS32C64x DSPs has several stages, namely the fast L1 Caches and the additional on-chip memory (IRAM) for fast access. Further external memory (SDRAM or DDR-RAM) already have much slower access times and 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. The method is called resource optimized slicing with direct memory access (ROS-DMA) and is described in detail in Zinner and Kubinger [138]. Figure 6.5 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.

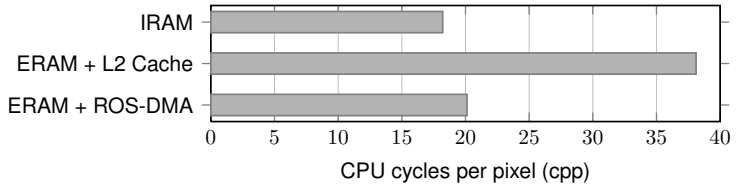


Figure 6.5: Impact of different memory configurations on the performance of the Census transform

6.4.3 Algorithm 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 compilers, a fact remains 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, we used the already mentioned special embedded performance primitives library, the PfeLib. 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. [127].

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, we did the optimizations 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. We verified algorithmic correctness against a generic ANSI-C version of the function.

After a brief discussion of the most important functions, Figure 6.6 shows the resulting speedups compared to the generic version.

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.

Costs Aggregation

For this, we implemented a dedicated 16-bit 5×5 sum filter function. In addition to using various intrinsics, we extended the inner loop in a way 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 costs 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, we combined the searching for the minimum costs values with the subpixel interpolation and the calculation of the confidence values within a single function. We optimized the minimum search 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 costs value.

Subpixel Refinement

The subpixel refinement is done by evaluating (4.10). Conversion into the fixed-point domain achieved a remarkable increase in speed, but the division operation is still processor-intensive. We substituted this integer division by an approximation method applying Newton's method. By using the `_norm()` intrinsic, we gained a first estimate and then three of Newton's iterations are enough to achieve sufficient accuracy.

Confidence Calculation

An evaluation of Equ. (4.15) 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.

6.5 Comparison

Each of the upper introduced reference implementations has its very own characteristics. The plain software version is flexible in Census mask size and all other

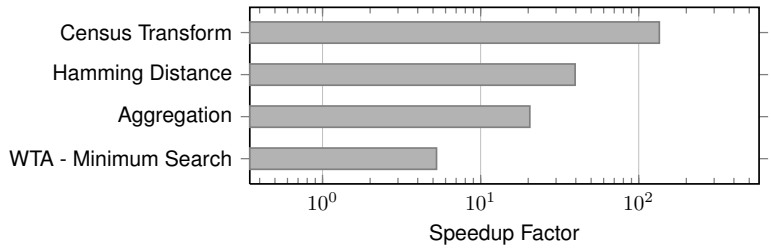


Figure 6.6: DSP low-level function performance optimization speedups

parameters, which is useful for matching quality evaluation. The optimized software can be used on standard PC platforms and is thus generally applicable. It is realized for multi-core CPUs which enables fast processing. The GPU reference implementation showed to be the fastest because of the possibility of massive parallelization. A purely embedded version of our algorithm is the DSP reference implementation. It is optimized for TI DSP platforms which can also be found in smart cameras which enables a compact realization of the stereo sensor.

Due to the fundamental differences of the target platforms, the achieved performance varies for each of them. In the following section, we first explain the possibilities of speed up by multi-core processing for the optimized CPU and the GPU implementations and then compare the processing time as well as the million disparity evaluates per second (Mde/s) between all implementations for several image dimensions and disparity search ranges. Especially the Mde/s perform quite different on the platforms.

6.5.1 Multi-Core Processing

To benefit from multi-core processing, the algorithm has to be parallelizable. This means that the functions, or even a bigger part, of the algorithm can be subdivided into portions which can be executed independently. Independent means that the needed resources and memory can be used, read, and written without interfering with the other in parallel processing portions. Gene Amdahl stated in his famous law [139] that the maximum achievable speedup by parallelizing an algorithm is

$$\text{Speedup} = \frac{1}{(1-P) + \frac{P}{N}}, \tag{6.1}$$

where P is the proportion of the algorithm which can be parallelized and N is the number of processor cores or parallel processing units.

Optimized Software

In the optimized software implementation, multi-cores processing means the use of more than one central processing units (CPUs), as typically provided in modern personal computers. OpenMP offers a fast and simple way to realize this. The idea is to subdivide `for` loops into independent threads which are then processed in parallel. Detailed information about parallel programming on CPUs with OpenMP can be found in [140]. We realized all computationally intensive parts of our algorithm in parallel up to thresholding and 3D reconstruction. The reason is that we want to

enable the possibility of global postprocessing which is not possible in the line-by-line scheme up to this point. Of course, thresholding as well as 3D reconstruction could be implemented in parallel as well.

To verify the achieved speedup of our implementation, we calculated the maximum possible speedup following Amdahl's law in Equ. 6.1. The results in Table 6.4 show that the speedup is near to the optimum which proves that the algorithm is well suited for parallel implementation. Table 6.4 also gives a detailed comparison of the processing times of each step of the algorithm. It may be puzzling that some steps are slower with more cores, but this is caused by cache and operating system. The overall performance stays nearly the same because if one step is slower this is compensated by an other which is faster.

Table 6.4: Performance on an Intel Core2 Quad @2.5GHz with the use of 1 to 4 cores and Windows XP with OpenMP; The image dimensions are 450x375 and the disparity search range is 60. All values, except frame rate and speedup, are in ms.

Function	**1**	**2**	**3**	**4**
Sparse Census Transform	9.73	5.00	3.38	2.68
DSI Calculation	47.81	16.20	9.89	10.85
Texture Map Calculation	1.59	1.58	1.58	1.58
Cost Aggregation	15.16	11.10	8.41	4.82
WTA + Subpixel Refinement	21.19	15.15	9.61	4.98
LR/RL Consistency Check	1.90	1.81	1.61	1.52
Thresholding	0.74	0.73	0.73	0.73
3D Reconstruction	2.85	2.85	2.85	2.86
Total (ms)	**100.97**	**54.42**	**38.06**	**30.02**
Total (fps)	**9.90**	**18.37**	**26.27**	**33.31**
Mde/s	**100.23**	**186.05**	**266.02**	**337.27**
Amdahl speedup	1.0	1.93	2.8	3.61
Multiprocessing speedup	**1.0**	**1.86**	**2.66**	**3.36**

Graphics Processing Unit

As can be seen in Table 6.2, the three graphics cards used significantly differ in nearly all characteristics. Important values are processor clock and number of CUDA cores. Table 6.5 compares the processing speed of the algorithm steps for the GPUs. Unsurprisingly, the lowest performance was achieved with the entry-level Quadro FX 570 GPU. It has a processor clock of 460 MHz and only 32 CUDA cores. The mid-range GPU, Geforce 9800 GT, has a high processor clock of 1500 MHz and is equipped with 112 CUDA cores. The achieved processing speed is about 10 times faster than the first. The third GPU is a high performance card in the consumer market. It has the highest number of CUDA cores, 240, and the highest memory bandwidth. The proposed algorithm achieves a frame rate of 105.4 fps for the Teddy dataset (450×375, 60 disparities) on this graphics card.

Table 6.5: Performance on three GPUs; The image dimensions are 450x375 and the disparity search range is 60. All values, except frame rate, are in ms. The value in brackets gives the number of CUDA cores of the GPU.

Function	**FX 570 (32)**	**9800 GT (112)**	**GTX 280 (240)**
Sparse Census Transform	9.55	0.89	0.52
DSI Calculation	54.93	4.76	2.42
Texture Map Calculation	4.51	0.51	0.24
Cost Aggregation	82.07	7.86	4.03
WTA + Subpixel Refinement	46.65	4.43	2.08
LR/RL Consistency Check	1.79	0.25	0.09
Thresholding	0.51	0.06	0.04
3D Reconstruction	0.40	0.03	0.02
Total (ms)	**200.41**	**18.79**	**9.49**
Total (fps)	**4.98**	**53.21**	**105.4**
Mde/s	**50.42**	**538.85**	**1067.17**

6.5.2 Performance

Table 6.6 gives a direct 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 33.31 fps and the DSP with 7.74 fps.

Table 6.6: Performance of the reference implementations; The image dimensions are 450×375 and the 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**	**GTX 280**	**DSP**
Sparse Census Transform	168	2.68	0.52	8.97
DSI Calculation	332	10.85	2.42	33.08
Texture Map Calculation	29	1.58	0.24	4.02
Cost Aggregation	573	4.82	4.03	33.57
WTA + Subpixel Refinement	555	4.98	2.08	39.11
LR/RL Consistency Check	8	1.52	0.09	2.55
Thresholding	7	0.73	0.04	2.48
3D Reconstruction	32	2.86	0.02	N/A[1]
Total (ms)	**1738**	**30.02**	**9.49**	**129.18**
Total (fps)	**0.575**	**33.31**	**105.4**	**7.74**
Mde/s	**5.82**	**337.27**	**1067.17**	**78.38**

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

Figures 6.7, 6.8 and 6.9 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 we chose commonly used image dimensions 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, 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 extending 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.

6.5.3 Power Consumption

For the power consumption measures, we used a MacMini with an Intel Core2 Duo clocked at 2 GHz for the optimized software implementation. The NVIDIA GTX 280 GPU is used within an Intel Core2 Quad system clocked at 2.4 GHz and equipped with 4GB RAM. For the DSP implementation, we used a Texas Instruments DSP Starter Kit (6416DSK). Table 6.7 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 6.7: 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

We introduced an additional evaluation parameter, million disparity evaluations per Joule (Mde/J), 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.

6.6 Summary

In this chapter, we introduced four reference implementations of the proposed stereo matching algorithm. One of them is a plain, not optimized, software version which enabled the evaluation of all parameter settings. The others are optimized for three fundamentally different platforms. The algorithm is well suited for parallel implementation, so the first reference implementation is optimized for multi-core CPU

machines. A comparison with the maximum achievable speedup, defined by Amdahl's Law, showed that the parallelization works well. It has to be mentioned that optimization with parallelization is only the second step. The comparison with the plain software version shows that a speedup is also possible with optimization of a sequential implementation. The second reference implementation is optimized for graphics processing units which became a powerful alternative processing platform for computational intensive algorithms. We ensured the compatibility of this implementation for at least three different GPU generations and presented a performance comparison. Unsurprisingly, the latest generation performed best. Finally, as a purely embedded solution, we optimized the algorithm for a Texas Instruments digital signal processor platform. This is the most lightweight platform and thus caused the most implementation effort. A known problem of embedded systems is for sure the limited amount of fast internal memory. We overcame this by using resource-optimized slicing which is implemented, among other highly optimized image processing functions for TI DSPs, in our own developed library named PfeLib. The performance comparison of all implementations showed that the algorithm is scalable. It can be used in lightweight embedded systems as well as in systems with high processing power with exactly the same results quality. Due to this scalability, applications with different requirements in terms of image resolution and depth accuracy can be realized in real-time. If the power consumption is considered as evaluation criterion, the DSP implementation can calculate the most disparity evaluations per Joule and is therefore the most efficient platform.

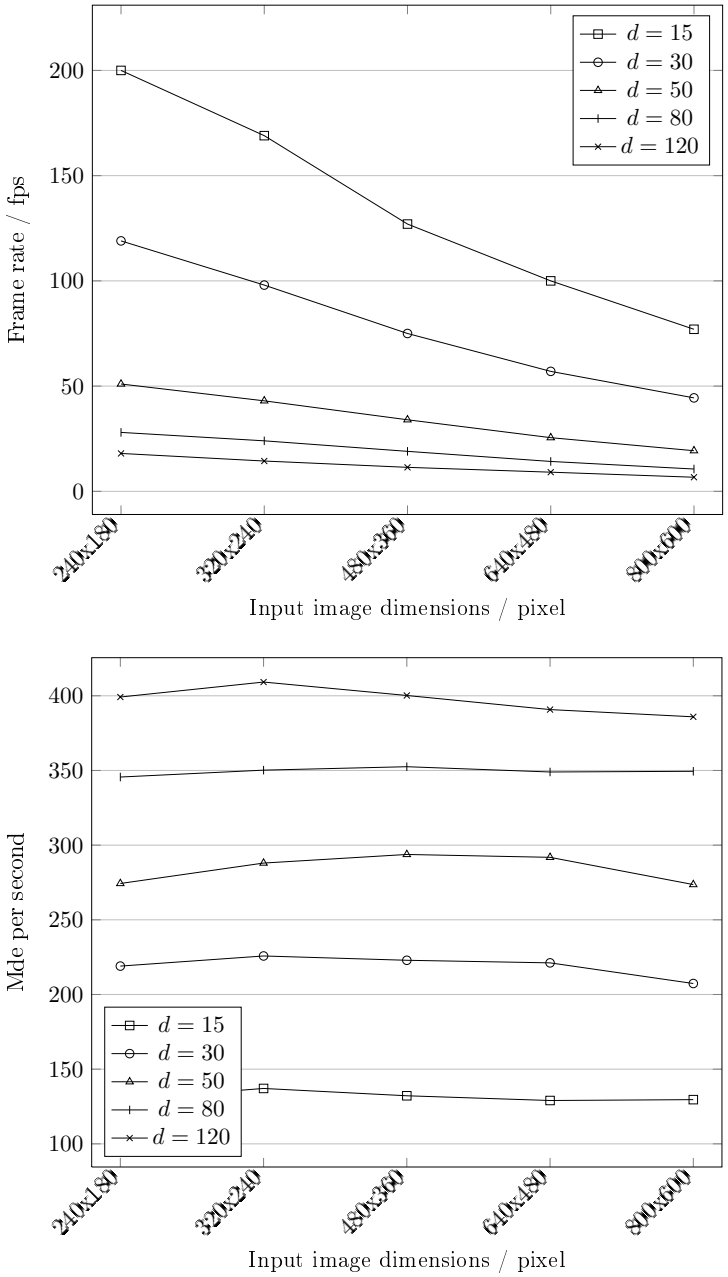


Figure 6.7: Optimized software implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on an Intel 2.5 GHz Core2 Quad CPU

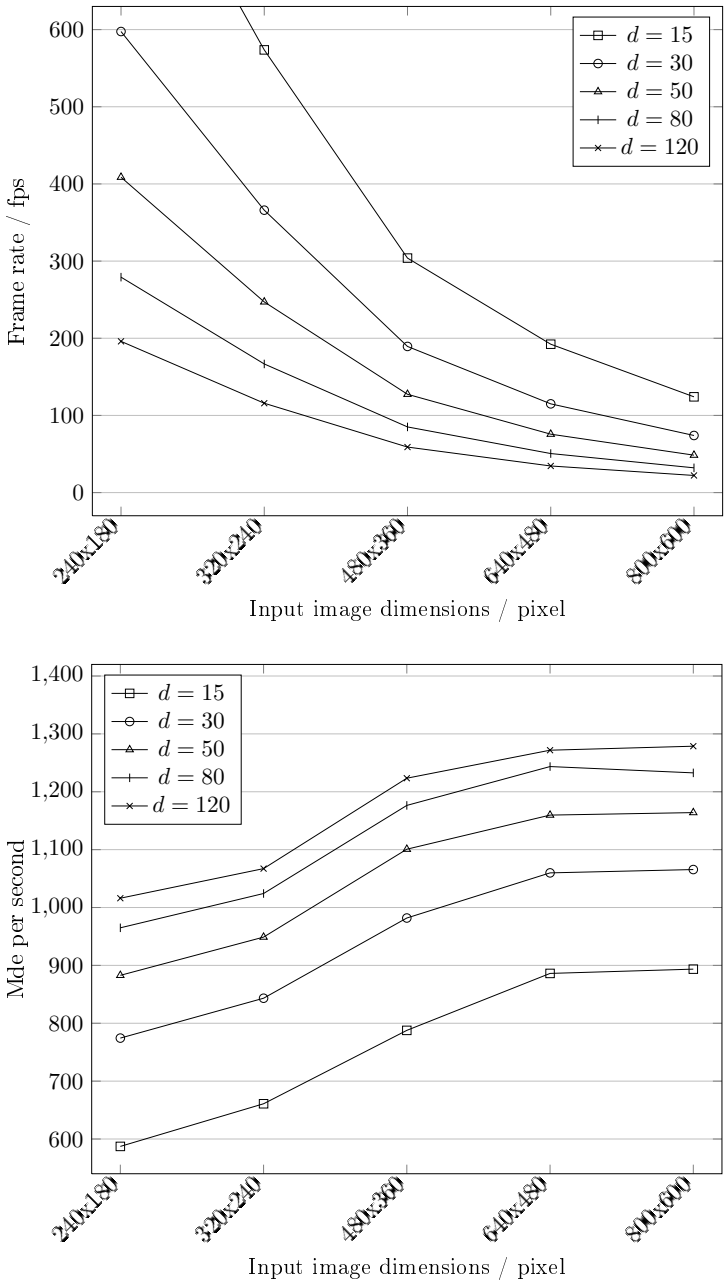


Figure 6.8: GPU implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on an NVIDIA GeForce GTX 280

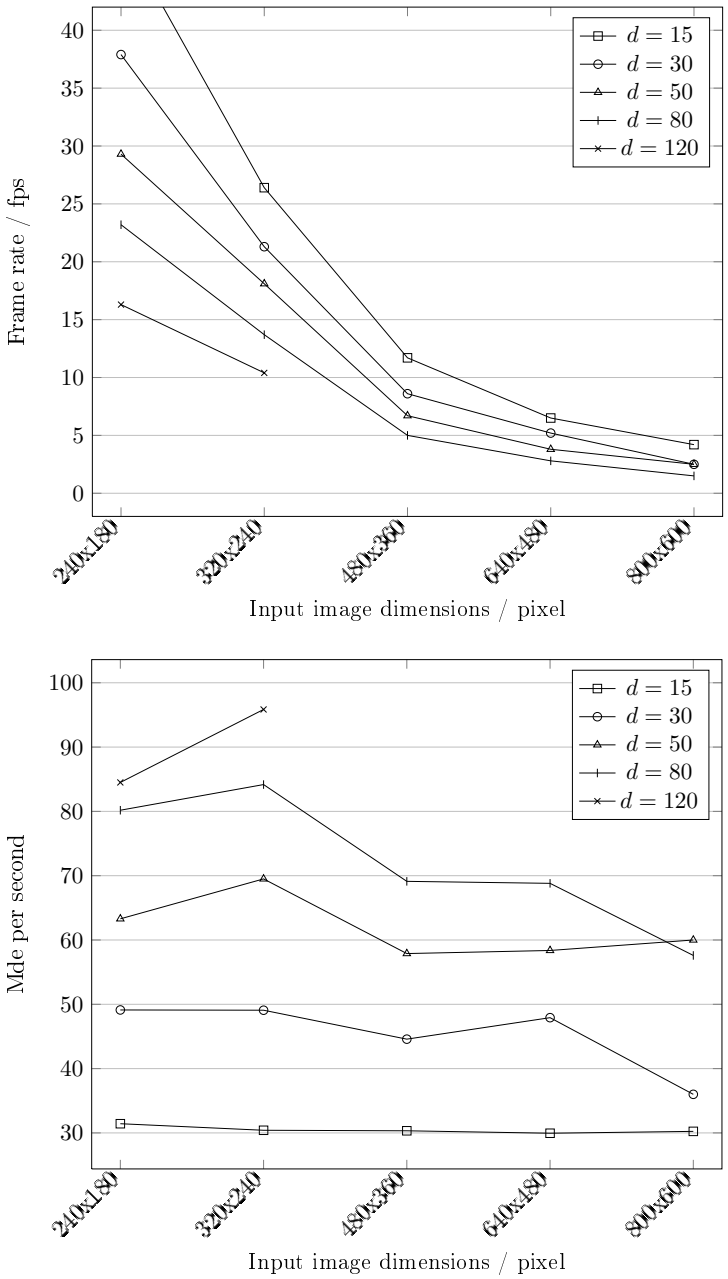


Figure 6.9: DSP implementation: Frame rates (fps) and million disparity evaluations per second (Mde/s) for different image sizes and disparity ranges on a 1GHz TMS320C6416 single core DSP. Some data points are missing due to memory restrictions of the evaluation board (6416DSK)

Chapter 7

Conclusion and Outlook

In this thesis, we presented a 3D perception system for robotic application based on stereo vision. The requirements for such a 3D sensor can be summarized with real-time capability, high speed processing, reliable and dense 3D data, and memory awareness due to embedded realization. A detailed analysis of state-of-the-art stereo matching algorithms showed on the one hand that most sophisticated algorithms deliver high matching quality but are not capable for real-time and embedded realization. On the other hand, existing real-time stereo matching algorithms suffer from low matching quality or scalability because of the use of dedicated hardware. The core part of the introduced system is the adapted and highly optimized stereo matching algorithm which uses the Census transform and block correlation to solve the correspondence problem.

Due to the gain in processing time and the insignificant loss of quality, a sparse Census transform is introduced. This strategy benefits from the proven fact that large sparse Census masks perform better than small dense masks with the same processing effort. This has been shown by evaluating the algorithm with 31 Middlebury datasets under normal conditions and with additive noise. As a consequence, the algorithm is robust, easy to parameterize and it delivers a good matching quality even under real-world conditions.

The algorithm has been implemented on a PC, a GPU as well as on a purely embedded DSP platform. All implementations, aside from the plain software, achieve real-time performance, whereby the GPU is by far the fastest but has also the highest power consumption. 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 on existing platforms. The comparison with the well-known local area-based stereo matching algorithm SAD showed that our approach clearly achieves better results. The comparison with a global optimizing algorithm, SGM, showed that our approach delivers nearly the same results. We achieve this within a fraction of processing time and, more important due to embedded realization, low memory consumption. We evaluated the resulting disparity maps on the Middlebury stereo website where the algorithm performs well in comparison to other real-time approaches. Especially in terms of processing times, the proposed algorithm outper-

forms algorithms with comparable matching quality.

Our research in real-time capable stereo matching algorithms for embedded realization showed that even if we achieved good matching quality on textured and low textured areas, there is room for further improvements. The projection of random light pattern onto the target scene to generate synthetic texture could be a possibility, but is out of scope in this work because passive technologies are preferred. A processing time improvement for embedded systems can be achieved by the use of upcoming multi-core DSPs. For generation of large 3D maps and models, an automated registration of the calculated 3D point clouds could be of interest.

Appendix A

Extension to Global Optimization

In this chapter, we give possible algorithmic improvements of our stereo algorithm by the use of semi-global Matching and plane fitting. The challenge is to keep the computational effort and the memory consumption low to enable embedded and real-time processing. First, we explain the workflow and the single steps of the improvements in Section A.1 followed by a detailed analysis of influence of the modifications in terms of matching quality, processing time, and memory consumption in Section A.2.

A.1 Workflow

Figure A.1 shows the workflow of the improved algorithm. The first two steps are taken from our sparse Census algorithm we described in this thesis. Then we apply a modified semi-global Matching (SGM) to increase the confidence of the matches, and thus to determine the initial disparity map. Afterwards, we do a segmentation on either the left stereo image or the texture map. Finally, we fit a planar model onto the segments which is then used to determine the refined disparity map.

A.1.1 Modified Semi-Global Matching

Semi-global matching determines the optimal paths through the whole image for each pixel, thus the costs of the path have to be stored for the whole image. In Section 6.2, we showed that an optimized high-speed implementation of a Census-based stereo matching approach benefits from a line-by-line processing of the images. Only a number of lines equal to the aggregation block size has to be stored at once. Especially for embedded systems, this approach is advantageous because the data can then be processed in the fast on-chip memory. To keep the benefit of line-by-line processing, we introduce a modified SGM technique in this work. It uses the assumption that a part of the image is enough for each pixel to benefit from the SGM. Therefore the initial costs matrix is divided into horizontal stripes with a range of n_r pixels (the last stripe may be smaller). The stripes are treated like the whole image and the paths are calculated with Equ. (3.13) as well. For determining the optimal paths through the stripes a number equal to the range of the initial costs has to be stored. Thus, the memory consumption depends on the size of the range and the

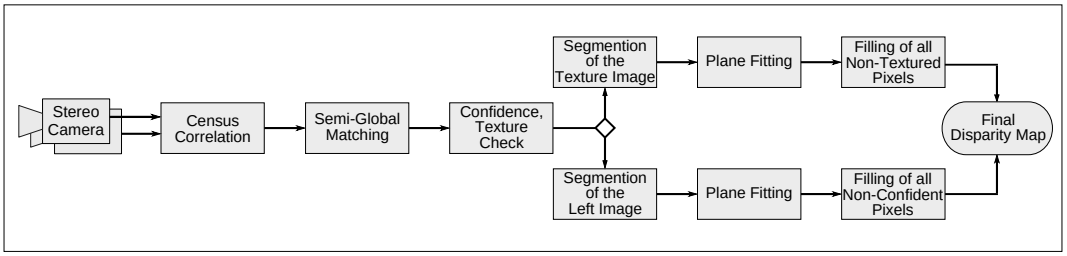


Figure A.1: The workflow of the proposed improved algorithm

number of disparities. The stripes are then processed separately and the resulting disparity map (DM) is stored as a combination of the total number of stripes.

A.1.2 Confidence and Texture

Even if SGM increases the reliability of the matches, a number of false positives remain. To determine them, we use the confidence value from Section 4.7.2. As mentioned above, large textureless areas are difficult to match even if SGM is done over the whole image. To identify them, we also use the texture image as calculated in Section 4.7.2.

A.1.3 Segmentation and Plane Fitting

Once the initial disparity map is calculated, textureless areas and non-confident pixels are optimized with segmentation and plane fitting. The segmentation can either be done by color on the left input image (mean-shift [141]) or binary on the texture image. The texture image (TI) is derived from the texture map with

$$\mathrm{TI}(u,v) := \begin{cases} 0 & \text{if } \mathrm{TM}(u,v) \leq t_{texture} \\ 255 & \text{otherwise} \end{cases} , \tag{A.1}$$

where $t_{texture}$ is the used threshold. The segmentation process on the binary texture image is straight forward. All white pixels are united to one segment and all connected black pixels are joint to single segments.

An advantage of the texture segmentation is that monochrome input images can be used as well as color images. On the one hand, monochrome cameras deliver images of higher native resolution than common industrial color cameras (because of the Bayer pattern) and on the other hand, for this kind of segmentation, the focus exactly lies on textureless areas which are the main regions of interest for optimization. The advantage of color segmentation is that the segments are more accurate and that occlusions can better be optimized. Section A.2 will show that color segmentation is more suitable for the Middlebury datasets and texture segmentation for real-world scenes.

Both segmentations have in common that only pixels which successfully passed the confidence check are used for the plane fitting step. A plane is represented by three parameters a, b, and c of equation

$$d(u,v) := au + bv + c \ . \tag{A.2}$$

These parameters can be estimated with the method of least squares by solving the linear equation system

$$\begin{bmatrix} \sum_{i=1}^{m} u_i^2 & \sum_{i=1}^{m} u_i v_i & \sum_{i=1}^{m} u_i \\ \sum_{i=1}^{m} u_i v_i & \sum_{i=1}^{m} v_i^2 & \sum_{i=1}^{m} v_i \\ \sum_{i=1}^{m} u_i & \sum_{i=1}^{m} v_i & \sum_{i=1}^{m} 1 \end{bmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} u_i d_i \\ \sum_{i=1}^{m} v_i d_i \\ \sum_{i=1}^{m} d_i \end{pmatrix}, \tag{A.3}$$

where m is the number of confident pixels in the segment. Unfortunately this is not robust against outliers, thus the method described by Bleyer and Gelautz [142] is used. The problem is solved by iteratively eliminating outliers until the calculated plane has reached its final state.

A problem of segmentation regarding real-time capability is that the processing time strongly depends on the number of segments found. This work tries to deal with this problem by limiting the number of possible segments. The authors know that this is just a first step towards real-time segmentation because also the absolute number of confident pixels inside the segments influences the processing time.

After plane fitting, the last step is to optimize and refine the initial disparity map with the calculated planar model.

A.1.4 Disparity Map Refinement

In contrast to traditional model-based segmentation optimization, in this work only non-confident pixels (which failed the confidence check) or pixels in textureless areas (which failed the texture check) are refined with the calculated planes. The others are taken from the initial disparity map. Additionally, only reliable segments are used for refinement because in difficult areas the initial data may be not good enough for a correct model estimation. The reliability of the planes is differently determined for color and texture segmentation.

For color segments the function

$$\Omega_c(C) := \begin{cases} \text{true} & \text{if } \frac{n_c}{n_p} \leq t_{confidence} \\ \text{false} & \text{otherwise} \end{cases} \tag{A.4}$$

is used where C is the segment, n_c the number of non-confident pixels and n_p the number of pixels in C. If the segment is reliable, thus the fraction of confident pixels in the segment is higher than the given threshold $t_{confidence}$, Ω_c is true and false otherwise.

In large textureless areas, often a low number of confident pixels exists. The use of Ω_c would not be advantageous because the percentage of confident pixels in textureless areas varies with the segment size. To overcome this, another reliability metric,

$$\Omega_t(C) := \begin{cases} \text{true} & \text{if } \delta \leq t_{plane} \\ \text{false} & \text{otherwise} \end{cases}, \tag{A.5}$$

is introduced to measure the quality of the estimated plane where t_{plane} is the used threshold. The criterion is the average distance

$$\delta := \frac{1}{m} \sum_{i=0}^{m} |d_i - (au_i + bv_i + c)|, \tag{A.6}$$

between the points and the estimated plane, where m is the number of confident pixels in the segment.

Summarizing, the last step of the proposed algorithm is the refinement of the initial disparity map. For color segmentation, only non-confident pixels and for texture segmentation only pixels in textureless areas are refined. The reliability of the estimated planes is determined and only reliable planes are used for this final optimization.

A.2 Evaluation

In this section, we present the results of the proposed improvements. First, the matching quality, the processing time, and the memory consumption of the modified semi-global matching is evaluated. Then, on the one hand, for evaluation of the matching quality, we again used the Middlebury ranking. The main advantage is the possibility of comparing the stereo vision algorithm with many others online. The datasets used for this evaluation are not realistic representatives for the target application, thus results for real-world scenes are shown on the other hand.

A.2.1 Modified Semi-Global Matching

Semi-global Matching optimizes the disparities in either 8 or 16 directions with the use of two penalties $P_1 = 54$ and $P_2 = 99$. The use of 16 directions showed no considerable enhancement of the results so 8 directions are used because of the shorter processing time. The optimal penalties were determined by evaluation of all meaningful combinations.

Figure A.2 shows an evaluation of matching quality and memory consumption for the modified SGM approach. As can be seen in Fig. A.2(a), the average percentage of matched pixels over the four main ranking Middlebury datasets with ranges $n_r = 5(5)190$ is very similar to the original approach (straight black line). The enhancement of the modified SGM is the reduced memory consumption. Original SGM has a memory consumption of about 40 MB for the Teddy dataset. As can be seen in Fig. A.2(b), if a range of about 55 is used, the memory consumption is about 5 MB. If a very small range of 5 is used, the memory consumption even is about 300 KB, which makes it very suitable for embedded realization. For better visualization, Fig .A.2(c) shows the memory consumption and the percentage of correct matches for the Tsukuba dataset plotted in one chart.

As a reminder, the confidence of the matches is essential for successful plane fitting. Figure A.3 shows the improvement of the confidence when modified SGM is used. Both costs functions show the same correctly matched pixel (disparity is at the lowest costs) for Census on the left side and for Census with modified SGM on the right side. The difference between the two best matching candidates is very low for Census, thus the confidence is very low. SGM highly increases the difference and thus the confidence as well. The more correct matches are marked as confident, the more can be used for the plane fitting step what again increases the quality of the final disparity map.

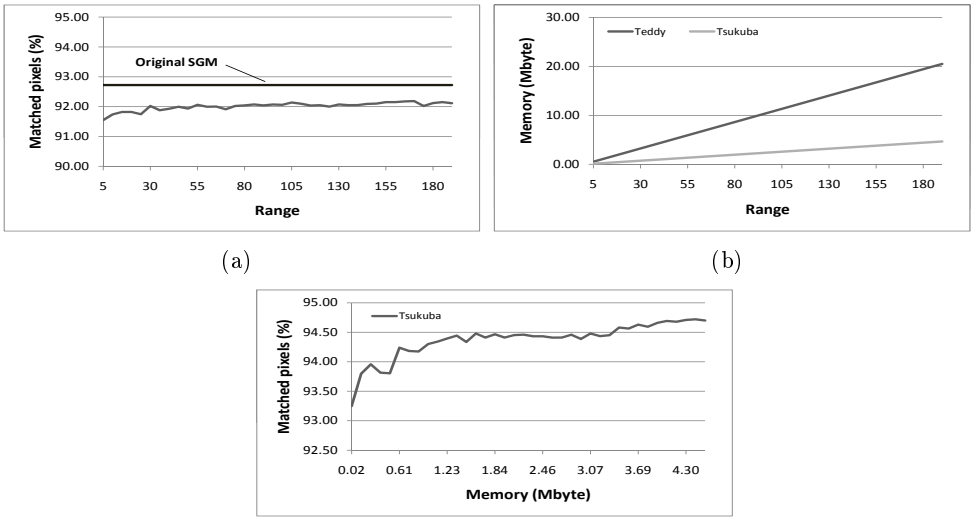


Figure A.2: Evaluation of different ranges n_r for modified SGM: (a) Percentage of correct matched pixels (average over the Middlebury datasets), (b) memory consumption, and (c) a combined chart of memory consumption and correct matches

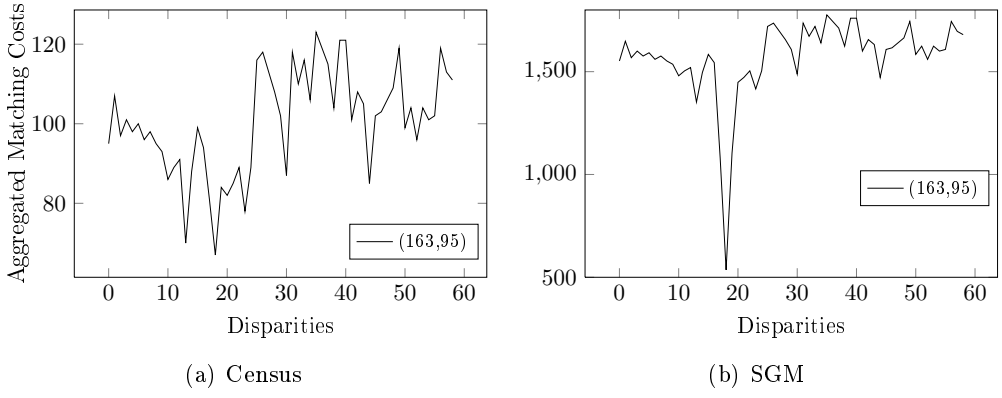


Figure A.3: Illustration of the confidence improvement by using modified SGM with $n_r = 55$. The confidence value using pure Census is $\mathrm{CM}(u, v) = 13,65$ and with SGM the maximum of $\mathrm{CM}(u, v) = 255$.

Figure A.4: The results of the proposed algorithm (Census correlation with SGM and color-based segmentation) for the Middlebury datasets.

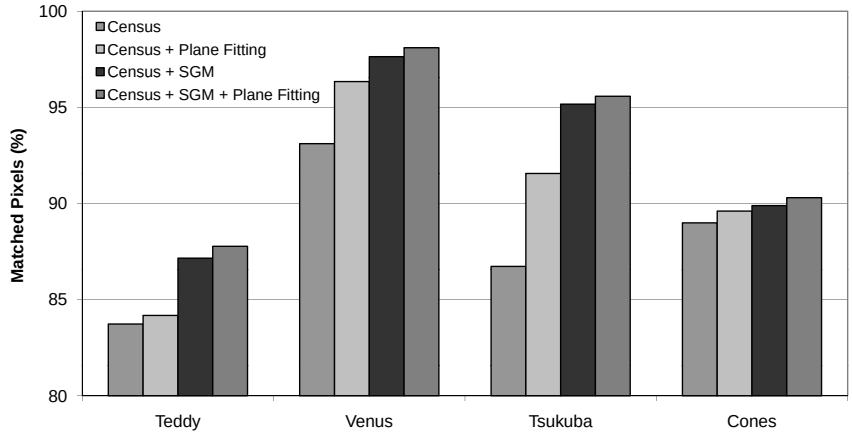


Figure A.5: The percentage of correctly matched pixels with Census correlation, plane fitting (color segmentation) and SGM.

A.2.2 Middlebury Ranking

We also evaluated our improvements with the Middlebury database. For this evaluation, the color-based segmentation approach is used because it has the big advantage that many occluded areas (if Ω_c is true) are filled with the calculated planes rather with extrapolation. Outliers are reduced with a final median filter.

The resulting disparity maps are compared with the ground truth, which is the reference disparity map of the scene. Figure A.4 shows the resulting disparity maps of the proposed algorithm. Figure A.5 shows the resulting improvements of the different algorithm steps.

Table A.1 compares different algorithm configurations in the Middlebury evaluation framework. The best result in the main ranking (rank 37) could be achieved with a combination of Census correlation, SGM and plane fitting. Additionally to the proposed algorithm steps, the results of standard SAD for local costs calculation are shown.

As can be seen, SGM clearly improves the quality of the matches. When using the proposed modified SGM technique the rank shrinks a few places. This is caused by the fact that the entries in the Middlebury ranking are very close together, so little worse results may cause significant degradation in the ranking. A meaningful metric is the average bad matches percentage. It shows that the overall performance of original SGM and the modified version is quite similar. Also interesting is, when using the average bad matches as criterion, that SGM produces nearly the same

matching quality for Census correlation and SAD.

An important factor in Table A.1 is the confidence threshold. As mentioned in the previous section, SGM significantly increases the confidence of matched pixels. In comparison to SAD and Census, for SGM a much higher confidence threshold can be used without eliminating too many true positives. This increases the number of confident matches which is essential for the plane fitting step.

In the main ranking of the Middlebury website, all matches within an error threshold of 1 are valid. If the error threshold is set to 0.5, which means that subpixel accuracy is supposed, the best position of the proposed algorithm increases to rank 10. If additionally only non-occluded areas are evaluated, the rank increases to position 2.

Table A.1: The proposed algorithm in different configurations evaluated with the Middlebury framework. PF stands for plane fitting.

	Error th. = 1.0		**Error th. = 0.5**		
Algorithm	Rank	Av. bad matches	Rank	Av. bad matches	**Conf.**
Census	56	9.86	16	9.86	30
SAD	66	13.20	57	22.20	5
Census + SGM	40	8.35	9	12.10	95
SAD + SGM	47	7.41	19	10.50	10
Census + SGM + PF	37	8.19	10	12.20	95
SAD + SGM + PF	46	8.35	19	15.20	10
Census + SGM ($n_r = 10$)	52	9.19	14	13.70	95
Census + SGM ($n_r = 55$)	55	9.70	14	13.90	95
Census + SGM ($n_r = 180$)	51	9.05	11	12.90	95
Census + SGM ($n_r = 10$) + PF	48	8.84	12	13.70	95
Census + SGM ($n_r = 55$) + PF	52	9.32	14	14.00	95
Census + SGM ($n_r = 180$) + PF	46	8.90	11	13.10	95

A.2.3 Real-World Scenes

To show the power of plane fitting with texture segmentation, two real-world scenes for robot applications are evaluated.

Figure A.6 shows a floor scene which is difficult for area-based stereo matching approaches. In general, randomly patterned surfaces, such as the carpet in this scene, can be matched well. The most difficult areas for stereo matching here are the monotone white walls (marked black in the texture image in Fig. A.6(c)). The pure Census correlation in Fig. A.6(e) can deal with the carpet well but has its problems with the walls. The same for the combination of Census and SGM in Fig. A.6(f) with the enhancement that the carpet is completely dense. The walls are in both disparity maps reduced to noise. To deal with this, the confidence check was introduced to eliminate obviously wrong matches. The result of Census correlation with confidence check in Fig. A.6(g) shows that the disparity map is very sparse and almost all matches of the walls are eliminated. The proposed improvements were developed exactly to optimize such scenes. The resulting disparity map in

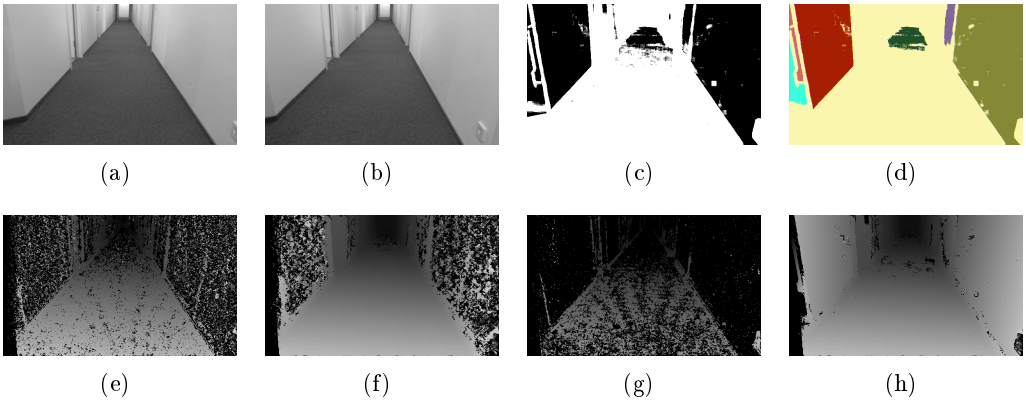
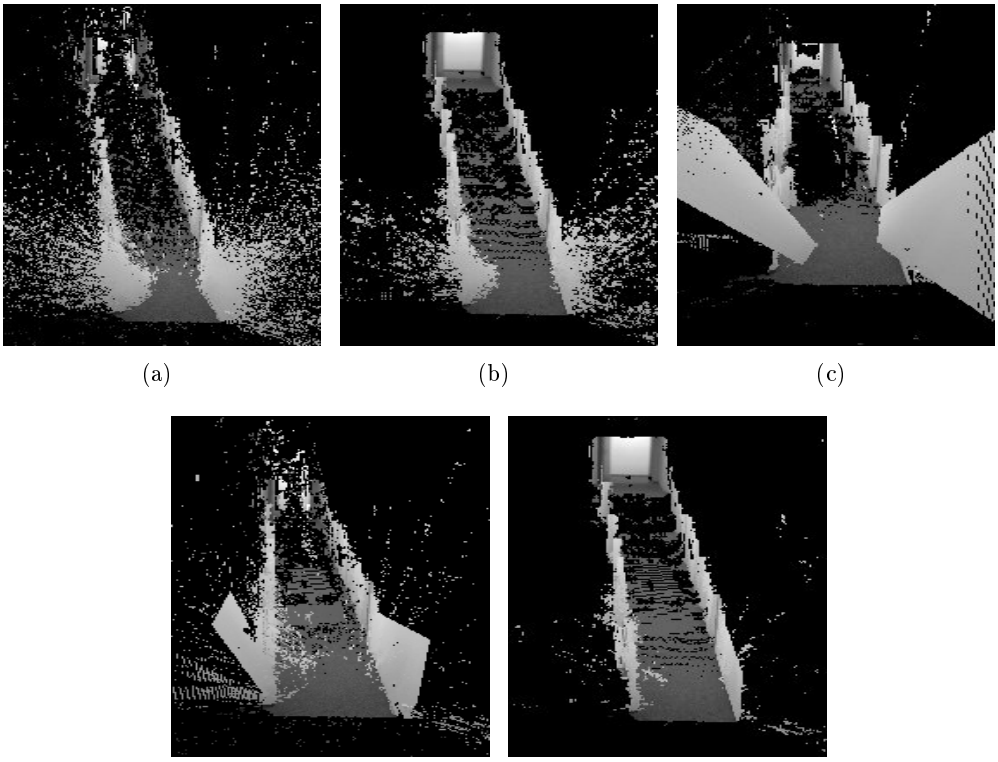


Figure A.6: The results of the floor scene with large textureless areas: (a) original left image, (b) original right image, (c) texture image ($\tau_2 = 20$), (d) texture-based segmentation, (e) disparity map for pure Census correlation, (f) disparity map for Census correlation and SGM, (g) disparity map for Census correlation with confidence check, (h) disparity map for Census correlation, SGM, and plane fitting

Fig. A.6(h) shows that areas of the image with enough texture (white in the texture image) are kept original and areas with low texture (black in the texture image) are used for the segmentation-based optimization. The quality of the planes strongly depends on the data used for fitting, so a high confidence threshold of $\tau_1 = 200$ is used. To show the quality of the 3D data, the 3D point clouds for three algorithm configurations are given in Fig. A.7. As can be seen, the walls are completely wrong when no optimization is used. Only the planes in Fig. A.7(e) are good estimates of the walls in the scene. Especially Fig. A.7(c) shows the impact of the higher confidence of Census in comparison to SAD. Not all textureless areas can be optimized using texture-based segmentation. Figure A.8 shows an example where an estimated plane does not fit correctly. Here, the estimated plane in Fig. A.8(c) of the wall behind the door is obviously wrong. Therefore, the threshold function Ω_t was introduced to eliminate such planes as shown in Fig. A.8(d). A limitation of the approach is that the fitted planes are only estimations of the real world. Problematic are curved surfaces because a plane cannot be fitted on there. However, most curved surfaces are not textureless in the images because of different light reflections on the surfaces. Additionally, the probability that such a surface would be eliminated by Ω_t is high because the distance of the points from the curved surface to the estimated plane is large. Nevertheless, in indoor home robot applications, the assumption that textureless areas are planar in many cases can be made.

A.3 Summary

In this chapter we introduced extensions to global optimization for our stereo matching approach consisting of a combination of SGM disparity optimization and segmentation-based plane fitting for enhancements on textureless and occluded areas. A modification of original SGM makes the approach capable for embedded realization as well. The image is divided into stripes that may fit into fast on-chip



(a) (b) (c)

(d) (e)

Figure A.7: 3D point cloud of the floor scene with (a) pure Census correlation (without confidence and texture check), (b) Census with SGM, (c) SAD with SGM and plane fitting, (d) Census with plane fitting, (e) Census with SGM and plane fitting.

(a) (b) (c) (d)

Figure A.8: Results of the desk scene: (a) Left stereo image, (b) the segmentation of the texture image, (c) 3D point cloud with SGM and texture-based optimization without plane check, and (d) with plane check and a threshold of $t_{plane} = 0.2$.

memory of digital signal processors. Semi-global matching significantly increases the confidence of the matches. It is shown that the segmentation-based plane fitting performs well with the Census-based correlation method. The main advantage is the improvement of the matching quality in occluded and textureless areas. Furthermore, it is shown that the texture-based segmentation approach makes it possible to match large textureless areas very well whereas these are a significant problem for standard area-based stereo matching approaches.

Bibliography

[1] I. Asimov. Runaround. *Astounding Science Fiction*, 1942.

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

[3] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2 edition, 1999.

[4] O. Faugeras. *Three-Dimensional Computer Vision*. The MIT Press, Cambridge, Massachusetts, 4 edition, 2001.

[5] R. C. Gonzalez and R. E. Woods. *Digital Image Processing, Second Edition*. Pearson Education International, 2002.

[6] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[7] E. R. Davies. *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann, San Francisco, CA, 3rd edition, December 2005.

[8] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Proc. Seventh IEEE International Conference on Computer Vision The*, volume 1, pages 666–673, 20–27 Sept. 1999.

[9] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1), 2000.

[10] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

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

[12] *OpenCV*. URL `http://opencv.willowgarage.com/`.

[13] R. I. Hartley. Theory and practice of projective rectification. *International Journal of Computer Vision*, 35:115–127, 1999.

[14] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3:323–344, 1987.

[15] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[16] M. Z. Brown, D. Burschka, and G. D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25: 993–1008, 2003.

[17] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002. ISSN 0920-5691.

[18] H. Hirschmueller. Improvements in real-time correlation-based stereo vision. In *Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision*, 2001.

[19] H. Hirschmueller, P. R. Innocent, and J. Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, pages 229–246, 2002.

[20] A. Fusiello, V. Roberto, and E. Trucco. Efficient stereo with multiple windowing. In *Proceedings of the International Conference of Computer Vision and Pattern Recognition*, pages 858 – 863, 1997.

[21] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:920 – 932, 1994.

[22] S. Yoon, D. Min, and K. Sohn. Fast dense stereo matching using adaptive window in hierarchical framework. In *Proceedings of the International Symposium on Visual Computing*, pages 316 – 325, 2006.

[23] K. J. Yoon and I. S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transanctions on Pattern Analysis and Machine Intelligence*, 28:650 – 656, 2006.

[24] O. Veksler. Fast variable window for stereo correspondence using integral images. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 551–561, 2003.

[25] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2008.

[26] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proceedings of 3rd European Conf. Computer Vision*, pages 151–158, Stockholm, 1994. URL `citeseer.ist.psu.edu/article/zabih94nonparametric.html`.

[27] R. Zabih. *Individuating Unknown Objects by Combining Motion and Stereo.* PhD thesis, Department of Computer Science, Stanford University, 1994.

[28] H. Hirschmueller and D. Scharstein. Evaluation of stereo matching costs on images with radiometric differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1582–1599, Sept. 2009.

[29] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (2):139–154, March 1985.

[30] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35:3:269–293, 1996.

[31] R. C. Gonzalez, J. A. Cancelas, J. C. Alvarez, J. A. Fernandez, and J. M. Enguita. Fast stereo vision algorithm for robotic applications. In *Proc. 7th IEEE International Conference on Emerging Technologies and Factory Automation ETFA '99*, volume 1, pages 97–104, 18–21 Oct. 1999.

[32] S. Forstmann, Y. Kanou, J. Ohya, S. Thuering, and A. Schmitt. Real-time stereo by using dynamic programming. In *In Conference on Computer Vision and Pattern Recognition Workshop*, page 29. IEEE Computer Society, 2004.

[33] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, Nov. 2001.

[34] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions using graph cuts. In *Proceedings of the International Conference on Computer Vision*, pages 508–515, 2001.

[35] J. Sun, N.-N. Zheng, and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7): 787–800, July 2003.

[36] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister. Real-time global stereo matching using hierarchical belief propagation. In *Proceedings of The British Machine Vision Conference*, pages 989–998, 2006.

[37] H. Hirschmueller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[38] H. Hirschmueller. Stereo vision in structured environments by consistent semi-global matching. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2386 – 2393, 2006.

[39] S. K. Gehrig, F. Eberl, and T. Meyer. A real-time low-power stereo vision engine using semi-global matching. *Computer Vision Systems*, 5815/2009:134–143, 2009.

[40] I. Ernst and H. Hirschmueller. Mutual information based semi-global stereo matching on the gpu. In *Proceedings of the 4th International Symposium on Advances in Visual Computing*, pages 228–239, 2008.

[41] D. Comaniciu and M. Peter. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603 – 619, 2002.

[42] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *Proceedings of the European Conference on Computer Vision*, pages 82–96, 2002.

[43] A. Klaus, M. Sormann, and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proceedings of the 18th International Conference on Pattern Recognition*, 2006.

[44] H. Hirschmueller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 Issue 2:328 – 341, 2008.

[45] M. Bleyer and M. Gelautz. A layered stereo algorithm using image segmentation and global visibility constraints. In *Proceedings of the IEEE International Conference on Image Processing*, pages 2997–3000, 2004.

[46] *Stereo Evaluation*. Middlebury Computer Vision. URL `http://vision.middlebury.edu/stereo/`.

[47] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society Press, 2003.

[48] Q. Yang, R. Yang, J. Davis, and D. Nister. Spatial-depth super resolution for range images. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR '07*, pages 1–8, 17–22 June 2007.

[49] Z.-F. Wang and Z.-G. Zheng. A region based stereo matching algorithm using cooperative optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[50] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister. Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31 Issue 3:492 – 504, 2009.

[51] L. Xu and J. Jia. Stereo matching: An outlier confidence approach. *Computer Vision – ECCV 2008*, 5305:775–787, 2008.

[52] M. Bleyer, C. Rother, and P. Kohli. Surface stereo with soft segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[53] M. Bleyer, M. Gelautz, C. Rother, and C. Rhemann. A stereo approach that handles the matting problem via image warping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[54] Y . Taguchi, B. Wilburn, and C. L. Zitnick. Stereo reconstruction with mixed pixels using adaptive over-segmentation. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2008.

[55] A. Hosni, M. Bleyer, M. Gelautz, and C. Rhemann. Local stereo matching using geodesic support weights. In *Proceedings of the IEEE International Conference on Image Processing*, 2009.

[56] Q. Yang, C. Engels, and A. Akbarzadeh. Near real-time stereo for weakly-textured scenes. In *Proceedings of the British Machine Vision Conference*, 2008.

[57] J. Sun, Y. Li, S. Bing Kang, and H.-Y. Shum. Symmetric stereo matching for occlusion handling. In *Proceedings of tht IEEE Conference on Computer Vision and Pattern Recognition*, pages 399 – 406, 2005.

[58] Z. Gua, X. Su, and Y. Liu. Local stereo matching with adaptive support-weight, rank transform and disparity calibration. *Pattern Recognition Letters*, 29 Issue 9:1230 – 1235, 2008.

[59] S. Mattoccia, F. Tombari, and L. Di Stefano. Stereo vision enabling precise border localization within a scanline optimization framework. *LNCS Computer Vision ACCV*, 4844:517 – 527, 2007.

[60] K.-J. Yoon and I. S. Kweon. Stereo matching with the distinctive similarity measure. In *Proceedings of the International Conference on Computer Vision*, 2007.

[61] C. L. Zitnick and S. B. Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision*, 75:49 – 65, 2007.

[62] T. Montserrat, J. Civit, O. D. Escoda, and J.-L. Landabaso. Depth estimation based on multiview matching with depth/color segmentation and memory efficient belief propagation. In *Proceedings of the IEEE International Conference on Image Processing*, 2009.

[63] D. Mukherjee, G. Wang, and J. Wu. Stereo matching algorithm based on curvelet decomposition and modified support weights. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 2010.

[64] F. Tombari, S. Mattoccia, and L. Di Stefano. Segmentation-based adaptive support for accurate stereo correspondence. *LNCS Advances in Image and Video Technology*, 4872:427 – 438, 2007.

[65] S. Mattoccia. A locally global approach to stereo correspondence. In *Proceedings of the International Conference on Computer Vision Workshops*, 2009.

[66] D. Min and K. Sohn. Cost aggregation and occlusion handling with wls in stereo matching. *IEEE Transactions on Image Processing*, 17 Issue 8:1431 – 1442, 2008.

[67] C. Lei, J. Selzer, and Y.-H. Yang. Region-tree based stereo using dynamic programming optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2378 – 2385, 2006.

[68] E. S. Larsen, P. Mordohai, M. Pollefeys, and H. Fuchs. Temporally consistent reconstruction from multiple video streams using enhanced belief propagation. In *Proceedings of the International Conference on Computer Vision*, pages 1 – 8, 2007.

[69] O. Stankiewicz and K. Wegner. Depth map estimation software version 3. In *ISO/IEC MPEG meeting M15540*, 2008.

[70] Y. Deng and X. Lin. A fast line segment based dense stereo algorithm using tree dynamic programming. In *Proceedings of the European Conference on Computer Vision, LNCS*, pages 201 – 212, 2006.

[71] A. Bhusnurmath and C. J. Taylor. Solving stereo matching problems using interior point methods. In *Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, pages 321 – 329, 2008.

[72] S. K. Gehrig and U. Franke. Improving stereo sub-pixel accuracy for long range stereo. In *Proceedings of the International Conference on Computer Vision, Workshop*, 2007.

[73] A. Banno and K. Ikeuchi. Disparity map refinement and 3d surface smoothing via directed anisotropic diffusion. In *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops*, pages 1870 – 1877, 2009.

[74] S. Mattoccia, S. Giardino, and A. Gambini. Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering. In *Proceedings of the 9th Asian Conference on Computer Vision*, 2009.

[75] R. Brockers. Cooperative stereo matching with color-based adaptive local support. *LNCS Computer Analysis of Images and Patterns*, 5702:1245, 2009.

[76] T. Yu, R.-S. Lin, B. Super, and B. Tang. Efficient message representations for belief propagation. In *Proceedings of the IEEE International Conference on Computer Vision*, 2007.

[77] K. Zhang, J. Lu, G. Lafruit, R. Lauwereins, and L. V. Gool. Real-time accurate stereo with bitwise fast voting on cuda. In *Proceedings of the IEEE International Conference on Computer Vision, 5th Workshop on Embedded Computer Vision*, 2009.

[78] K. Zhang, J. Lu, and L. Gauthier. Cross-based local stereo matching using orthogonal integral images. *IEEE transactions on circuits and systems for video technology*, 19:1073–1079, 2009.

[79] O. Woodford, P. Torr, I. Reid, and A. Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31 Issue 12:2115 – 2128, 2009.

[80] T. Liu, P. Zhang, and L. Luo. Dense stereo correspondence with contrast context histogram, segmentation-based two-pass aggregation and occlusion handling. *LNCS Advances in Image and Video Technology*, 5414:449 – 461, 2008.

[81] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Near real-time stereo based on effective cost aggregation. In *Proc. 19th International Conference on Pattern Recognition ICPR 2008*, pages 1–4, 8–11 Dec. 2008.

[82] H. Trinh and D. McAllester. Unsupervised learning of stereo vision with monocular cues. In *Proceedings of the British Machine Vision Conference*, 2009.

[83] L. C. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23:600–608, 2004.

[84] W. Yu, T. Chen, F. Franchetti, and J. C. Hoe. High performance stereo vision designed for massively data parallel platforms. *To appear in IEEE Transactions on Circuits and Systems for Video Technology*, 2010.

[85] M. Vanetti, I. Gallo, and E. Binaghi. Dense two-frame stereo correspondence by self-organizing neural network. *LNCS Image Analysis and Processing – ICIAP 2009*, 5716:1035–1042, 2009.

[86] S. Kosov, T. Thormaehlen, and H. P. Seidel. Accurate real-time disparity estimation with variational methods. In *Proceedings of the International Symposium on Visual Computing*, 2009.

[87] J. Lu, G. Lafruit, and F. Catthoor. Anisotropic local high-confidence voting for accurate stereo correspondence. In *Proceedings of SPIE*, 2008.

[88] J. Salmen, M. Schlipsing, J. Edelbrunner, S. Hegemann, and S. Lueke. Real-time stereo vision: Making more out of dynamic programming. *LNCS: Computer Analysis of Images and Patterns*, 5702/299:1096–1103, 2009.

[89] T. Pock, T. Schoenemann, G. Graber, H. Bischof, and D. Cremers. A convex formulation of continuous multi-label problems. *ECCV '08 Proceedings of the 10th European Conference on Computer Vision: Part III*, 5304:792 – 805, 2008.

[90] C. Strecha, R. Fransens, and L. Van Gool. Combined depth and outlier estimation in multi-view stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2394 – 2401, 2006.

[91] P. Mordohai and G. Medioni. Stereo using monocular cues within the tensor voting framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:968–982, 2006.

[92] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 798–805, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2825-2.

[93] R. Brockers, M. Hund, and B. Mertsching. Stereo vision using cost-relaxation with 3d support regions. In *Proceedings of the IEEE International Conference on Image Processing*, pages 389 – 392, 2005.

[94] M. Gong and Y.-H. Yang. Near real-time reliable stereo matching using programmable graphics hardware. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2005*, volume 1, pages 924–931, 20–25 June 2005.

[95] O. Veksler. Stereo correspondence by dynamic programming on a tree. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 384–390, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2.

[96] Q. Yang, L. Wang, and N. Ahuja. A constant-space belief propagation algorithm for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[97] L. Nalpantidis and A. Gasteratos. Biologically and psychophysically inspired adaptive support weights algorithm for stereo correspondence. *Robot. Auton. Syst.*, 58(5):457–464, 2010. ISSN 0921-8890.

[98] C. Richardt, D. Orr, I. Davies, A. Criminisi, and N. A. Dodgson. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. In *Proceedings of the European Conference on Computer Vision*, 2010.

[99] O. Stankiewicz and K. Wegner. Depth map estimation software version 2. In *ISO/IEC MPEG meeting M15338*, 2008.

[100] D. Miyazaki, Y. Matsushita, and K. Ikeuchi. Interactive shadow removal from a single image using hierarchical graph cut. In *Proceedings of the Asian Conference of Computer Vision*, 2009.

[101] S. El-Etriby, A. K. Al-Hamadi, and B. Michaelis. Dense stereo correspondence with slanted surface using phase-based algorithm. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 1807 – 1813, 2007.

[102] S. El-Etriby, A. K. Al-Hamadi, and B. Michaelis. Dense depth map reconstruction by phase difference-based algorithm under influence of perspective distortion. In *Proceedings of the International Conference on Computer Vision and Graphics*, pages 349 – 361, 2006.

[103] L. Nalpantidis and A. Gasteratos. Stereo vision for robotic applications in the presence of non-ideal lighting conditions. *Image and Vision Computing*, 2009.

[104] G. Olague, F. Fernandez de Vega, C. B. Perez, and E. Lutton. The infection algorithm: An artificial epidemic approach for dense stereo matching. *LNCS Parallel Problem Solving from Nature - PPSN VIII*, 3242:622–632, 2004.

[105] M. Humenberger, D. Hartermann, and W. Kubinger. Evaluation of Stereo Matching Systems for Real World Applications Using Structured Light for Ground Truth Estimation. In *Proc. of the IAPR Conference on Machine Vision and Applications*, 2007.

[106] K. Ambrosch. *Mapping Stereo Matching Algorithms to Hardware*. PhD thesis, Vienna University of Technology, 2009.

[107] Mobile robots inc. mobileranger, datasheet. URL `http://www.activrobots.com/ACCESSORIES/MobileRanger SpecSheet.pdf`.

[108] VidereDesign. *Stereo-on-a-Chip Stereo Head User Manual 1.3*. Videre Design, 2007. URL `http://www.videredesign.com/vision/stoc.htm`.

[109] Y. Miyajima and T. Maruyama. A real-time stereo vision system with fpga. In *Field-Programmable Logic and Applications*, volume 2778 of *Lecture Notes in Computer Science*, pages 448–457. Springer Berlin / Heidelberg, 2003.

[110] R. Yang, M. Pollefeys, and S. Li. Improved real-time stereo on commodity graphics hardware. In *Computer Vision and Pattern Recognition Workshop*, volume 3, pages 36–42, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2158-4.

[111] *Triclops, Technical Manual*. Point Grey Research Inc., 2004. URL `http://www.ptgrey.com/products/triclopsSDK/triclops.pdf`.

[112] J. I. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck. The Tyzx DeepSea G2 Vision System, A Taskable, Embedded Stereo Camera. In *Proceedings of the 2006 Conference on Computer Vision and Pattern Recoginition Workshops*, 2006.

[113] N. Chang, T.-M. Lin, T.-H. Tsai, Y.-C. Tseng, and T.-S. Chang. Real-time dsp implementation on local stereo matching. In *Proc. IEEE International Conference on Multimedia and Expo*, pages 2090–2093, 2–5 July 2007.

[114] J. Woodfill and B. Von Herzen. Real-time stereo vision on the parts reconfigurable computer. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 242–250. IEEE Computer Society Press, 1997.

[115] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 196–202, 1996.

[116] S. Kimura, T. Shinbo, H. Yamaguchi, E. Kawamura, and K. Naka. A convolver-based real-time stereo machine (sazan). In *Computer Vision and Pattern Recognition*, pages 457–463, 1999.

[117] B. Khaleghi, S. Ahuja, and Q. Wu. An improved real-time miniaturized embedded stereo vision system (mesvs-ii). In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops CVPR Workshops 2008*, pages 1–8, 23–28 June 2008.

[118] O. Faugeras, B. Hotz, H. Mathieu, T. Vieville, Z. Zhang, P. Fua, E. Theron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real-time correlation based stereo: algorithm implementations and applications. Technical Report 2013, INRIA, 1993.

[119] K. Ambrosch, M. Humenberger, and S. Olufs. *Smart Cameras*, chapter Chapter 5: Embedded Stereo Vision. Springer Verlag, 2009.

[120] J. Banks and P. Corke. Quantitative evaluation of matching methods and validity measures for stereo vision. *International Journal of Robotics Research*, 20:512–532, 2001.

[121] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In *Proceedings of the 2007 Conference on Computer Vision and Pattern Recoginition*, 2007.

[122] H. Hirschmueller and D. Scharstein. Evaluation of cost functions for stereo matching. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition CVPR '07*, pages 1–8, 17–22 June 2007.

[123] K. Konolige. Small vision system: Hardware and implementation. In *Proceedings of Eighth International Symposium on Robotics Research*, Japan, 1997. URL `citeseer.ist.psu.edu/konolige97small.html`.

[124] J. I. Woodfill, B. v. Herzen, and R. Zabih. Frame-rate robust stereo on a pci board. 1998.

[125] AMD. *Software Optimization Guide for AMD64 Processors*. Advanced Micro Devices, Inc., rev. 3.06 edition, September 2005.

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

[127] C. Zinner, W. Kubinger, and R. Isaacs. Pfelib: A performance primitives library for embedded vision. *EURASIP Journal on Embedded Systems*, 2007 (1):14, 1 2007. ISSN 1687-3955.

[128] *Intel Integrated Performance Primitives for Intel Architecture*. Intel Corporation, 2007. Document Number:A70805-021US.

[129] A. Kuznetsov. Bitmagic library: Sse2 optimization, 10 2008. URL `http://bmagic.sourceforge.net/bmsse2opt.html`.

[130] C. Zinner, M. Humenberger, K. Ambrosch, and W. Kubinger. An optimized software-based implementation of a census-based stereo matching algorithm. In *Advances in Visual Computing, Lecture Notes in Computer Science*, volume 5358, pages 216–227. Springer, 2008. ISBN 978-3-540-89638-8.

[131] M. Houston. High level languages for gpus overview. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 5, New York, NY, USA, 2007. ACM.

[132] NVIDIA. *GeForce 9800 GT*. NVIDIA Corporation, 11 2008. URL `http://www.nvidia.com/object/product_geforce_9800gt_us.html`.

[133] NVIDIA. *GeForce GTX 280*. NVIDIA Corporation, 11 2008. URL `http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf`.

[134] PCI-SIG. Pci express specifications, 2009. URL `http://www.pcisig.com/specifications/pciexpress/specifications/`.

[135] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *International Journal of Computer Vision*, 75(2):283–296, 2007. ISSN 0920-5691.

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

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

[138] C. Zinner and W. Kubinger. Ros-dma: A dma double buffering method for embedded image processing with resource optimized slicing. In *Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 361–372, 04–07 April 2006.

[139] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485. ACM, 1967.

[140] K. S. Gatlin and P. Isensee. Openmp and c++: Reap the benefits of multithreading without all the work. *MSDN Magazine*, 2005. URL `http://msdn.microsoft.com/en-us/magazine/cc163717.aspx`.

[141] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. 24(5):603–619, 2002.

[142] M. Bleyer and M. Gelautz. A layered stereo matching algorithm using image segmentation and global visibility constraints. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59:128–150, 2005.

Curriculum Vitae

Martin Humenberger was born on June 1st, 1981 in Vienna. After secondary school in the *BG6 Amerlingstrasse* he fulfilled the Austrian military forces in Baden. In 2000 he attended the course of study *Hardware/Software Systems Engineering* at the *Upper Austrian University of Applied Science, Hagenberg*. He finished with a degree of *Dipl.-Ing. (FH)* in 2004. During the last semesters he started to work for the *AIT Austrian Institute of Technology* where he also wrote his diploma theses entitled *"Development of a TT-Vision Node Platform"*. In 2005 he started his PhD studies at the *Vienna University of Technology* on the *Automation and Control Institute (ACIN)*. Employed at the AIT as PhD student, he was part of the European Union funded project robots@home under grant FP6-2006-IST-6-045350 which has been finished in 2010. Now he works at the AIT on stereo vision for fall detection in systems for ambient assisted living. His main research interests are stereo vision and 3D reconstruction for scene representation, interpretation, and three dimensional modeling.