The University of Western Australia
Faculty of Engineering, Computing and Mathematics
School of Electrical, Electronic and Computer Engineering
Centre for Intelligent Information Processing Systems

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Theoretische und Technische Informatik
Fachgebiet Neroinformatik und Kognitive Robotik

Comparison of Stereo Matching Algorithms for Mobile Robots

by Annika Kuhl

October 2004 - February 2005

1 Abstract

For navigation and obstacle avoidance in the field of mobile robots, perception and identification of the surrounding environment is necessary. Depth maps therefore provide an essential description of the world seen through cameras.

This thesis will investigate with different types of stereo matching algorithms for calculating depth maps. The task is to build and to implement a system for generating depth maps on a mobile robot.

Many stereo algorithms make use of the epipolar constraint, meaning that for a pixel in the left image the corresponding point in the right image lies on the same horizontal line, the epipolar line. This strong constraint is used to reduce the search space of the correspondence algorithms that calculates depth maps.

To make use of the epipolar constraint the camera system needs to be calibrated first, to get the intrinsic and extrinsic camera parameters, in order to rectify images according to these parameters. The first chapters therefore deal with the necessary pre-processing steps to calibrate cameras and to rectify images.

Contents

2 System Overview

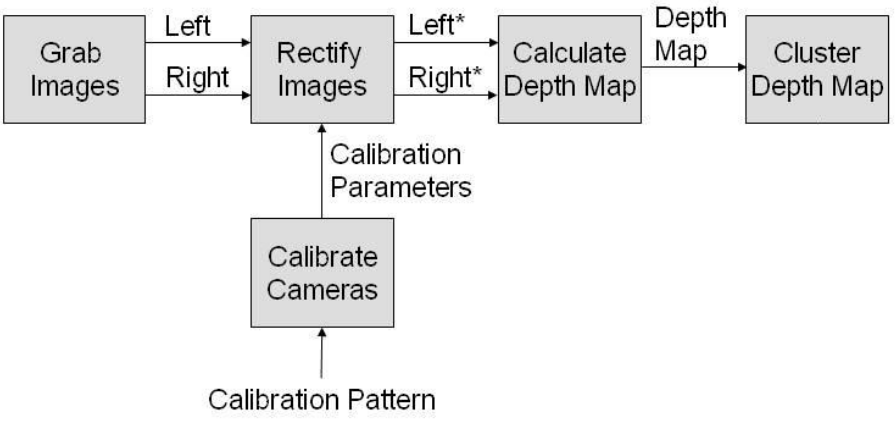

Figure 1: System overview

The system for calculating depth maps starts with grabbing the images. Two USB web cameras on top of the robot view the scene from a left and right point of view.

Due to lens distortion and camera displacements the next step is to rectify the images (see chapter 3.6) in order to make use of the epipolar constraint as described in chapter 2.2. Therefore both cameras need to be calibrated first to get the camera parameters that are explained in section 3.2. The Bouguet method, implemented in Matlab, is used for calibration (see section 3.5).

The knowledge of the camera parameters is used to rectify both images. After rectification several stereo matching algorithms will calculate the depth map of the surrounding scene. The algorithms implemented and tested in this thesis are Sum of Absolute Differences, Sum of Squared Differences and Census. The results are shown in section 4.

The final step is to cluster the resulting depth map. This is described in section 5.3.

After implementing the three above mentioned algorithms on a PC the best was chosen to be implemented on the robot. A discussion of these results and a comparison to a feature based algorithm that came with the robot can be found in chapter 6.

2.1 Correspondence Analysis

Correspondence analysis tries to solve the problem of finding which pixels or objects in one image correspond to a pixels or objects in the other. This is known as the Correspondence Problem. The algorithms can roughly be divided into *feature based* and *area based*, also known as region based or intensity based.

Area based algorithms solve the correspondence problem for every single pixel in the image. Therefore they take colour values and / or intensities into account as well as a certain pixel neighbourhood. A block consisting of the middle pixel and its surrounding neighbours will then be matched to the best corresponding block in the second image.

These algorithms result in dense depth maps as the depth is known for each pixel. But selecting the right block size is difficult because a small neighbourhood will lead to less correct maps but short run times whereas a large neighbourhood leads to more exact maps at the expense of long run times.
Typical algorithms are Marr and Poggio (1979), Grimson (1981) or the algorithm of Ohta and Kanade (1985) which uses dynamic programming.

Feature based correspondence algorithms on the other hand extract features first and then try to detect these features in the second image. These features should be unique within the images, like edges, corners, geometric figures, hole objects or part of objects.

The resulting maps will be less detailed as the depth is not calculated for every pixel. But since it is much more unlikely to match a feature incorrectly because of its detailed description, feature based algorithms are less error sensitive and result in very exact depth maps.
A typical algorithm is described by Okutomi and Kanade (1992).

Besides the major correspondence algorithms, area based and feature based, there are also *phase based* algorithms that transform the images using FFT (fast fourier transformation) first. The depth is therefore proportional to the phase displacement. *Wavelet based* algorithms are a subcategory of phase based algorithms and use a wavelet transformation first.

There are a number of problems all correspondence analysis algorithms have to deal with. An object seen by one of the cameras could be occluded when seen by the other camera that has a slightly different point of view. This object will cause wrong correspondences when trying to match images.

The cameras itself my cause distorted images due to lens distortion which will lead to wrong correspondences especially in the outer regions of the image.

Some more problems are caused by the objects themselves. Having lots of small objects that look alike or having a special pattern that iterates quite often makes it hard to find the matching object as there is more than one possible match. This is known as the aperture problem.

Another big problem is homogeneity. Big homogeneous regions are difficult to match when seen through a small window only. The same textures on different positions in the image will cause similar problems.

There are a number of constraints that ease the corresponding problem and improve the results. According to [9] these are:

- *Similarity constraint*: the matching pixel / feature must have similar intensities / attribute values
- *Uniqueness constraint*: a given pixel / feature can match no more than one pixel / feature
- *Continuity constraint*: disparity should vary smoothly over the image
- *Ordering constraint*: if pixel / feature a is left to b the matching pixel / feature a' needs to be left to b'

Another possibility to improve these correspondence algorithms are some pre-processing steps like the pre-reduction of noise with a low-pass filter, the adjustment of different illuminations or a white balance of each camera. But the most effective pre-processing step is the calibration of the cameras and the use of the epipolar constraint which will be described throughout the next chapters.

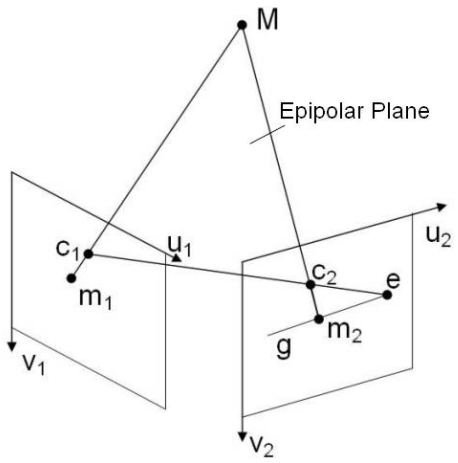

Figure 2: General epipolar geometry

2.2 Epipolar Geometry

With two cameras arranged arbitrarily, the general epipolar geometry is shown in figure 2. The relative position of both cameras is known and C_1 and C_2 point out the optical centres of each camera. The straight line connecting both optical centres is called baseline.

Each point M observed by the two cameras at the same time along with the two corresponding light rays through the optical centres C_1 and C_2 form an epipolar plane.

The epipole e is the intersection of the baseline with the image plane. The epipolar line is therefore defined as a straight line g through e and m that is the intersection of the line through M and the optical centre with the respective image plane.

The point M in figure 2 is projected as m_1 in the left image plane. The corresponding point in the right image therefore lies on the previous described epipolar line g. This reduces the search space from two dimensional, which would be the whole image, to one dimensional, a straight line only.

A simplification of the general epipolar geometry is shown in figure 3. Both cameras are arranged in parallel, their focal length is identical and the two retinal planes are the same. Assuming these conditions all epipolar lines are horizontal within the retinal planes and the projected images m_1 and m_2 of a point M will have the same vertical coordinate. Therefore the

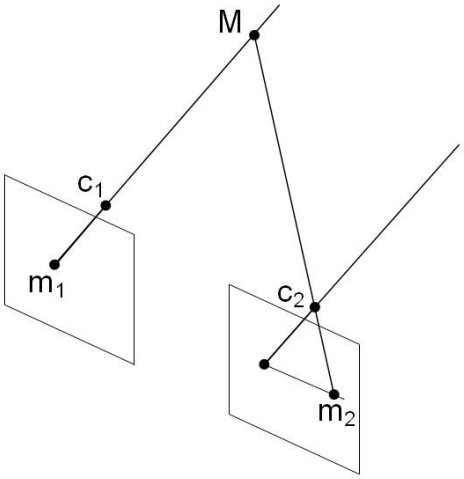

Figure 3: Stereo epipolar geometry

corresponding point of m_1 lies on the same horizontal line in the right image.

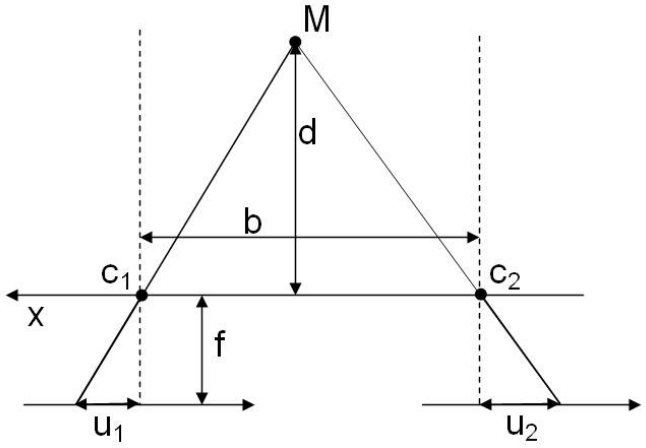

Figure 4: Disparity

According to the stereo epipolar geometry the disparity as seen in figure 4 is defined as $D = c_2 - c_1$. The depth d therefore is calculated by triangulation

$$d = b\frac{f}{D} \tag{1}$$

where b is the distance of the two optical centres and f is the focal length. A disparity of zero indicates that the depth of the appropriate point equals

infinity.

In order to assure the stereo epipolar geometry the rectification of both images is necessary. Therefore both cameras need to be calibrated first in order to get the camera parameters that are needed for the rectification. The next section will deal with the problem of camera calibration and image rectification.

3 Image Preprocessing

Image pre-processing is an essential step to simplify the correspondence problem. As mentioned in section 2.1 the reduction of noise or a white balance of each camera are possibilities therefore.

The stereo epipolar geometry as described in section 2.2 is a strong constraint to reduce the search area from two dimensional to a one dimensional horizontal line. This chapter will deal with the problem of calibrating cameras and rectifying images to assure this constraint.

3.1 The Pinhole Camera Model

Every camera maps the points of the three dimensional environment to a two dimensional image. The simplest camera model that models this mapping is the pinhole camera model.

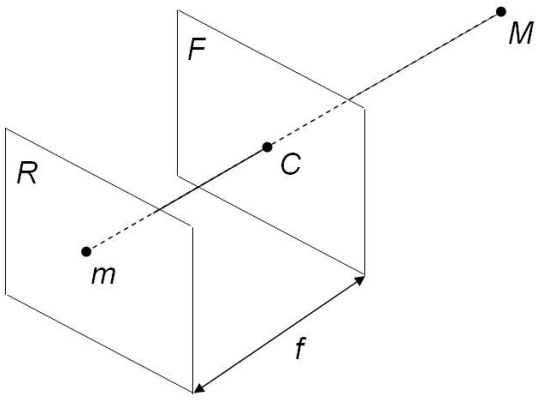

Figure 5: The pinhole camera model

As shown in figure 5 the pinhole camera model consists of two screens.

R is The retinal plane where the two dimensional image is formed, F is the focal plane with the optical centre C in the middle. Both planes are parallel at a certain distance f which is the focal length.
The straight line going through a point M of the three dimensional world and C is called the optical axis. Via perspective projection this point M is mapped onto the two dimensional image.
Points on the focal plane have no image on the retinal plane as there is no intersection of the optical axis as it is parallel to the retinal plane.

3.2 Camera Parameters

In order to transform a point of the three dimensional world into a two dimensional point of the image the knowledge of special camera parameters is necessary. There are two kinds of camera parameters. The intrinsic or internal parameters that describe the internal geometric and optical characteristics of the camera, and the extrinsic or external parameters defining the position and orientation of the camera in a world reference system.

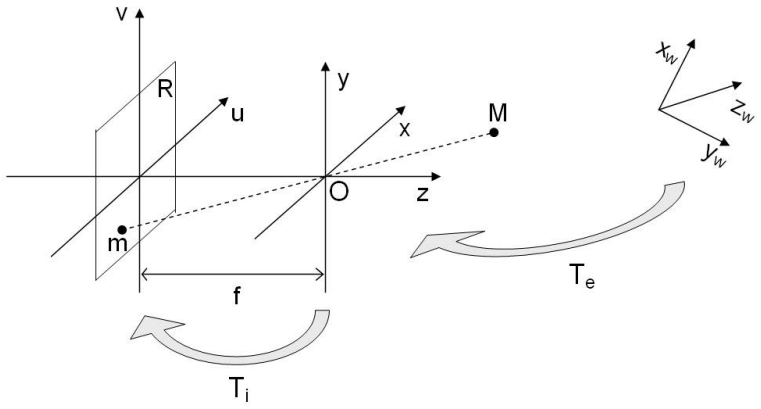

Figure 6: Intrinsic and extrinsic parameters

As seen in figure 6 the system for modelling two or more cameras consists of three different coordinate systems, the world reference frame (x_w, y_w, z_w), the camera frame (x, y, z) with the optical centre as origin and the image frame (u, v).
A three dimensional point given in homogeneous world coordinates can be converted into the camera frame by a rotation r_{ij} and a translation t_j which

is expressed by the extrinsic parameters T_e.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = T_e \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \quad T_e = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2}$$

Then this point is converted to the two dimensional image plane using the intrinsic parameters. These are in particular the focal length *f*, the principle point (u_0, v_0), which is the centre of the image plane, and (k_0, k_1) the pixel size in mm or $\alpha = f/k_0$ and $\beta = f/k_1$ respectively. The transformation using the intrinsic parameters is as follows:

$$\begin{bmatrix} u \\ v \\ s \end{bmatrix} = T_i \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad T_i = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Since (u, v, s) is homogeneous, all three numbers are divided by s in order to get pixel coordinates *u'* and *v'*.

Points on the focal plane, where $z = 0$ and $s = 0$ respectively, can not be transformed to image plane coordinates as division by zero is not defined and the straight line going through this point and the optical centre does not intersect with the image plane as it is parallel to the image plane.

In summary, a point given in world coordinates is transformed onto a two dimensional image plane using the following equation:

$$\begin{bmatrix} u \\ v \\ s \end{bmatrix} = T_i T_e \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{4}$$

The knowledge of the intrinsic and extrinsic camera parameters allows for the rectification of images and ensures the epipolar constraint. The calculation of these parameters is the aim of the camera calibration and will be discussed within the next chapters.

3.3 Camera Calibration using CamChecker

CamChecker [7] is a free camera calibration tool that calculates the intrinsic parameters of a camera. It is based upon the work of Zhang [11] and therefore works with a chessboard as calibration pattern. It uses the camera model defined by Zhang, which is an extension of the simple pinhole camera model described in section 3.1. However, it also takes lens distortion into account, which makes image undistortion more accurate.

For camera calibration at least 5, and up to 20, pictures of the chessboard need to be taken. The software then extracts the corners of the board automatically by thresholding the images to black and white. According to [7] the intrinsic parameters are then calculated using Zhang's approach as follows

1. Finding a closed-solution homography for each image
2. Using these to find a closed solution for the intrinsics/extrinsics
3. Optimizing these intrinsics/extrinsics with Levenberg-Marquardt routines found in the GNU Scientific Library

This tool can be downloaded as win32 executable or as source code for Visual Studio 6 and 7. I also adapted the code to compile it under Suse Linux.

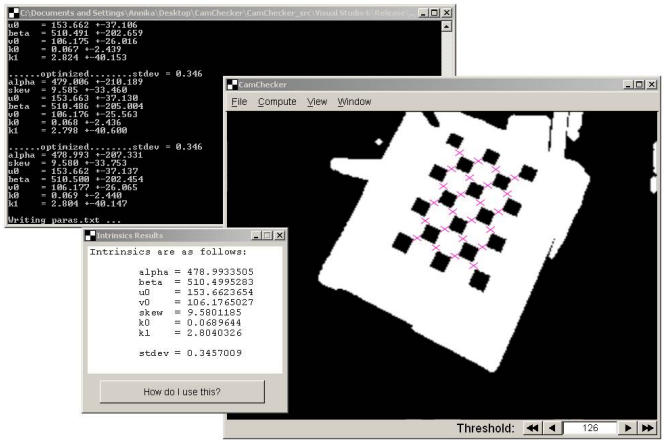

Figure 7: CamChecker software that calculates the intrinsic parameters

After calculating the intrinsic parameters for each camera the images can now be undistorted. For this a helpful little program from the Hugin

[2] project of Pablo d'Angelo is used. The code file zhang_undistort.cpp takes all the intrinsic parameters calculated by CamChecker and undistorts a given image according to Zhang's model.

Using these programs each camera can be calibrated independently. But to assure the epipolar constraint with aligning horizontal image lines the two cameras need to be calibrated together. Therefore the camera calibration is done by using the Matlab implementation of Bouguet's calibration algorithm (see section 3.5).

3.4 Epipolar Constraint

Knowing the intrinsic and extrinsic parameters of each camera separately is sufficient to undistort the appropriate images but it is not sufficient to assure the epipolar constraint. The epipolar constraint ensures that the epipolar lines coincide with the horizontal scan lines, and therefore corresponding points in both images are only horizontally shifted, which reduces the search space from three dimensional to two dimensional.

Given the two cameras, every point $w = [x_w\ y_w\ z_w\ 1]^T$ of the world reference frame can be projected to the appropriate image frame (m_1 and m_2) using the linear transformation matrix $P = T_i\ T_e$ as known from section 3.2 as follows

$$m_1 = P_1\ w, \quad m_2 = P_2\ w \tag{5}$$

In order to rectify the images according to the epipolar constraint these projection matrices P_1 and P_2 have to be adapted following special conditions that are:

- both camera systems need to have equal focal length
- both camera systems need to have the same focal plane
- the optical centres need to kept constant
- correspondence of the vertical coordinate of each point in the left and the right image

Using these and further conditions P_1 and P_2 can be calculated and the taken images can be transformed according to the epipolar constraint.

A further and more detailed description may be found in the appendix of the thesis of Nico Kämpfchen [5].

3.5 Camera Calibration á la Bouguet

Camera calibration is an essential preliminary step of calculating depth maps. According to the chosen camera model the intrinsic and extrinsic camera parameters as described in 3.2 need to be acquired. Jean-Yves Bouguet's developed a camera model that is based on the Tsai / Lenz camera model and extends it.

The Tsai / Lenz camera model improves the simple pinhole model that is described in 3.1. It is more suitable for precise tasks as it takes circular distortion caused by the lens system into account. Points of the world reference frame are transformed to the image frame using the extrinsic and intrinsic parameters from equation 2 and 3.

$$\begin{bmatrix} u \\ v \\ s \end{bmatrix} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{6}$$

In order to get pixel coordinates *u'* and *v'*, u and v are divided by s. Then the radial distortion first and second order (κ_0, κ_1) are taken into account by

$$d_1 = \left(\frac{u - u_0}{\alpha}\right)^2 + \left(\frac{v - v_0}{\beta}\right)^2 \tag{7}$$

$$d_2 = d_1^2 \tag{8}$$

$$u_{undistorted} = u' + (u' - u_0)(\kappa_0 d_1 + \kappa_1 d_2) \tag{9}$$

$$v_{undistorted} = v' + (v' - v_0)(\kappa_0 d_1 + \kappa_1 d_2) \tag{10}$$

The camera model of Bouguet [3] extends the Tsai / Lenz model and is inspired by Heikkilä and Silvén [4]. It accounts for radial distortion first, second and third order, tangential asymmetric distortion, skew and affinity,

which makes image rectification much more precise. The intrinsic parameters are therefore extended by the parameters for the lens distortion.

To obtain both intrinsic and extrinsic parameters of the camera system Bouguet adapted the method of Zhang [11] that uses a chess board as calibration pattern. According to Zhang the process of camera calibration can be divided into three steps:

1. Image acquisition
2. Extraction of the chess board corners in each image
3. Computing the intrinsic and external parameters

For image acquisition the two cameras on top of the robot are used to grab several pictures of a chess board from different distances and angels simultaneously (see figure 8).

Figure 8: Images taken from the chess board

The extraction of the chess board corners needs to be done manually within the Matlab implementation of Bouguet's calibration algorithm [3]. After that the computing of the intrinsic and extrinsic parameters is done

automatically. First every camera is calibrated separately before the global extrinsic parameters are calculated.

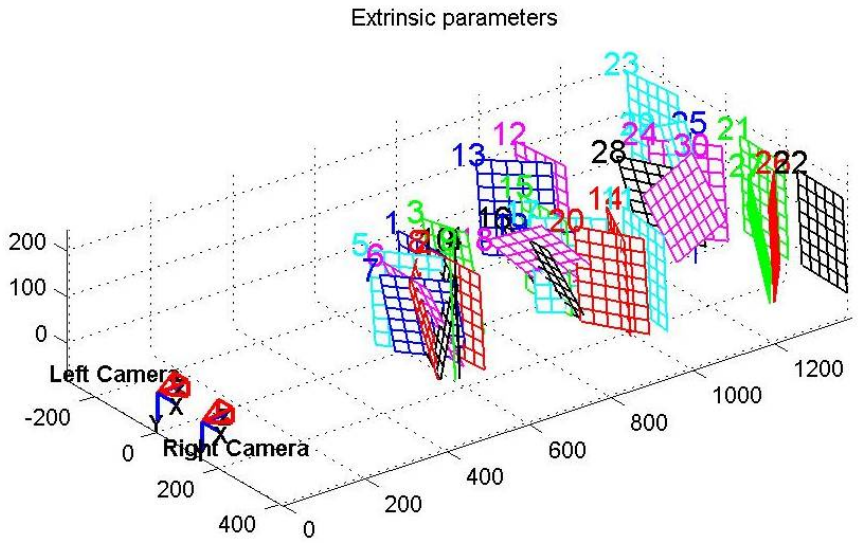

Figure 9: Extrinsic parameters

Figure 9 shows the extrinsic parameters of the camera system and the position of the 30 chess boards in front of the cameras within the pictures seen in figure 8. The world coordinates of the chess board corners relative to each camera are known and the position of the right camera with respect to the left camera is calculated.

3.6 Image Rectification

Image rectification is the undistortion according to the calibration parameters calculated in the camera calibration step in section 3.5.

After all intrinsic and extrinsic camera parameters are calculated they can be used to rectify images according to the epipolar constraint. Therefore Bouguet's algorithm pre-computes the necessary indices and blending coefficients to enable a quick rectification afterwards. The images are then rectified by:

$$I_{new}(x_0, y_0) = a_1 I_{old}(x_1, y_1) + a_2 I_{old}(x_2, y_2) + a_3 I_{old}(x_3, y_3) + a_4 I_{old}(x_4, y_4) \tag{11}$$

with I_{new} and I_{old} as the original and the rectified image and the blending coefficients a_i separate for each camera.

After pre-computing, the parameters x_i, y_i and a_j are then saved to a text file. To be independent from Matlab these parameter files, one for each camera, are reloaded using C++ or any other programming language. Once reloaded, these parameters are used according to equation 11 to rectify images quick and easy.

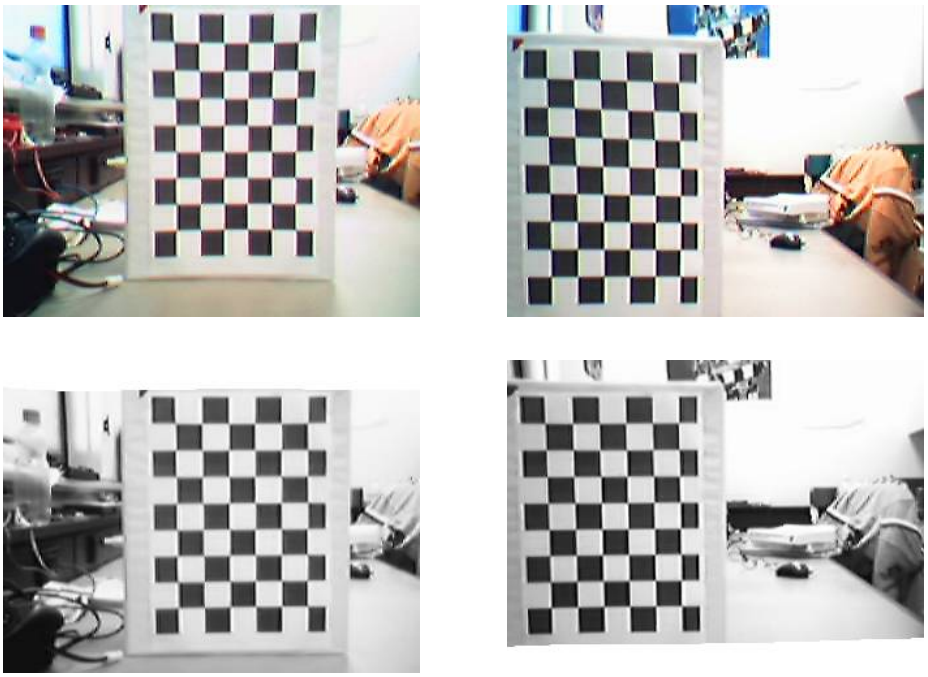

Figure 10: Left and right image before (upper) and after (lower) image rectification

Figure 10 shows the images taken by the left and right camera before and after the rectification. The upper images show a great vertical displacement indicate by the chessboard due to the camera mounting system.

After calculating the intrinsic and extrinsic parameters both stereo images need to be rectified according to these parameters. This way both images fulfil the epipolar constraint, meaning that the corresponding pixels lie on the same horizontal line in both images.

The result of this rectification process is shown in the lower two images. The vertical displacement vanished and every chessboard square has the

same hight in both images. Drawing a straight horizontal line through both images every pixel in the left image on that line will have its corresponding pixel of the right image on that same line.

The image rectification causes a reduction of the image size. The left image is shifted downwards and the right image is shifted upwards due to the mentioned displacement. Therefore the image size is reduced from 320 by 240 to about 320 by 185 as the upper and lower strip will be cut off in both images.

4 Implemented Stereo Matching Algorithms

This section describes the implementation and results of the following stereo matching algorithms using the epipolar constraint:

- Census
- Census Modification
- Sum of Absolute Differences
- Sum of Squared Differences

Subsection 4.6 comprises a comparison of all these algorithms concerning runtime and depth map quality. The best one is chosen for some improvements in section 5.

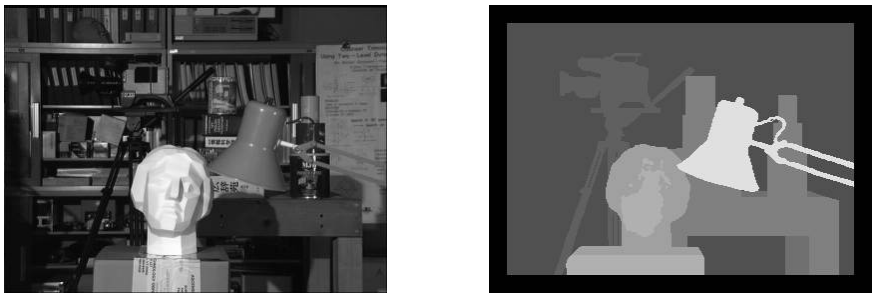

Figure 11: Test image *Tsukuba* (left) and groundtruth (right)

In order to compare the results of the different stereo matching algorithms one fixed test image is chosen. Throughout this section the image *Tsukuba* and its ground truth is used to allow a true comparison between the different algorithms.

4.1 Software Architecture

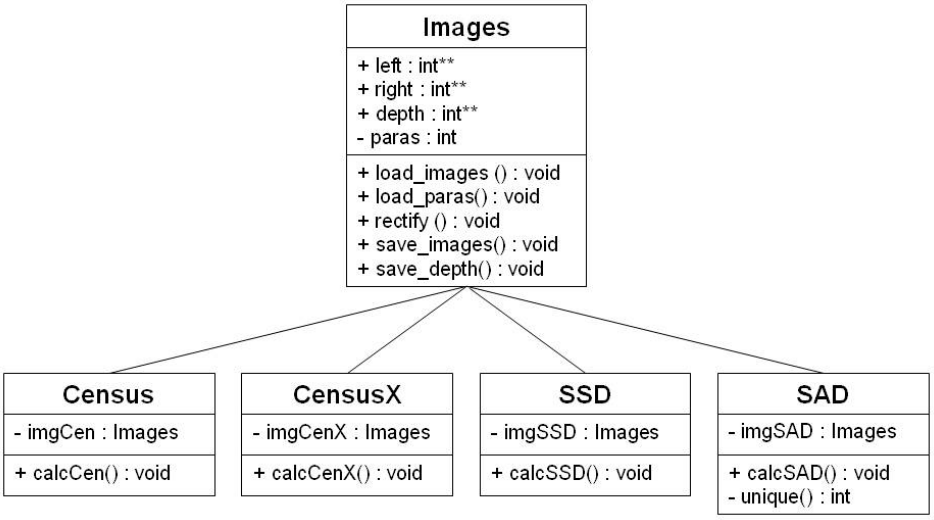

Figure 12: Simplified UML class diagram of the software architecture

In order to make software programming easier the class diagram in figure 12 is designed before implementation. The class *Images* is used to handle all functions to open and save images and files. After reading in the left and the right image and the parameter files, respectively, the function *rectify* will undistort and rectify both images.

For loading and saving files and image processing the computer vision library ***VIGRA*** - Vision with Generic Algorithms [6] is used. It provides functions and data structures to access and handle images easily and to adopt own algorithms.

All algorithms in this chapter are run on a Pentium M (Centrino) laptop with 1500 MHz and 512 MB RAM running Windows XP.

4.2 Census

Census is an area-based correspondence algorithm described by Zabih and Woodfill [10]. The algorithm computes the displacement in two steps. First the so called *Census Transform* is defined for each pixel in both images. It describes the relationship between that pixel and its surrounding neighbourhood. Neighbour pixels with an intensity smaller than the centre pixel

result in zero in the *Census Vector* and one otherwise (see figure 13).

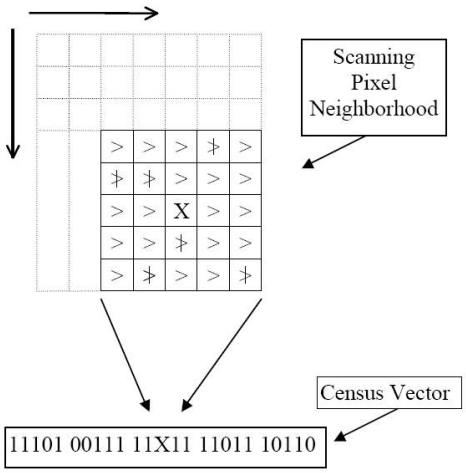

Figure 13: Census vector

The second step computes the displacement by summing up the Hamming distances of a small window in the left and in the right image. The minimum for each horizontal line defines the displacement.

The algorithm is implemented in C++ and tested on the Tsukuba picture in figure 11. The results are shown in table 1 where *CT-Window* is the size of the window used to generate the Census Vector and *Window* is the window size for computing the pixel displacements.

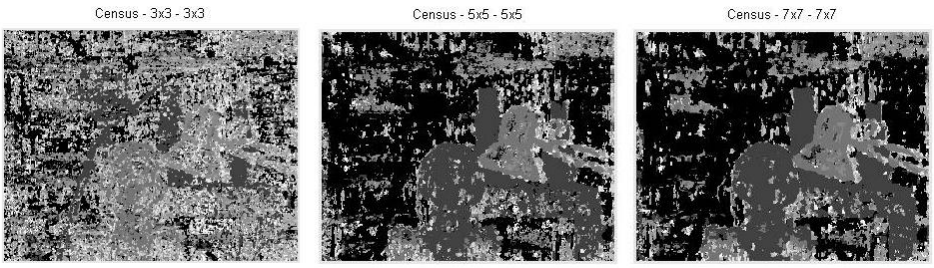

Figure 14: Depth map of the Census algorithm with CT-Window and Window of 3x3 (left), 5x5 (center) and 7x7 (right)

Resolution	CT-Window	Window	Time in sec.
384 x 288	3x3	3x3	82
384 x 288	3x3	5x5	205
384 x 288	3x3	7x7	390
384 x 288	5x5	3x3	177
384 x 288	5x5	5x5	468
384 x 288	5x5	7x7	879
384 x 288	7x7	3x3	301
384 x 288	7x7	5x5	903
384 x 288	7x7	7x7	1702
192 x 144	3x3	3x3	10
192 x 144	3x3	5x5	24
192 x 144	3x3	7x7	69
192 x 144	5x5	3x3	20
192 x 144	5x5	5x5	53
192 x 144	5x5	7x7	78

Table 1: Results of the Census implementation

4.3 Census Modification

In order to improve the runtime of the Census algorithm a modification is implemented. Therefore the window size that defines the Census Vector is increased. This vector contains the information of all surrounding pixels which should be sufficient to compute the best matching pixel in the right image without an extra second window. The corresponding pixel in the right image is now defined as the minimum of the Hamming distance of a single pixel at the epipolar line.

This modification of the Census algorithm is much faster, but the resulting depth maps are reduced in quality as show in figure 15.

The poor quality is due to the reduced intensity information. Instead of the exact intensity values the algorithm gets binary information only - zero if the intensity is smaller than in the centre and one otherwise. This information is insufficient to compute proper depth maps.

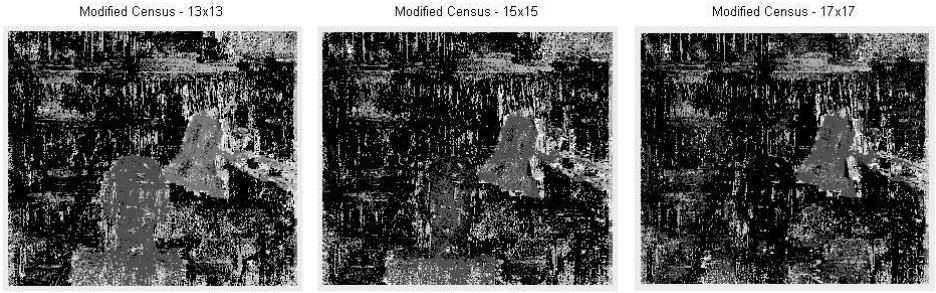

Figure 15: Depth map of the modified Census algorithm with CT-Window of 13x13 (left), 15x15 (center) and 17x17 (right)

4.4 Sum of Absolute Differences

The SAD algorithm is also an area-based correspondence algorithm. It computes the intensity differences for each center pixel (i, j) in a window v_x by v_y as follows

$$SAD_v(x,y) = \sum_j \sum_i ||g_t(x+i, y+j) - g_{t-1}(x+v_x+i, y+v_y+j)||$$

It sums up the intensities of all surrounding pixels in the neighbourhood for each pixel in the left image. The absolute difference between this sum and the sum of the pixel, and its surrounding, in the right image is calculated. The minimum over the row in the right image is chosen to be the best matching pixel. The disparity then is calculated as the actual horizontal pixel difference.

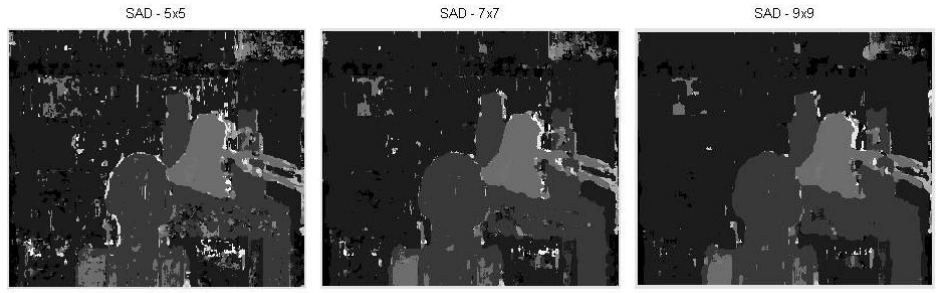

Figure 16: Depth maps of the SAD implementation with window sizes of 5x5(left), 7x7(centre) and 9x9(right)

Resolution	Window	Time in sec.
384 x 288	3x3	53
384 x 288	5x5	66
384 x 288	7x7	83
384 x 288	9x9	108
192 x 144	3x3	7
192 x 144	5x5	8
192 x 144	7x7	10
192 x 144	9x9	13

Table 2: Results of the SAD implementation

This implementation of the SAD algorithm results in proper depth maps as seen in figure 16. The bigger the window the better the depth map as there is a greater neighbourhood to ensure finding the correct corresponding pixel. But the bigger the window the slower is the algorithm, as shown in table 2. Depending on the application a compromise between runtime and quality needs to be chosen.

4.5 Sum of Squared Differences

The area-based SSD algorithm is similar to the previously described SAD. Instead of computing the absolute value, SSD squares the intensity differences as follows

$$SSD_v(x,y) = \sum_j \sum_i (g_t(x+i, y+j) - g_{t-1}(x+v_x+i, y+v_y+j))^2$$

The runtime of the SSD algorithm is nearly double that of the SAD. This is due to the much slower square calculation. The computer-internal implementation of the square operation is a multiplication which takes much more time than the simple comparison needed to calculate the absolute value.

The resulting depth maps in figure 17 are quite similar to those of the SAD algorithm.

Resolution	Window	Time in sec.
384 x 288	3x3	148
384 x 288	5x5	322
384 x 288	7x7	580
384 x 288	9x9	915
192 x 144	3x3	18
192 x 144	5x5	39
192 x 144	7x7	69
192 x 144	9x9	107

Table 3: Results of the SSD implementation

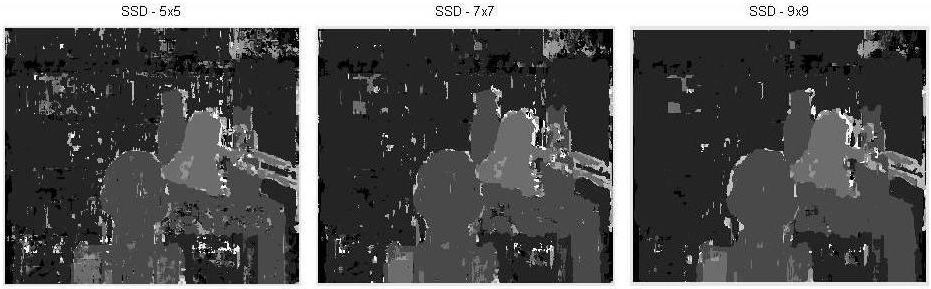

Figure 17: Depth maps of the SSD implementation with window sizes of 5x5(left), 7x7(center) and 9x9(right)

4.6 Comparison and Conclusion

The best depth map of each of the implemented algorithms (Census, SAD and SSD) is shown in figure 18. It is easy to see that the implementation of the SSD and SAD results in much better depth maps than the Census algorithm.

A comparison of the runtime between SAD and SSD (see tables 2 and 3) leads to the selection of the SAD algorithm for further improvements, because the SSD algorithm needs nearly twice the time.

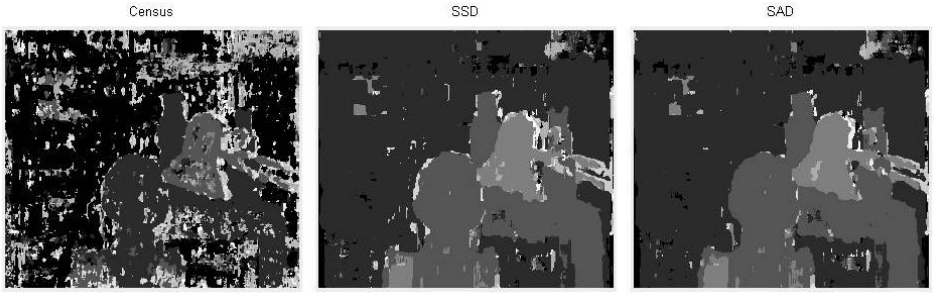

Figure 18: Comparison of the depth maps of Census (left), SSD (center) and SAD (right)

5 Algorithm Improvements

After choosing the best area based algorithm some improvements will be tested to achieve an even better performance and quality.

5.1 Reduction of Search Area

In order to improve the runtime of the algorithm a reduction of the search area is implemented. The search area is reduced to a ratio of the whole epipolar line.

Resolution	Ratio	Time in sec.	Resolution	Ratio	Time in sec.
384 x 288	1	108	192 x 144	1	10
384 x 288	2/3	83	192 x 144	2/3	10
384 x 288	2/4	74	192 x 144	2/4	9
384 x 288	2/5	66	192 x 144	2/5	8
384 x 288	2/6	62	192 x 144	2/6	8
384 x 288	2/7	59	192 x 144	2/7	7
384 x 288	2/8	55	192 x 144	2/8	7
384 x 288	2/9	54	192 x 144	2/9	7
384 x 288	2/10	52	192 x 144	2/10	6

Table 4: Different ratios of the searching line with a constant window size of 9x9

The resulting improvement in time is shown in table 4. Depending on

the maximum possible displacement in an image the *Ratio* could be reduced to 2/10 of the whole width.

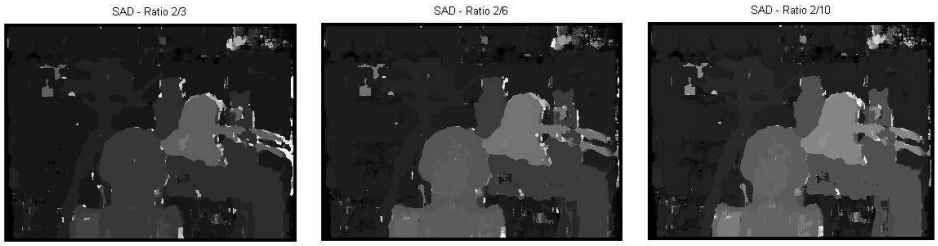

Figure 19: Different ratios of 2/3 (left), 2/6 (centre) and 2/10 (right)

The reduction of the search area also leads to an improvement of the resulting depth maps as shown in figure 19. The error rate is much smaller due to the elimination of impossible false out of range areas a priori.

5.2 Uniqueness Constraint

The uniqueness constraint validates the correctness of the corresponding pixel. It therefore again executes the stereo algorithm on the matching pixel found in the right image again in order to find the first pixel again.

This algorithm first searches, for each pixel in the left image, for the most similar one in the right image at the same epipolar line. After a winner is found the uniqueness is validated by applying the algorithm vice versa. It searches for the best matching pixel in the left image. If the position of the best matching pixel in the right image does not differ to that found again in the left image the calculated depth is valid. Otherwise the depth value is set to zero indicating invalid depths.

As seen in the chart in figure 20 the run times for the SAD algorithm applying the uniqueness constraint are more than doubled compared to SAD without uniqueness constraint. This is due to the second search along the epipolar line in the left image to validate the corresponding pixel that was found.

The uniqueness constraint leads to an improvement of the resulting depth maps as seen in figure 21. Much more false depth values can be filtered out and set to zero. This case leads to a sparse depth map due to invalid values.

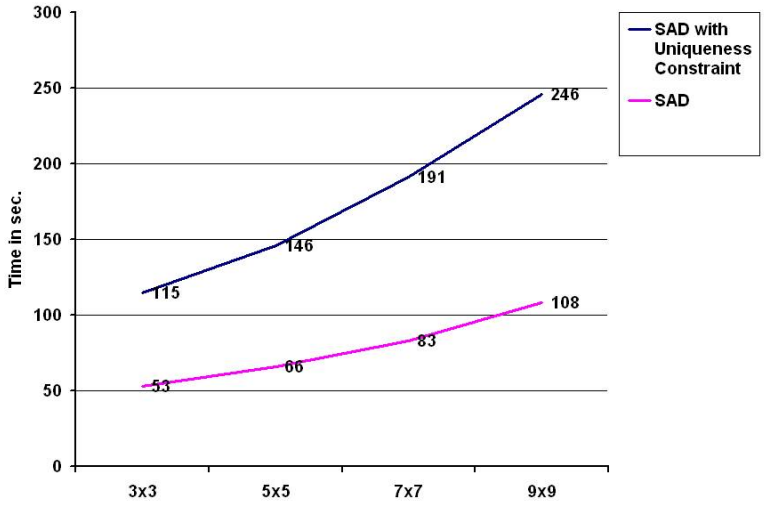

Figure 20: Comparison of SAD with and without uniqueness constraint

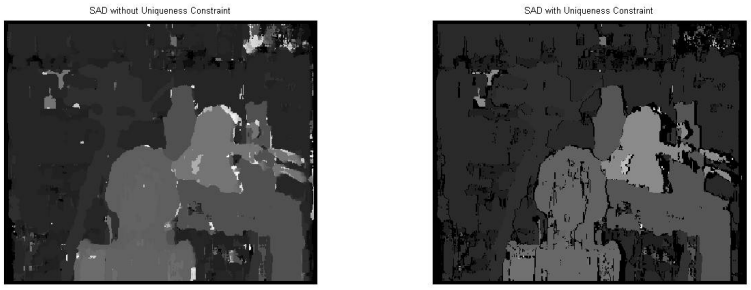

Figure 21: Comparison of SAD without uniqueness constraint (left) and with (right) both with a window size of 9x9 and search line ratio of 2/8

5.3 Clustering

Clustering is the last step according to the system overview in section 2. It simplifies the depth map by merging all depth steps to a certain number of depth cluster.

While calculating the depth map the maximal (max) and minimal (min) depths are memorised. After choosing the number of clusters they are calculated by

$$cluster(i) = floor\left(\frac{(i - min)\, numCluster}{max - min}\right) + 1 \qquad (12)$$

where i is the current depth, *numCluster* is the number of clusters and *cluster(i)* is the resulting new cluster.

Clustering is just a convenient addition to find obstacles much quicker and to simplify the depth map. It takes a loop over the whole image or ... seconds to calculate the clusters.

6 Robot Experiments

This section will deal with the actual implementation on a mobile robot. The algorithm chosen in section 4 is adapted to fit into the software already installed on the robot (see section 6.2). The mobile robot itself is described in detail in section 6.1. Finally some experiments are conducted to compare the feature based approach that was already implemented on the robot with this area based solution.

6.1 The Hummer Robot

The mobile robot was build by Jacky Baltes at the University of Manitoba in Canada [1]. It is an extended remote controlled Hummer H2 toy car.

Figure 22: The Hummer robot

A VIA Epia Mini-ITX motherboard at 533MHz is build in under the roof of the car. Instead of a hard drive there is a CompactFlash card reader

with a 256MB flash card accessible through the hatchback of the car. Due to the reduced drive space a mini version of the Debian Linux distribution was developed. A wireless USB device on top of the roof allows connecting to the robot via Laptop or PC. The vision hardware consists of two USB Intel Me2Cam cameras as there is a Linux driver support for the OV511+ chipset used in these cameras. They capture colour images of 320 by 240 pixel. The cameras are mounted on a servo motor that allows them to pan up to forty-five degrees in either direction. The stereo rig had to be improved to fulfil the requirements of the camera calibration. The original mounting was much too lose and the cameras changed their position to each other while panning or driving. The new rig fixes the cameras so the rectification is always valid once the cameras have been calibrated.

The remote control of the robot has been adapted and connected to the embedded motherboard. The robot can now be steered by any joystick plugged into a computer that is wirelessly connected to the robot.

6.2 Robot Software

The software that came with the robot is a feature based approach as described in [8]. In order to calculate the depth of the surrounding objects the algorithm performs 5 steps:

1. Colour Correction
2. Image Blur
3. Edge Detection
4. Region Extraction
5. Stereo Matching

The colour correction is necessary to correct possible imbalances in the colour response. Finding corresponding pixels / features is much more accurate on images with identical brightness, saturation et cetera.

The next step is to blur the images in order to smoothen it for the edge detection. Small inconsistencies caused by low quality cameras, textured

surface or noise in general may interfere with the finding of edges. Simple blurring is done by adapting the centre pixel to the average colour of the neighbour pixels but will be improved by a Gaussian blur in further versions.

Finding edges and boundaries is an important step for the purpose of feature extraction. Therefore a Sobel filter is chosen for its simplicity and robustness.

The last pre-processing step is the extraction of features. Therefore a region growing method is implemented that starts with finding non-edge pixels as starting points. Pixels are then added to regions according to their colour match. Once all pixels are part of a region, small regions are rejected, these objects are defined by their size, mean value colour, centroid, bounding box and region pixel map. These attributes are used by the stereo matching to find corresponding regions.

The stereo matching algorithm starts by superimposing two regions, one from each image, to calculate the union size which is the number of common pixels. The region with the strongest match and a certain percentage of pixel overlap will be considered as the corresponding region.

The whole matching algorithm works with uncalibrated cameras. Therefore an accurate depth calculation is impossible.

6.3 Implemented Area Based Algorithm

The last step is to implement the chosen area based stereo matching algorithm SAD from section 4.6. The existing framework is extended

6.4 Conclusion

References

[1] J. Baltes. http://www.cs.umanitoba.ca/~jacky.

[2] P. d'Angelo. http://hugin.sourceforge.net/. 2004.

[3] J.-Y. B. C. C. T. for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/index.html. 2004.

[4] J. Heikkilä and O. Silvén. A four-step camera calibration procedure with implicit image correction. *CVPR*, 1997.

[5] N. Kämpfchen. Modelbasierte lagebestimmung von objekten in stereobildsequenzen. *Diplomarbeit*, 2001.

[6] U. Köthe. http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/. *University of Hamburg, Germany*, 2004.

[7] M. Loper. http://loper.org/~matt/camchecker/camchecker_docs/html/index.html.

[8] B. McKinnon and J. Baltes. Practical region-based matching for stereo vision. In R. Klette and J. D. Zunic, editors, *IWCIA*, volume 3322 of *Lecture Notes in Computer Science*, pages 726–738. Springer, 2004.

[9] Science and engineering at the University of Edinburgh. http://homepages.inf.ed.ac.uk/rbf/cvonline/local_copies/owens/lect11/.

[10] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. *Third European Conference on Computer Vision, Stockholm, Sweden*, May 1994.

[11] Z. Zhang. A flexible new technique for camera calibration. *EEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.