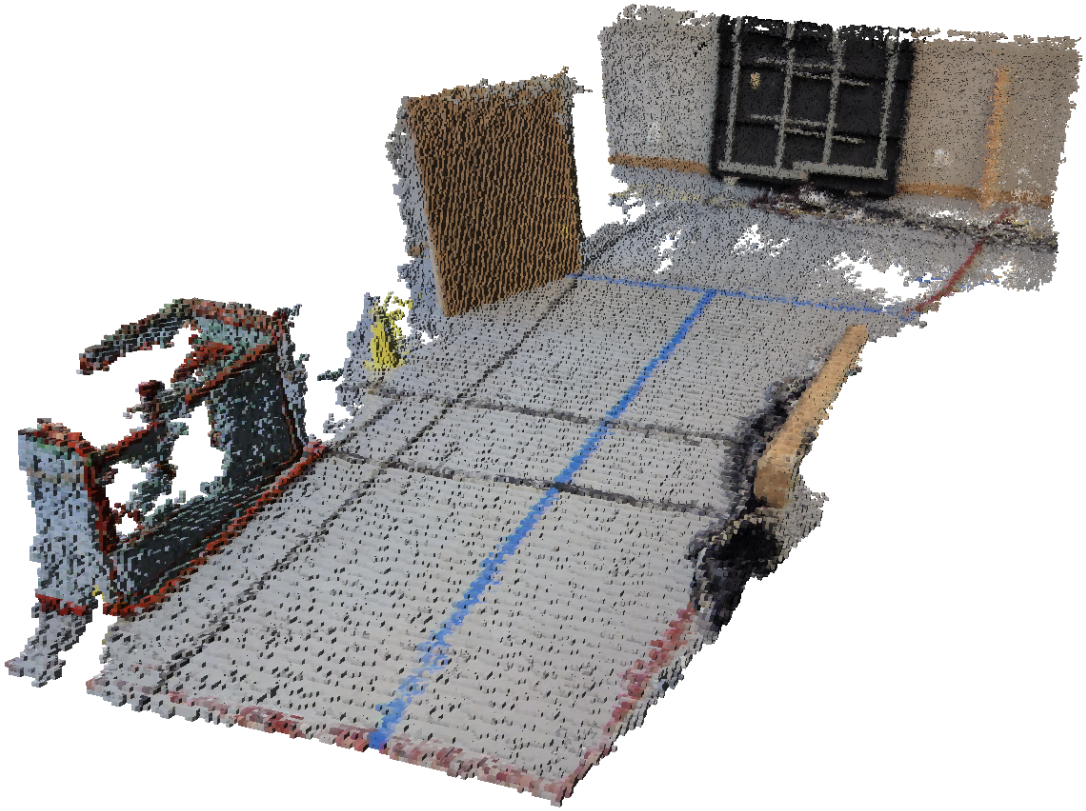
CHALMERS



Utilization of Quadnocular Stereo Vision for Simultaneous Localization and Mapping in Autonomous Vehicles

Master's thesis in Systems, Control and Mechatronics

ROBERT ANDERSSON
OSKAR NORESSON

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2015

MASTER'S THESIS IN SYSTEMS, CONTROL AND MECHATRONICS

Utilization of Quadnocular Stereo Vision for Simultaneous Localization and Mapping in Autonomous Vehicles

ROBERT ANDERSSON
OSKAR NORESSON

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2015

Utilization of Quadnocular Stereo Vision for Simultaneous Localization and Mapping in Autonomous Vehicles
ROBERT ANDERSSON
OSKAR NORESSON

Master's thesis 2015:06
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
A 3D voxel grid map representation of an indoor environment

Chalmers Reproservice
Göteborg, Sweden 2015

Utilization of Quadnocular Stereo Vision for Simultaneous Localization and Mapping in Autonomous Vehicles
Master's thesis in Systems, Control and Mechatronics
ROBERT ANDERSSON
OSKAR NORESSON
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

The introduction of autonomous vehicles in commercial applications has an enormous potential to drastically improve both working conditions and productivity. Among the many challenges of realizing autonomous driving, precisely determining the vehicle's pose and constructing an accurate map of an unknown environment might be the two most crucial to overcome. Simultaneously dealing with both these problems is known as SLAM, simultaneous localization and mapping.

We have designed and implemented a stereo camera system with which SLAM is performed using only visual information. The proposed system achieves multi-range depth resolution by utilizing four cameras and is implemented exclusively with the help of open-source libraries. Depth images from separate camera pairs are computed in real time with a correlation based block matching algorithm and merged employing a perspective transform. The joint depth map and the corresponding RGB image are then used by a graph-based SLAM algorithm to produce a trajectory estimate and a probabilistic 3D voxel grid map.

The system was furthermore evaluated on simulated and real data. The evaluations suggested that with the current configuration it is not motivated to merge depth images from more than two camera pairs, since adding an extra camera pair considerably decreases the frame rate without necessarily improving the quality of the resulting depth map. A number of different image feature extractors were investigated which revealed that binary feature descriptors are suitable for landmark representation in visual SLAM.

We can conclude that the system is indeed a fully working prototype, but in order to increase robustness and accuracy to a degree that it is useful in a commercial application a number of future improvements are needed.

Keywords: Autonomous vehicles, computer vision, SLAM, stereo vision, image feature extraction

Acknowledgements

This master's thesis is the concluding part of our studies within the M.Sc. programme Systems, Control and Mechatronics at Chalmers University of Technology. The thesis project was carried out during the spring of 2015 at CPAC Systems AB, a product development company mainly focusing on safety critical control systems in the automotive, construction and marine industries. CPAC Systems AB has provided all the necessary resources required to make this project possible, for which we are deeply grateful. We would also like to take this opportunity to thank a number of people, without whom this thesis could not have been achieved.

First and foremost, we would like to express our deepest gratitude to our supervisor Peter Forsberg for offering invaluable support and never failing to share his passion for technology. Our warmest thanks also goes to other employees and management at CPAC Systems AB for creating an inviting, enthusiastic and helpful atmosphere at the office. Furthermore, we would like to thank our examiner at the department of Applied Mechanics, Krister Wolff, for guiding us through the academical process and giving us feedback, ultimately making this thesis possible. Finally, we would like to show our appreciation to fellow students and thesis workers at CPAC Systems AB for practical assistance as well as making the last semester even more enjoyable.

Nomenclature

Acronym	Definition
BRIEF	**B**inary **R**obust **I**ndependent **E**lementary **F**eatures
BRISK	**B**inary **R**obust **I**nvariant **S**calable **K**eypoints
EKF	**E**xtended **K**alman **F**ilter
EMM	**E**nvironment **M**easurement **M**odel
FAST	**F**eatures from **A**ccelerated **S**egment **T**est
FPS	**F**rames **P**er **S**econd
FREAK	**F**ast **RE**tin**A** **K**eypoints
GPU	**G**raphics **P**rocessing **U**nit
IMU	**I**nertial **M**easurement **U**nit
IR	**I**nfra**R**ed
LIDAR	**LI**ght **D**etection **A**nd **R**anging
OpenCV	**Open** source **C**omputer **V**ision library
ORB	**O**riented FAST **R**otated **B**RIEF
PCL	**P**oint **C**loud **L**ibrary
RANSAC	**RAN**dom **SA**mple **C**onsensus
RGB-D	**R**ed **G**reen **B**lue **D**epth
RMS	**R**oot **M**ean **S**quare
ROI	**R**egion **O**f **I**nterest
ROS	**R**obot **O**perating **S**ystem
SGBM	**S**emi-**G**lobal **B**lock **M**atching
SGM	**S**emi-**G**lobal **M**atching
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SURF	**S**peeded **U**p **R**obust **F**eatures

Contents

1 Introduction

Although the concept of autonomous vehicles has been around for quite a while, it has recently become a very hot topic, gaining a lot of public attention as well as research grants. Early spectacular autonomous concept vehicles, including for instance the Google self-driving car [1] or the Mercedes-Benz F015 [8] have been smaller vehicles for personal transportation or surveillance. However, there are also potential upsides of replacing the driver in vehicles operating in heavy industrial and logistic applications, such as construction trucks and wheel loaders. Monotonous and hazardous driving tasks can be obliterated, costs for salaries and training can be minimized and utilization rates can be increased dramatically since robot drivers do not mind working around the clock.

An essential part in realizing autonomous driving is to find methods for acquiring accurate estimates of the vehicles' position and orientation as well as creating maps of the environment in which the vehicle operates. This type of problem is closely related to the *simultaneous localization and mapping* problem, also known under the acronym SLAM, which for the last decade has been a thoroughly studied topic within the field of autonomous robotics. The problem consists of estimating the robot's pose in a map representation of the surrounding world while at the same time incrementally updating the map with the information from current and past sensor readings [10]. The complexity of the SLAM-problem lies in its innate "chicken or egg"-characteristics; to update the map the robot needs to know its position and to know its position it needs to have a reliable map.

Localization and mapping for autonomous robots is traditionally based on input data from a laser scanner. This has the advantage of providing accurate distance measurements with high frequency, from which topological features can be extracted and used as landmarks for awareness of egomotion and position. However, high precision laser scanners tend to be extremely expensive. Furthermore, they traditionally only sweep in one or a few layers, resulting in a 2D map of the environment. If the laser scanner is to be mounted on a vehicle moving on a non-planar surface, which is the case for any terrain vehicle, the roll or pitch of the vehicle can not be inferred from a planar sensor reading.

Ideally, the sensor readings for a mapping application would be depth images which are not limited to one plane. This could be achieved by using an RGB-D sensor such as the Microsoft Kinect, which utilizes an infrared laser and an image sensor to project an IR-grid pattern and calculate depth through triangulation of the discrepancy between the projected pattern and the reflected pattern. In any outdoor application however, the possibilities of using this active sensor approach would be limited since sunlight would interfere with the IR-grid [25]. Although accurate and fast, it is also limited in its depth range [20]. A stereo camera on the other hand is a passive light sensor, thus more appropriate for outdoor use. Compared to laser scanners, it is a versatile and inexpensive sensor with many other potential application areas in autonomous vehicles, such as object detection or road sign classification. This master's thesis aims to develop and implement a system where this versatility is utilized, resulting in a proof-of-concept prototype that can contribute to make a commercial terrain vehicle autonomous.

1.1 Purpose & objective

The purpose of this thesis is to explore the possibilities of utilizing a quadnocular (i.e. using four cameras) stereo camera in a localization and mapping application so that autonomous navigation is possible. The intention is to create a stereo camera system capable of replacing or complementing an RGB-D camera or a LIDAR sensor, although no formal comparison will be conducted between the precision of our system and the accuracy of available sensor systems.

The main objective of this thesis is to create and implement a functioning system for extracting and analyzing information from quadnocular stereo vision data in real time in a moving vehicle. This includes developing a proper camera setup, computing depth from visual data, extracting and comparing appropriate image features, as well as implementing algorithms for localization and map building. The objective is furthermore to evaluate this system in order to find a suitable system configuration.

1.2 Scope

The goal of the project is not to optimize algorithms or to do in-depth research on a single particular algorithm. Instead, we aim at creating a functioning concept utilizing state of the art methods for the different subsystems

needed in the system. Thus, implementation and configuration of the methods will be a substantial part of the project work.

The final prototype created in this project should be possible to use in a real, all-terrain truck. This ambition poses different system requirements than if it was only mounted on a small, short-ranged and differentially steered robot which is moving on a planar surface. The broad scope of this thesis makes it absolutely necessary not to re-invent the wheel if the project is to be finished within the given time frame. Thus, available open-source software will be utilized where applicable. This includes the robotics framework ROS and available 3rd party packages therein, or libraries such as OpenCV and PCL.

An important limitation is that the system relies *only* on the data provided by the stereo cameras. In a future development of the prototype however, it would be useful to utilize sensor fusion techniques to integrate other sensors such as IMUs or wheel sensors with the visual odometry in order to make the SLAM procedure more robust and precise.

1.3 Thesis contribution

The novelty of this thesis lies in the system perspective and the integrated implementation of separate subsystems such as camera calibration, depth map creation, visual odometry and incremental map building. As the main objective of the project is to build a fully functioning prototype stereo vision SLAM system, including hardware design and full software implementation, it is not sufficient to solely evaluate our prototype on simulated data or pre-recorded benchmark data sets.

One of the most commonly used sensors in SLAM-applications are laser range scanners [30]. The use of stereo vision to solve the SLAM-problem is a rather novel approach. A traditional binocular stereo camera setup yields a single depth map, which is typically most accurate on a specific depth. We propose a quadnocular stereo camera setup in order to allow for pairwise combination of cameras with different baseline - shorter baseline for close range focus and depth perception, and wider baseline for higher accuracy on longer distances. This allows for creation of depth images with varying depth resolution that can be merged into a single depth map which is accurate on both short and long distances.

A central part of the SLAM problem is to recognize visual landmarks when they are reobserved. We have therefore emphasized this aspect by performing an extensive evaluation of image feature extraction algorithms, in a simulated environment as well as in a more applied context. In particular we investigate the potential of utilizing binary feature representations to speed up the feature matching in the SLAM application without compromising with the application's robustness.

1.4 Thesis outline

This report is divided into five parts. This introductory chapter is followed by a theory chapter which aims to give the reader an understanding of stereo vision, image feature extraction and map building. Next, we explain how the system was constructed and evaluated in a methodological chapter and then proceed to present and comment on the outcome of the evaluation. Finally we conclude the report with a short summary and propose possible topics for continuation and improvement.

2 Theoretical background

The stereo camera mapping and localization system developed in this thesis relies on four concepts which are given a fundamental theoretical background in the following chapter. First, we introduce stereo vision and describe how depth can be reconstructed from visual data alone. Next, we present algorithms for image feature extraction which is required for visual SLAM as well as utilized in the merging of depth maps from different camera pairs. The third concept we introduce is SLAM itself, including the problem formulation and the most common solution approaches. Finally, we present methods for representing 3D maps in a robotics application. The aim of all four sections is to first briefly describe and motivate the overall concept, and then to give the reader a more focused orientation of the algorithms that were actually implemented.

2.1 Stereo vision

The concept of stereo vision is based on the fact that perception of depth is possible by observing a 3D world coordinate point from different viewpoints. The relative displacement of the image plane projection of the point on the respective camera image planes, the so-called disparity, can be used to triangulate the depth if geometric information about the cameras is available. The geometric relationships used to reconstruct depth from disparity is explained further in section 2.1.5.

The main principle behind stereo vision is illustrated in figure 2.1, where the disparity is the difference between the horizontal image projections u and u' on the left and right cameras [33]. However, in order to actually compute the disparity, it is necessary to first have an understanding of the camera model and how the images are prepared for stereo matching.

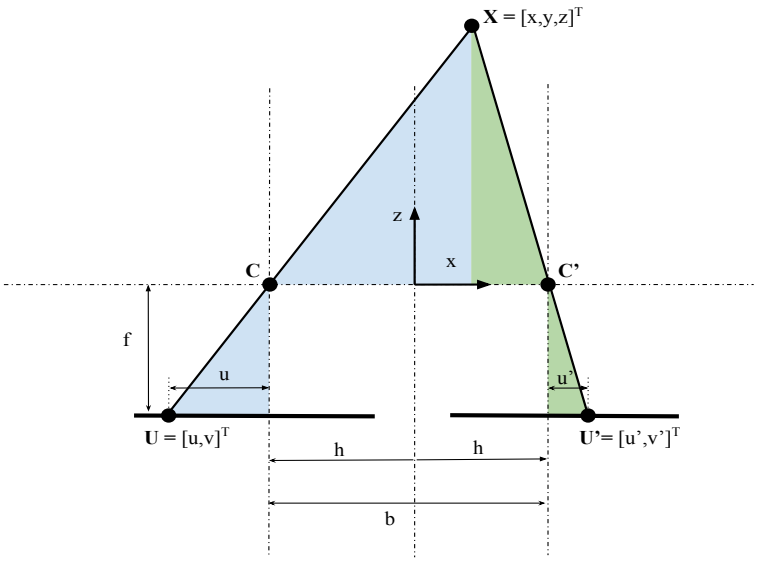


Figure 2.1: *The fundamental geometry of a rectified (see section 2.1.3) stereo camera system seen from above. A point* $\mathbf{X}$ *will be projected to different horizontal image coordinates* u *and* u' *from which the depth* z *can be inferred. As described in section 2.1.5, two pairs of similar triangles (green and blue) can be utilized to form a relation between the disparity and the depth.*

2.1.1 Camera model

The most simple camera model is the pinhole camera, where an object is projected onto an image plane by only allowing a single ray of light to enter the camera from any single point. To ensure this, light is only allowed to travel though a microscopic pinhole. However, such a model makes the exposure process slow and does not work in practice. Introducing lenses considerably accelerates the exposure process, but at the price of also introducing lens distortions. Manufacturing a camera that introduces no image distortions is in practice

unfeasible, especially at a reasonably modest price tag.

Two of the most prominent lens distortions are radial and tangential distortions. The former occur when the lens shape is non-perfect, typically when the lens is stronger than necessary towards the periphery causing noticeable image distortions towards the edges and corners of the resulting image. The latter occur when the lens is not completely parallel to the image sensor, typically due to poor precision during manufacturing, causing elliptical displacement of the image.

The camera intrinsics matrix

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

contains the focal lengths of the camera (f_x and f_y) and the displacement between the center of the image sensor and the optical axis of the lens (c_x and c_y). A point P with real world coordinates (X, Y, Z) can be reprojected into a point p on the image sensor with homogeneous coordinates (x, y, w) using the the relationship

$$p = MP. \tag{2.1}$$

Solving equation 2.1 will yield that $w = Z$. Since p is written in homogeneous coordinates, dividing equation 2.1 by w (or Z) will give the actual pixel coordinates in terms of real world coordinates.

For describing the distortions introduced by the camera lens, the "plumb bob" model can be used consisting of two pairs of equations:

$$\begin{cases} x_{undistorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{undistorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{cases} \tag{2.2}$$

$$\begin{cases} x_{undistorted} = x + [2p_1 y + p_2(r^2 + 2x^2)] \\ y_{undistorted} = y + [2p_2 x + p_1(r^2 + 2y^2)] \end{cases} \tag{2.3}$$

where (k_1, k_2 and k_3) describes the radial distortion, (p_1 and p_2) describes the tangential distortion and r is the radial distance from the optical center of the camera. The camera intrinsics matrix together with the distortion coefficients, commonly arranged as $[k_1\ k_2\ p_1\ p_2\ k_3]$, make up the intrinsic camera parameters [5].

2.1.2 Camera calibration

The cameras need to be calibrated in order to estimate and compensate for the intrinsic camera parameters, the transformations from world to camera coordinates (the so called extrinsic parameters), as well as for angular and translational differences between camera pairs. This can be done in a two stage process where first single camera calibration and then stereo calibration is performed. During single camera calibration, the intrinsic and extrinsic parameters are estimated individually for each camera. During stereo calibration, the transformation (rotation and translation) between the two camera planes is estimated.

The calibration process is normally performed using a calibration pattern, an object with known dimensions and easily detectable features. A typical calibration pattern is a checkerboard with alternating black and white squares of known size. In order to end up with a successful calibration, it is important to both translate and rotate the calibration pattern in between the calibration views.

Single camera calibration can be performed by observing a set of images each containing different views of the checkerboard. On this set of images, a corner detection algorithm can be applied to find the image locations of all interior corners of the checkerboard pattern. Information about the geometry of the checkerboard and the image coordinates of the checkerboard corners can then be used as input for an optimization problem, which when solved yields the intrinsic parameters (as described in section 2.1.1) as well as the extrinsic parameters for the camera as a 3×3 rotation matrix and a 3×1 translation vector.

Stereo calibration can be done by performing single camera calibration for two different cameras both observing the same static calibration pattern. The goal is to calculate the rotation matrix R and the translation vector T describing the transformation from the right camera frame to the left camera frame. A point P (in world coordinates) on the calibration pattern can be expressed in the left camera coordinate frame as

$$P_{Left} = R_{Left} P + T_{Left} \tag{2.4}$$

and similarly in the right camera coordinate frame as

$$P_{Right} = R_{Right} P + T_{Right}. \tag{2.5}$$

Combining the fact that the two cameras are related as

$$P_{Right} = RP_{Left} + T \tag{2.6}$$

with equation 2.4 and 2.5 gives the rotation matrix

$$R = R_{Right}R_{Left}^{T} \tag{2.7}$$

and translation vector [5]

$$T = T_{Right} - RT_{Left}. \tag{2.8}$$

2.1.3 Undistortion and rectification

To be able to construct a high quality depth map, the images taken by the cameras need to be preprocessed. First a distortion map should be created from the distortion coefficients in the intrinsic camera parameters which describes how the lenses have distorted the images. This distortion map can then be used to undistort the images. After undistortion, the images look as if captured with a pinhole camera, given that the calibration was successful. The images must also be rectified, by utilizing the stereo parameters described in section 2.1.2. Rectification means that the image planes are aligned such that corresponding points lie in the same row on both images and that the cameras are horizontally aligned. The points are thus located on the same so-called *epipolar lines*. Since the points are located on the same line, the *epipolar constraint* can be formulated, which reduces the otherwise two dimensional stereo correspondence problem (see section 2.1.4) to a one dimensional search problem. Simply put, it is only required to look for the corresponding pixel or block of pixels in one row instead of in the whole image when comparing the left and right images [33].

2.1.4 Stereo correspondence

The next step in estimating depth from a stereo image pair is to calculate the disparity by performing stereo correspondence. The problem is described in figure 2.2. For a given point in 3D space, the stereo correspondence problem consists of finding the corresponding image locations of the point's projections in two rectified images which depict the point from two different viewpoints. The difference between the horizontal coordinates is the disparity d. The output from the stereo correspondence process is a disparity map $d(x, y)$, which is an image where the intensity of a pixel with coordinates (x, y) is the disparity d.

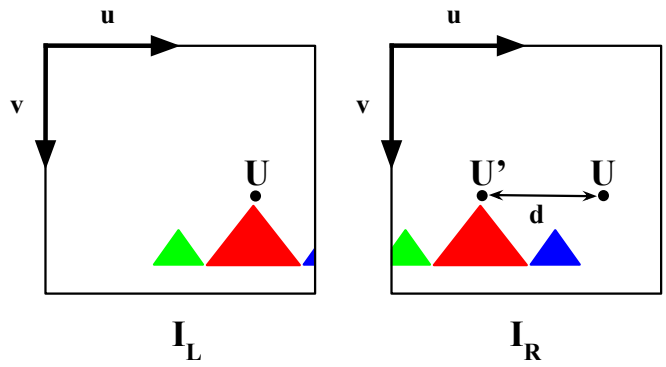


Figure 2.2: *The objective of a stereo correspondence algorithm is to find the horizontal distance d between the corresponding points* **U** *and* **U**′ *in two rectified images.*

Stereo matching is a very active research area and algorithms exist in a wide variety. It could be done for instance by extracting feature keypoints (such as those described in section 2.2) and matching keypoints from the two images. Such an approach is, although reliable and fast in finding the correct pixel matches, not sufficient for this kind of application. The reason is that 3D map building relies on dense disparity images, whereas keypoint matching methods only generate sparse disparity images where disparity is calculated only for the actual keypoints.

A more suitable approach is thus a correlation based method, where a region in the first image is compared with a region of the same size in the image it is to be matched with. The region is then horizontally shifted in the matching image. The location where the regions have a correlation peak gives the disparity [6].

One popular correlation based stereo correspondence method is the semi-global matching algorithm (SGM). The objective is to iteratively minimize a cost function based on comparing pixel intensities while traversing epipolar lines in the matching images. The larger the correspondence of the intensities, the smaller the cost. In theory, the cost is minimal when the matched pixels correspond to the same world point. In practice however, due to e.g. noise, this will often not be the case and finding a distinctive minimal cost can be very difficult. The SGM cost function therefore also penalizes change in disparity between neighboring pixels. This penalty term minimizes the number of false discontinuities in the disparity and thereby increases the smoothness of the final disparity map [16].

2.1.5 Reprojection

Given a rectified image pair, depth can be estimated from the disparity image by taking advantage of the geometric relations illustrated in figure 2.1. In the figure, two pairs of similar and right-angled triangles can be found with hypotenuses $\mathbf{UC}$, $\mathbf{CX}$ and $\mathbf{U'C'}$, $\mathbf{C'X}$ respectively. The catheti of the similar triangles are related as

$$\frac{u}{f} = -\frac{h+x}{z}, \qquad \frac{u'}{f} = \frac{h-x}{z} \tag{2.9}$$

which when eliminating x and solving for z yields

$$z = \frac{2hf}{u'-u} = \frac{bf}{u'-u} = \frac{bf}{d}. \tag{2.10}$$

The depth z is thus directly proportional to the baseline b and the focal length f, and inversely proportional to the disparity d [33]. In a stereo camera system, there are thus two central design parameters; the baseline and the focal length. Depending on how these design parameters are selected, the range in which objects can be detected will be altered. A shorter baseline or focal length will decrease the disparity for a given depth and hence depth resolution will be poorer on longer distances. A longer baseline or focal length will analogously increase the depth resolution far away from the camera. However, if the baseline is larger, the overlapping area of the two images will be reduced, limiting the field of vision on shorter distances. The stereo correspondence search problem will furthermore have a larger search space since corresponding pixels are located further apart. One way to balance this trade-off between optimal depth perception performance on short and long distances would be to use a multi-baseline camera system [19].

2.2 Feature extraction

As localization and map-building rely on matching landmarks seen in separate time instances, a crucial step in the visual SLAM process is to detect, extract and match reliable image features which represent these landmarks. Features are prominent keypoints in the image, which are assigned descriptors that characterize some properties of the image patch surrounding the keypoint to enable comparison and recognition. By detecting features in two different frames and computing the distance between the keypoints in descriptor space, it can be determined whether the features represent the same real world landmark or not.

2.2.1 Vector-based features

The problem of detecting, describing and matching image features is a well-studied topic in the field of computer vision. Early attempts for feature detection include the combined edge and corner detection method proposed by Harris and Stephens [15]. Their method is based on the idea that when a window function is shifted in all directions over the image, the change in intensity can be used to detect corners and edges. When the window is shifted over a flat region without corners or edges, no intensity change is seen. However, when the window is shifted over a corner, the intensity will change in all shifting directions. When shifted over an edge, large intensity changes will be observed only when the shifting direction is perpendicular to the edge. This idea is illustrated in figure 2.3.

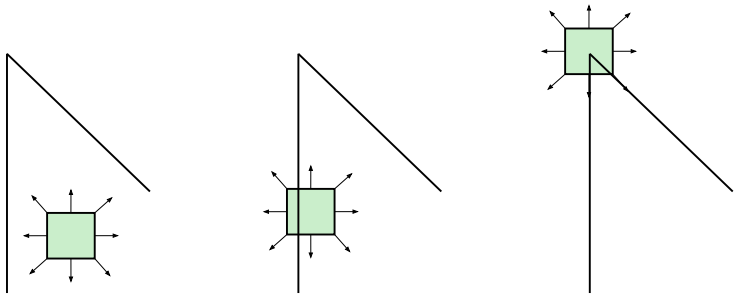


Figure 2.3: *Shifting a window function over a flat region will not lead to any significant intensity changes in any direction (left panel), whereas shifting over an edge will do so when the shifting direction is perpendicular to the edge (middle panel). If a corner is present in the window, all directions will give intensity changes (right panel).*

A major issue with these so called Harris corners are that they are highly sensitive to changes in image scale [32]. As a moving vehicle will observe the same landmark from different distances, a robust feature extractor must take this into account and detect similar features regardless of the scale of the image in which they are observed. Other desired properties of a feature detector and descriptor includes robustness against variation in viewpoint, rotation and illumination, as all these are subject to change due to the mobile nature of the application. If high robustness for those primary distortions is achieved, it is a reasonable assumption that also robustness against various perspective changes and skew distortions will be an implicit consequence [4].

Furthermore, it is necessary to keep the computation runtime to a minimum to enable an online, real-time execution of the SLAM application. We therefore present a set of more recent feature extraction algorithms that to some extent balances this trade-off between robustness and computational complexity.

SIFT

Proposed by Lowe in 1999, the *Scale Invariant Feature Transform* (SIFT) is often referred to as the quality benchmark for feature extraction algorithms due to its very high robustness against changes in scale, rotation and illumination. In an extensive performance evaluation by Mikolajczyk and Schmid, SIFT or SIFT-based descriptors were found to be of outstanding superiority to other descriptors for correctly matching features in a textured scene [27]. However, this comes at the price of being considerably slower than other feature extractors.

Keypoint detection in SIFT is based on local extrema detection in the so-called Difference of Gaussian-pyramid (DoG). To construct the pyramid, the image I is first transformed into scale space by convoluting it with a Gaussian kernel G of scale σ giving the image

$$L(x, y, \sigma) = I(x, y) * G(x, y, \sigma) \tag{2.11}$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.12}$$

Each level D in the DoG-pyramid is then computed by varying the scale σ with a factor k and finding the difference between the scale space images giving the DoG-image

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma). \tag{2.13}$$

The DoG-pyramid is searched through in a bruteforce manner, where any pixel which is an extreme point in a 3-by-3-by-3 neighbourhood (as seen in figure 2.4) is considered a potential keypoint. The keypoints then undergo a screening process in which two categories of keypoints are pruned:

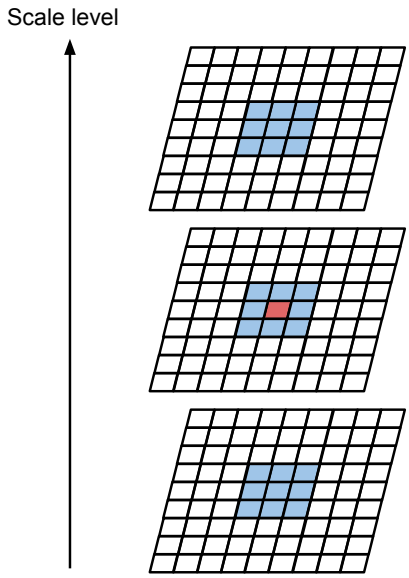


Figure 2.4: *Three different levels in the DoG-pyramid illustrating the local extrema search. If the intensity of the central pixel (red) is higher or lower than all its neighbours' intensity (blue) it is considered a keypoint candidate.*

1. Keypoints that have a low contrast are sensitive to noise and are therefore removed by deleting all keypoints with DoG-values below a certain threshold.

2. Keypoints that are aligned along edges (and not at corners) will yield a strong response from the DoG-function but are difficult to localize correctly, especially under noisy conditions. They can be removed by applying a similar approach as in the Harris detector as described above.

The keypoints are then assigned an orientation by finding the highest peak in a 36-bin histogram (each bin corresponding to a 10-degree interval) in which the orientation of the gradients in a neighbourhood around the keypoint are added. If the orientation of a keypoint is known, it can be used to rotate the keypoint and its neighbourhood back to a normalized orientation before its descriptor is calculated. By doing so, all keypoints are described in the same global orientation which is a how rotation invariance is achieved. Finally, the descriptors are calculated from another histogram of oriented gradients, where 16 histograms with 8 bins each result in a 128-dimensional descriptor vector for each keypoint [24].

SURF

A lot of effort has been made to achieve feature extraction with similar robustness as SIFT, but with a reduced computational complexity to enable use in real-time or embedded applications. One such feature extractor is SURF, *Speeded Up Robust Features*, which was proposed by Bay et al. in 2006 [3]. SURF builds on the same principles as SIFT, but scale space images are instead constructed from integral images. The integral image $I_{\sum}$, computed from the original image I as

$$I_{\sum}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i,j) \tag{2.14}$$

has the advantage that once computed, it can be used to do the convolution with the Gaussian kernel using only a few additions or subtractions regardless of the kernel size. The different scales can then be computed efficiently by varying the size of the kernel, starting at 9×9. A key to SURF's efficiency is furthermore that the different scales can be computed in parallel [4].

Orientation of keypoints is computed from the sum of the so-called Haar wavelet responses in the x- and y-direction. Wavelet transforms are functions that model both temporal (or more correctly spatial as it is an image and not a function of time) and frequency information [33]. As the orientation computation is

a time-consuming process, there is also an upright version of SURF (U-SURF) where this step is skipped. U-SURF can thus be utilized in applications that do not rely on rotation invariance to achieve higher extraction speeds.

The descriptors are also based on wavelet responses. A square patch with normalized orientation around the keypoint is divided into 4×4 subregions. For each subregion, the horizontal (d_x) and vertical (d_y) wavelet responses are calculated on a 5×5 sampling grid. Each of the 16 subregions are then described by a four-dimensional vector, consisting of $\sum d_x$, $\sum d_y$, $\sum |d_x|$ and $\sum |d_y|$. The SURF descriptor is thus a 64-dimensional floating point number vector [4].

2.2.2 Binary features

Floating point descriptor vectors such as those generated by SIFT or SURF contain abundant information, but are not ideal from an implementation perspective, especially in a real-time application. They quickly occupy a lot of memory resources and require time consuming processor operations both for extraction and matching. In the matching process, a metric of similarity is needed to compare features from the training set and query set respectively and decide whether they are similar enough to be considered representing the same landmark. Traditionally, the Euclidean distance is used, formulated as

$$Euc(x, \hat{x}) = \sqrt{\sum_{i=1}^{n} (f_i(x) - f_i(\hat{x}))^2} \tag{2.15}$$

for the distance between features x and $\hat{x}$ with descriptors $f(x) \in \mathbb{R}^n$ and $f(\hat{x}) \in \mathbb{R}^n$ of length n. This calculation requires at least n multiplications of floating point numbers, a simple but nonetheless relatively slow task for a processor. Descriptor extraction and matching can be sped up significantly if they are instead formulated as binary descriptors. A suitable metric of similarity is then the so-called Hamming distance. The Hamming distance between features x and $\hat{x}$ with the respective binary descriptors $b(x) \in \{0, 1\}^n$ and $b(\hat{x}) \in \{0, 1\}^n$ of length n can be computed as

$$Ham(x, \hat{x}) = \sum_{i=1}^{n} b_i(x) \oplus b_i(\hat{x}) \tag{2.16}$$

where $\oplus$ is the exclusive or-operator [12]. Each of these operations can be performed in one or a few clock cycles, making the binary feature extraction and matching procedure several magnitudes faster.

Because of memory limitations, a visual SLAM application running for more than a very brief period of time requires feature descriptors that are compact to store. The 128-dimensional floating point vectors generated by SIFT occupy 512 bytes, which is far from optimal when the number of descriptors increase by thousands every second. This problem could be addressed by for instance reducing the dimensionality of the vector or to use integer representation of the floating point numbers. However, reduction of dimensionality tends to degrade performance and neither approach permits the use of the Hamming distance as a similarity metric. Furthermore, it is not very efficient to first perform a series of time-consuming computational steps to produce a long vector, and then spend even more time making it shorter. Instead, the binary feature extractors compute the descriptors as binary strings directly. This yields for instance a 256-bit binary representation, i.e. a 16 times more compact descriptor than SIFT [7]. For this thesis, three such binary features are of special interest: *ORB*, *BRISK* and *FREAK*.

ORB

ORB, which is an acronym for *Oriented FAST and rotated BRIEF*, is a binary feature which was proposed in 2011 by Rublee et al. As the name suggests, it builds on FAST for keypoint detection. FAST is a highly efficient corner detector, but unlike SIFT and SURF it does not have an orientation component. Calculating the orientation of a keypoint however, normally requires long computational times, especially when using the histogram of oriented gradients-approach as is done in SIFT. In ORB, the orientation is instead based on the orientation of a vector between the corner's center and its intensity centroid. The intensity centroid of the keypoint neighbourhood, analogous to a physical object's center of mass, can be calculated as

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.17}$$

where

$$m_{ij} = \sum_x \sum_y x^i y^j I(x, y). \tag{2.18}$$

Finding the orientation of the vector from the corner center to the intensity centroid is then only a matter of computing the inverse tangent. This intensity centroid approach has proven to be drastically more robust to image noise than histogram-based approaches [31].

The descriptor calculation part of ORB is based on BRIEF, which uses a binary intensity test that is performed 128, 256 or 512 times around a keypoint [7]. The test τ on the keypoint neighbourhood patch $\mathbf{p}$ is defined as

$$\tau(\mathbf{p}, \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases} \tag{2.19}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two pixel locations in the patch and $\mathbf{p}(\mathbf{x})$ is the intensity of the pixel at location $\mathbf{x}$. The test is performed n times on a smoothed image with location of points $\mathbf{x_i}$ and $\mathbf{y_i}$ selected from a Gaussian distribution giving the binary string b of length n

$$b_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}, \mathbf{x_i}, \mathbf{y_i}). \tag{2.20}$$

Since the orientation of the keypoint is known from the detection step, the image patch on which the tests are performed can be rotated to the global neutral orientation, resulting in a descriptor known as *steered BRIEF* (sBRIEF). However, this procedure makes the appearance of the keypoint neighbourhoods more uniform, so that the variance (and thus the discriminative property) of the descriptors is reduced. This problem is solved in ORB by instead utilizing *rotated BRIEF* (rBRIEF), which includes a learning method for selecting a subset of binary tests to maximize the variance. The final result is a feature detector and descriptor which is both relatively fast (typically 10 times faster than SURF and 100 times faster than SIFT), rotation invariant and discriminating [31].

BRISK

The *Binary Robust Invariant Scalable Keypoints*, abbreviated BRISK, is another attempt to merge the high scale and rotation invariance of the SIFT and SURF descriptors with the low computational time of the binary features. Just like the ORB detector, the keypoint detection stage in BRISK is based on FAST with a computed orientation. However, as in SIFT, maxima are searched for in the scale space and not only in the image plane which is a key procedure to achieve scale invariance. By calculating a metric of scale for each feature as well as an orientation, they can be scale normalized in the same way as rotation is set to a neutral level before descriptor extraction.

Descriptor computation is done by sampling and smoothing points in a circular pattern, concentric with the keypoint location, as illustrated in figure 2.5. The sampling points are smoothed with different standard deviation σ depending on their distance to the keypoint in order to avoid aliasing effects. A set of sampling point pairs from the sampling pattern is formed by pairwise combination of the points. Each point may then contribute to several pairs.

A subset of pairs with longer point to point distance is used for determining the orientation of the keypoint and a subset with 512 pairs of rotated points with shorter distance is used for building the descriptor. The descriptor bit string is formed by using an intensity test on the sampling point pairs, in a similar fashion as is done when calculating ORB descriptors [22].

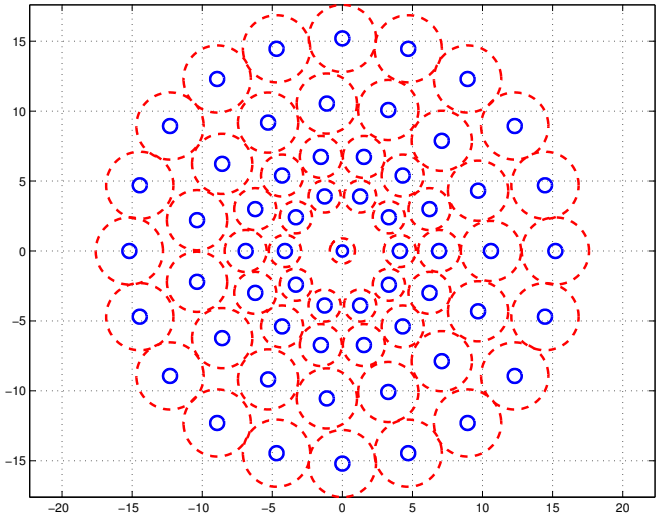


Figure 2.5: *The circular sampling pattern for BRISK descriptor computation. Blue circles mark the location of the sampling points and the radius of the red circles correspond to one standard deviation in the Gaussian smoothing around that sampling point. The keypoint itself is located at the origin.*

FREAK

FREAK, short for *Fast REtinA Keypoints*, is only a method for feature description and does not include a detection step. It is biologically inspired by the human retina, hence the name. The sampling pattern, which in BRISK was a number of concentric circles, is in FREAK also circular but analogous to the ganglion cells of the retina the sampling points are more concentrated closer to the center. Furthermore are the Gaussian kernels with which the receptive fields around sampling points are smoothed different from those used in BRISK. Also, the receptive fields are overlapping. Similar to in ORB, the pairs for bit testing are selected according to a learned pattern in order to maximize variance. The binary string is then built in a cascade approach with bit testing on the pairs mainly in the peripheral circles first. The more central pairs are thus compared last. This enables a sped up, coarse-to-fine matching procedure where bits are compared in groups of 128. If a match is confirmed using a coarse bit sequence, there is no need to continue and compare bits from the central sampling points [2].

2.3 SLAM

Simultaneous Localization And Mapping, SLAM, is the process of incrementally building and updating a map of an unknown environment while at the same time using the incomplete map for localization within the map. SLAM might initially appear to be a "chicken or egg"-problem since an accurate map is needed for accurately estimating the pose, and an accurate pose estimate is needed for accurately building the map [10]. It is however possible to find a solution to the SLAM problem [9].

2.3.1 Problem formulation

The SLAM problem deals with simultaneously estimating an agent's (such as a robot or a vehicle) location and the location of landmarks in the vicinity of the agent, without knowing the true respective locations. This is done by taking sensor measurements of the true distance between the agent and the landmarks, as is shown in figure 2.6. The error between true and estimated landmark positions is primarily due to agent pose uncertainty at the time of landmark observation, implying that said error is highly correlated between landmarks. The relative location between landmarks can therefore be accurately determined without accurate estimates of the absolute landmark locations [10].

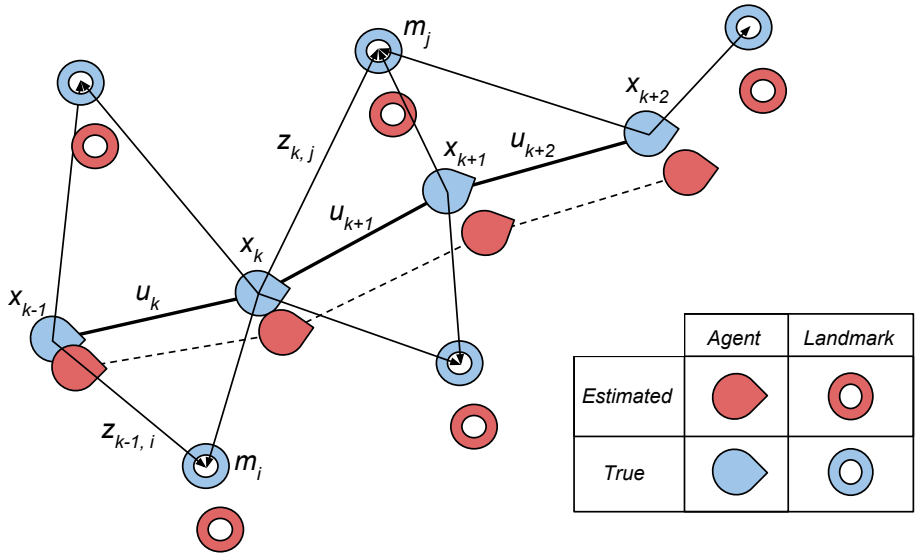


Figure 2.6: *The SLAM problem deals with simultaneously estimating the agent and landmark locations using sensor measurements between the true location of the agent and surrounding landmarks.*

In order to describe how SLAM is solved, the following notation is needed:

- A state vector, $\mathbf{x_k}$, describing the agent's pose.
- A control vector, $\mathbf{u_k}$, describing an estimation of the agent's motion between time $k-1$ and k, derived from e.g. the control input or wheel odometry.
- A landmark location vector, $\mathbf{m_i}$, for landmark i.
- A landmark observation vector, $\mathbf{z_k}$, describing the location relative the agent of the observable landmarks at time k.
- The set of agent poses, the trajectory $\mathbf{X_{0:k}} = \{\mathbf{x_0}, \mathbf{x_1}, ..., \mathbf{x_k}\}$.
- The set of control signals, $\mathbf{U_{0:k}} = \{\mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u_k}\}$.
- The set of landmark locations, $\mathbf{m} = \{\mathbf{m_1}, \mathbf{m_2}, ..., \mathbf{m_n}\}$, constituting the map.
- The set of landmark observations, $\mathbf{Z_{0:k}} = \{\mathbf{z_1}, \mathbf{z_2}, ..., \mathbf{z_k}\}$.

In order to solve the SLAM problem in a probabilistic way, the joint posterior density of agent and landmark location needs to be calculated in each time step, using the initial pose, $\mathbf{x_0}$, the set of control signals $\mathbf{U_{0:k}}$, and the set of landmark observations $\mathbf{Z_{0:k}}$

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}). \tag{2.21}$$

The probability of making a landmark observation, $\mathbf{z_k}$, given a known agent and landmark location is known as the *observation model* and is described by

$$P(\mathbf{z_k} | \mathbf{x_k}, \mathbf{m}). \tag{2.22}$$

The vehicle *motion model* describes how the next state, $\mathbf{x_k}$, depends only on the preceding state $\mathbf{x_{k-1}}$ and the control signal $\mathbf{u_k}$

$$P(\mathbf{x_k} | \mathbf{x_{k-1}}, \mathbf{u_k}), \tag{2.23}$$

here assumed to be a Markov process.

Equation 2.21 can then be solved in a recursive fashion by alternating between a prediction (time-update) step

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}}, \mathbf{x_0}) = \int P(\mathbf{x_k} | \mathbf{x_{k-1}}, \mathbf{u_k}) \times P(\mathbf{x_{k-1}}, \mathbf{m} | \mathbf{Z_{0:k-1}}, \mathbf{U_{0:k-1}}, \mathbf{x_0}) \, dx \tag{2.24}$$

and a correction (measurement-update) step

$$P(\mathbf{x_k}, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}) = \frac{P(\mathbf{z_k}|\mathbf{x_k}, \mathbf{m})P(\mathbf{x_k}, \mathbf{m}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}}, \mathbf{x_0})}{P(\mathbf{z_k}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}})}. \tag{2.25}$$

2.3.2 Solution approaches

Different approaches to solving the SLAM problem involves different ways of representing the observation and motion model, leading to different ways of calculating equation 2.24 and 2.25 [10].

There are primarily three main approaches which most SLAM solutions are based on:

- Extended Kalman filter (EKF) based methods
- Particle filter based methods
- Graph-based methods

In EKF SLAM, a single state vector and covariance matrix is used to represent the estimated agent and landmark locations and the corresponding associated errors respectively. An extended Kalman filter is used when updating the state vector and covariance matrix. EKF SLAM is computationally limited since the size of the covariance matrix grows quadratically with the number of observed landmarks [35], clearly making it an infeasible approach for performing visual SLAM where landmarks are constituted of a large number of image features.

The particle filter based methods use particles for estimating the agent and landmark locations. A direct implementation of particle filters for solving the SLAM problem is not computationally feasible due to the high dimensionality of the state-space [10]. By instead using a Rao-Blackwellized particle filter, which is used in e.g. FastSLAM [28], it is possible to solve the SLAM problem. FastSLAM relies on the fact that it is possible to factorize the posterior according to:

$$P(\mathbf{X_{0:k}}, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}) = P(\mathbf{m}|\mathbf{X_{0:k}}, \mathbf{Z_{0:k}})P(\mathbf{X_{0:k}}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}) \tag{2.26}$$

where the the probability distribution is conditioned on the entire trajectory $\mathbf{X_{0:k}}$ rather than the pose $\mathbf{x_k}$ as in equation 2.21. Each particle is associated with an individual map of the landmark locations. The landmarks locations are updated assuming perfect agent pose estimates. Each particle thus corresponds to a different vehicle trajectory hypothesis. By utilizing many particles, a probabilistic model of the agent location can be achieved [10].

The graph-based methods can be thought of as nodes connected in a graph, where each node correspond to agent or landmark locations. Arcs in the graph represent constraints between different agent poses, e.g. from odometric readings, or between agent poses and landmark locations, from sensor readings. The graph generated in graph-based SLAM is sparse meaning that relatively few connections exist between nodes. At worst, the number of constraints between nodes is linear in elapsed time and the total number of nodes. The sparse nature of the graph-based methods enables efficient solving of the SLAM problem using sparse nonlinear optimization techniques. Compared to EKF SLAM, longer mapping sequences are possible since the memory requirement of graph-based SLAM is a linear function of time [35].

Graph SLAM is usually divided into two parts, the front-end and the back-end. The former deals with finding the constraints mentioned above in order to construct the graph. The front-end is thus highly dependent on the type of sensors used. The back-end on the other hand optimizes the constructed graph and is sensor independent [14].

2.3.3 RGB-D SLAM

One of the graph-based methods for solving the SLAM problem is RGB-D SLAM. It is a six degrees of freedom SLAM system designed for use with RGB-D sensors such as Microsoft Kinect or ASUS Xtion Pro Live. A schematic overview of the RGB-D SLAM system architecture is given in figure 2.7 and as other graph-based SLAM methods it consists of a front-end and a back-end.

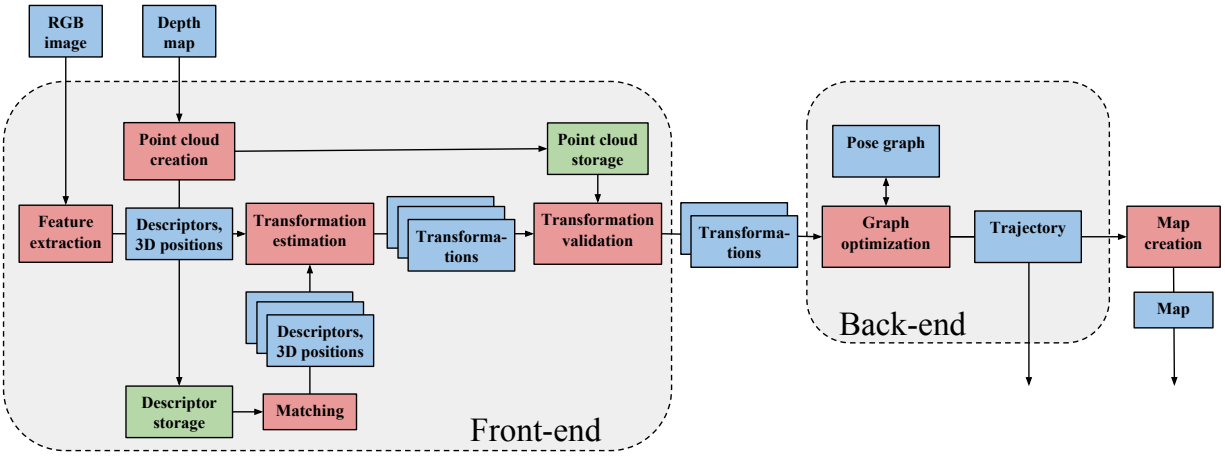


Figure 2.7: *The system architecture of RGB-D SLAM [11].*

The RGB-D SLAM front-end takes RGB images and depth maps as input. In each time step, image features are extracted from the RGB image and the detected keypoints are assigned a descriptor vector as well as a 3D position from the depth map data. The features and their 3D position data are stored in a database from which the features from the current image frame are matched against previous frames. Employing the least squares method, the found correspondences are used to compute an estimate of the transform, which describes the egomotion of the camera system compared to a previous frame [11]. RGB-D SLAM utilizes the RANSAC method [13] to protect against outlier feature correspondences, which otherwise would have distorted the transform.

By performing egomotion estimation sequentially over a series of frames, a process known as visual odometry, a set of transforms is generated that describes the movement of the camera. The transforms are furthermore verified using an environment measurement model (EMM), penalizing poses corresponding to improbable sensor measurements. The EMM used is beam-based, meaning that motion transformations corresponding to depth measurements that should have been occluded by other depth measurements are penalized.

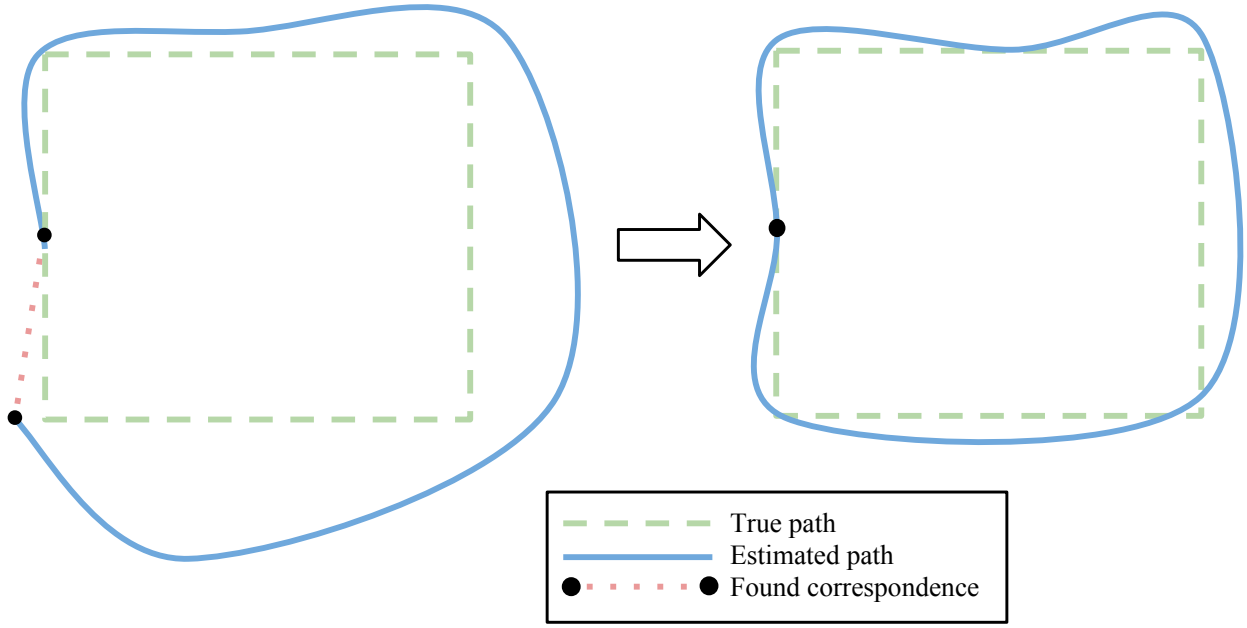


Figure 2.8: *Illustration of how loop closure reduces accumulated errors when correspondence is found to a previous node.*

When searching in the database for correspondences to the current frame's features, the best matches do not necessarily have to generate a transform to the most recently added nodes. If for instance the camera has moved around the block in a city and revisits a location along the traveled path, the best match is likely found in a much earlier node. This information can be used to perform loop closure, i.e. connecting the frontier node to a much previous node and thereby correcting the trajectory between the nodes that are connected. Loop closure is very efficient for correcting accumulated drifting errors in the visual odometry procedure, as is illustrated in figure 2.8.

The set of transformations exits the front-end as pairwise connections of poses, constituting the constraint arcs in the pose graph. Because of measurement noise, it is not a trivial task to create a consistent trajectory. To form this trajectory, a global optimization of the graph is required. As is explained in section 2.3.2, this is the task of the back-end [11]. The graph optimization is performed using the g^2o framework [21], which produces an optimal trajectory by minimizing an error function consisting of the constraints and their uncertainties. The output is thus an optimized pose graph, which can be used to create a global map by projecting the stored point clouds to the corresponding positions in the global map coordinate frame [11].

2.4 3D map representations

The map representation is of great importance in a successful map building application. It is necessary to represent the map in a compact yet descriptive manner, so that longer mapping sessions do not yield unmanageable file sizes. For navigation purposes, maps need to be efficiently searchable by allowing fast access and queries, requiring an intelligent data structure.

The most direct way of representing 3D maps is probably a point cloud. Every measured point can simply be described by its x-, y- and z-coordinate in 3D space, and there is thus no need for data discretization. Consequently, the memory demands grows linearly with each sampled point, making it a highly memory consuming map representation, especially for depth sensors with high frequency or spatial resolution. Furthermore, since points have no extension in space, there is no direct method for distinguishing between free and occupied space.

The latter could be achieved by discretizing the points into a 3D voxel grid representation. If a point is found within the bounds of a voxel (i.e. a cell in a cubic grid in 3D space), that voxel is set to occupied. If no point is within the voxel, it is considered free. Although convenient and easily implemented, this could lead to incorrectly occupied voxels caused by outliers or noisy sensor measurements, which is very probable when using e.g. stereo cameras. Another major drawback with the voxel grid representation is that the entire extent of the map has to be known when allocated and it is furthermore not very memory efficient [17].

A more compact representation is a 2.5D map, such as an elevation map. Elevation map data is stored in a 2D grid, where each grid cell value corresponds to the average height of that cell. This could however be problematic if vertical objects are to be represented, and when considering e.g. underpasses or bridges where a cell is supposed to be free at a lower elevation but occupied further up. One solution would be to use multi-level surface maps, which allows more than one surface level in each grid cell [36].

OctoMap

OctoMap is a 3D-map framework based on octrees, which is a tree data structure in which each node can be divided into eight children. In the OctoMap framework, each node on a certain level correspond to a voxel with a certain side length, so that each voxel can be subdivided into eight equally sized voxels until desired map precision is obtained.

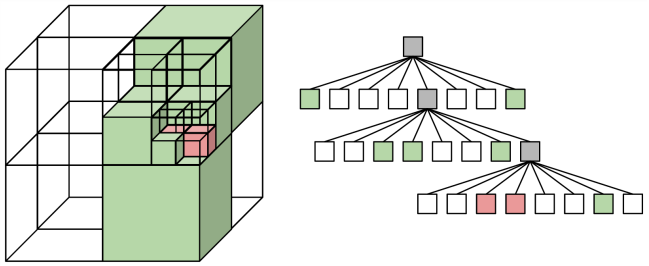


Figure 2.9: *Left panel: An OctoMap with free space (green), occupied space (red) and unexplored space (transparent). Right panel: The corresponding octree.*

OctoMap is a probabilistic map representation. Each voxel is assigned an occupancy probability, i.e. the probability of a certain voxel being occupied given sensor measurements up until the current time. If this probability is above a certain threshold, the voxel is considered as occupied, otherwise it is considered as free. The probabilistic framework allows for the creation of accurate maps even when the measurements are noisy. The probabilistic representation also has an innate means of removing outliers since solitary points likely will not lead to a voxel being considered as occupied. Another consequence of this is that moving objects will only be present in the OctoMap if they are static enough to be observed in the same location a sufficient number of times.

By utilizing the OctoMap framework, it is possible to model occupied, free and unexplored areas. Furthermore, it has the advantage of using high precision only where needed, making it very memory efficient. This property is illustrated in figure 2.9 where two occupied voxels are shown in red. Instead of allocating memory for voxels the same size as the smallest voxels, larger voxels can be used in uniform regions, reducing the total number of voxels in the map and hence also reducing the memory required to store the map [17].

3 Method and evaluation setup

An overview of the overall system architecture is shown in figure 3.1. First the cameras were calibrated and geometrically related to each other resulting in a set of intrinsic and extrinsic parameters. Using these parameters, the raw images taken from each of the four cameras were undistorted, rectified and cropped. This resulted in three pairs of grayscale images between the main camera (number 0) and the other three cameras, as well as an RGB image from the main camera. Using a correlation based block matching algorithm (SGBM) three depth maps, relating the main camera to the other three cameras, were generated. The depth map from cameras 0 and 1 was assigned as the main depth map, acting as a base for the merging with the other depth maps. Using estimated geometric transformations between the main and the other two depth maps, a single depth map was created, with the aim to cover a larger depth range than the individual depth maps. The merged depth map was sent together with the main camera RGB image to the RGB-D SLAM application, which generated an estimate of the pose trajectory and an OctoMap of the environment.

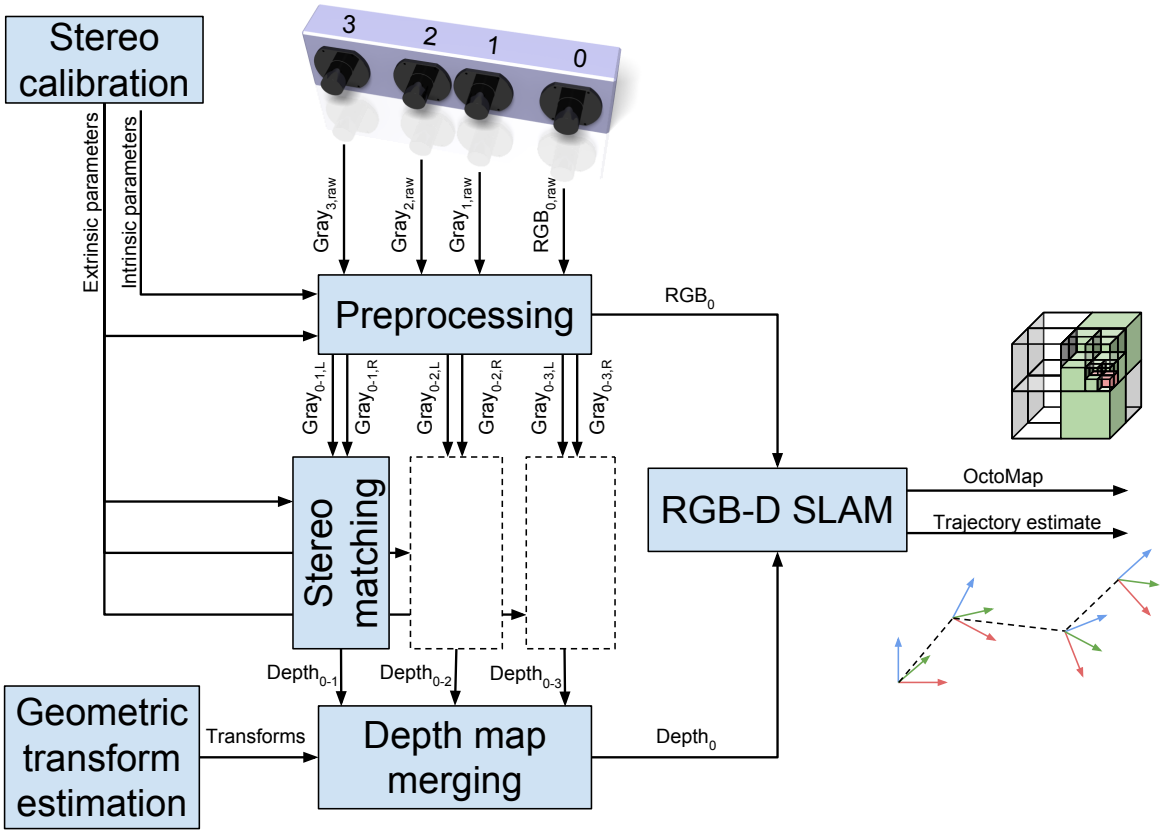


Figure 3.1: *Overview of system architecture.*

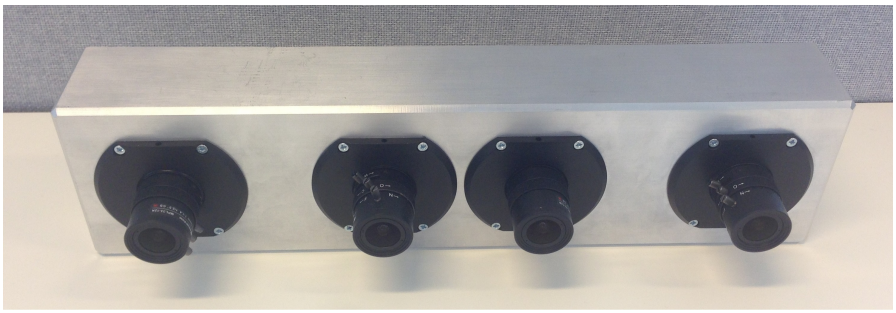
3.1 Hardware construction

A camera rig with four vertically aligned cameras was constructed. The casing of the camera rig, shown in figure 3.2, was milled from a solid block of aluminum. This made the entire structure rigid and hence not prone to flexing meaning that the extrinsic camera parameters were not subject to change. The aluminum casing of the camera rig also ensured that the heat dissipated from the cameras was evenly distributed over the entire rig, ensuring no overheating of the cameras. The main camera was paired with the other three cameras (see figure 3.5), providing three camera pairs with a baseline of 11, 19 and 30 cm respectively. The focal length of the lenses could be manually set in the range 3.3-12 mm, but since all stereo pairs shared the main camera they all had to be set to approximately the same length. Unfortunately, this means that the long and short range pairs have to be focused on the same distance.

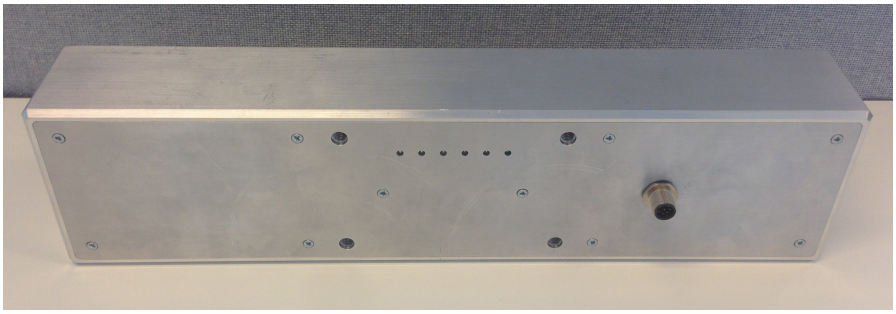
Figure 3.2: *A rendered 3D model of the aluminum casing of the stereo camera rig.*

The cameras used were 2 megapixel network cameras from Arecont Vision, model AV2116DNv1, although cropping and downsampling was performed internally so that the output images had a resolution of 960x464 pixels. The AV2116DNv1 cameras have global shutters, as opposed to the alternative rolling shutters. With a rolling shutter camera, the image sensor is exposed to light in a rolling fashion causing each row of the image sensor to collect the same amount of light but in slightly shifted time windows. This leads to geometric image distortions if the camera is moved relatively to the scene during image capture. With a global shutter camera however, the entire image sensor is exposed to light at the same time [23]. Maximum framerate of the cameras was 30 FPS, although the shutters were not synchronized. The lack of synchronization between the cameras could in worst case cause a timing error of $\frac{1}{60}$ s between corresponding frames in two cameras. Assuming an agent speed of 10 m/s, the environment will in the worst case move approximately 0.17 m likely reducing the quality of the resulting depth map.

The aluminum casings of the cameras were removed in order to mount them in the camera rig. The four cameras were internally connected to a 5-port D-Link Gigabit DGS-1005D network switch, which enabled data transfer over Ethernet in a single cable. The camera rig also contained a voltage regulator allowing a single power source to be used for the cameras and the network switch. Power and data cables were then joined in a single 8-connector output contact. The finalized camera system can be seen in figure 3.3. The software was implemented on a platform with a 5.0 GHz 8-core processor and 32GB 2133MHz RAM.



(a) *Front view*



(b) *Back view*

Figure 3.3: *The camera rig with four network cameras and joint data and power 8-connector contact visible.*

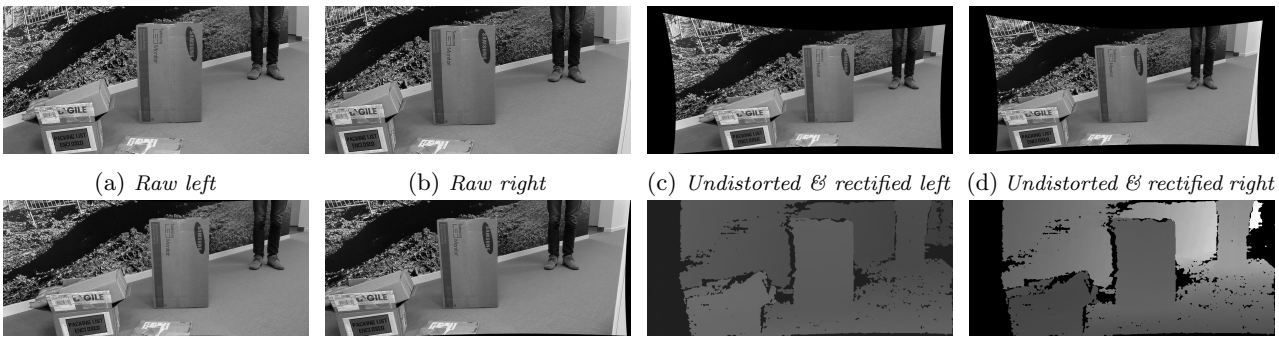
3.2 Camera calibration and parameter estimation

The estimation of the intrinsic and extrinsic camera parameters was performed using an asymmetric checkerboard, ensuring an unambiguous orientation determination of the checkerboard. The checkerboard was translated and rotated in between the different views in order to estimate all parameters with sufficient accuracy. By reprojecting the positions of the checkerboard corners with the suggested calibration model and observing the RMS of the reprojection errors, a metric of the accuracy of the calibration result was achieved. If the RMS was higher than a threshold of 0.3 pixels, the calibration of the camera or camera pair was considered unsuccessful and therefore redone.

Two different types of calibrations were performed; single camera (once for each camera) and stereo calibration (once for each camera pair). During single camera calibration, the main objective was to obtain an accurate estimate of the lens distortions. It was observed that when the calibration pattern was translated such that all parts of the image had at least once been covered by the calibration pattern, the most accurate lens distortion parameter estimates were obtained. The stereo calibration was instead aimed at finding an accurate estimate of the geometric transformation between the cameras rather than finding accurate estimates of the lens distortions of the individual cameras. It was discovered that observing the calibration pattern at different angles and distances was more important than observing the calibration pattern at all parts of the image in order to obtain an accurate estimate of how the cameras were geometrically related to each other.

3.3 Stereo vision

In order to create an accurate depth map, the steps depicted in figure 3.4 had to be performed. First, the raw images from the cameras, (a) and (b), were undistorted and rectified in order to successfully be able to compare the images. This included finding values for the distortion coefficients k_1, k_2, p_1, p_2 and k_3 as well as estimating the geometric transformation relating the cameras in each camera pair. The undistortion and rectification process will generate invalid pixels, seen as black regions along the edges in (c) and (d). The ROI within each rectified image was selected as the largest rectangle containing only valid pixels. Each image was then cropped to remove invalid pixels, by omitting the regions outside the union of the individual ROIs, giving (e) and (f). Using the semi-global block matching algorithm, SGBM, a disparity map of the surroundings was generated (g). SGBM is the OpenCV implementation of the SGM method described in section 2.1.4, and is largely equivalent except for some pre- and post-processing steps and that correlation is computed on blocks of pixels rather than single pixels [29]. By using the relation between disparity and depth described in equation 2.10, the corresponding depth map was computed (h).



(a) *Raw left* (b) *Raw right* (c) *Undistorted & rectified left* (d) *Undistorted & rectified right*

(e) *Cropped left* (f) *Cropped right* (g) *Disparity map* (h) *Depth map*

Figure 3.4: *Example of the most important steps in the stereo vision process. The raw images from the left (a) and right (b) camera are undistorted and rectified (c, d), and cropped (e, f). A correlation based block matching algorithm is then used on the cropped images to calculate a disparity map (g) which can be reprojected to a depth map (h).*

3.4 Depth map merging

Pairing the cameras as in figure 3.5 gave three stereo camera pairs with different baselines, all sharing the leftmost camera (number 0). The procedure described in section 3.3 was performed once for each camera pair to produce three rectified RGB-images and their three associated depth maps. The RGB-images were thus actually captured with the same camera and the corresponding depth maps depicted the same view, although with different depth ranges depending on the baseline of the camera pair. Due to differences in the camera calibration of each stereo pair, a joint depth map could not be achieved by simply overlaying the depth maps. However, since the three depth maps are related spatially the same way as the three RGB-images, finding a geometrical transformation which relates the RGB-image enables depth map overlaying if the same transformation is first applied to the depth images.

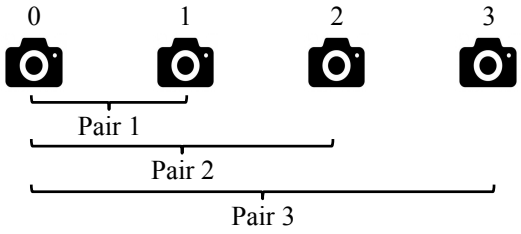


Figure 3.5: *The leftmost camera was selected as the main camera, which was the left camera in each stereo pair.*

This geometrical transform was approximated as a perspective transform. A perspective transform P transforms a point (x, y) into (x', y') according to

$$P = \begin{bmatrix} a_1 & a_2 & t_1 \\ a_3 & a_4 & t_2 \\ p_1 & p_2 & 1 \end{bmatrix} \qquad x' = \frac{a_1 x + a_2 y + t_1}{p_1 x + p_2 y + 1} \qquad y' = \frac{a_3 x + a_4 y + t_2}{p_1 x + p_2 y + 1} \tag{3.1}$$

where a_1, a_2, a_3 and a_4 describes the scaling, rotation and skewing, t_1 and t_2 describes the translation and p_1 and p_2 describes the perspective projection. In order to calculate the eight parameters of P in equation 3.1, four or more pairs of corresponding points (x_i, y_i) and (x'_i, y'_i) are needed [18].

To find these four pairs of corresponding points, SURF-keypoints were detected in the rectified and undistorted RGB-images. Each keypoint was assigned a descriptor and the descriptors were matched between the images. In one of the images, the keypoint in each quadrant with the shortest distance in descriptor space to the corresponding keypoint in the other image was selected, together with the corresponding points in the other image.

The calculated perspective transforms were then applied to the depth maps, resulting in three depth maps $D_1(x, y)$, $D_2(x, y)$ and $D_3(x, y)$ from camera pair 1, 2 and 3 respectively. For all x and y, these images either has a depth value in millimeters or 0 if there is no depth available. Depth could be missing for mainly two reasons; either because the calculated depth was outside the optimal range of the camera pair or because of stereo correspondence occlusions. The merged depth image $D_m(x, y)$ was then assigned with highest priority for the depth data from the camera pairs with shorter baseline as is described in algorithm 1.

```
foreach x,y do
    if D_1(x,y) ≠ 0 then
        D_m(x,y) := D_1(x,y);
    else
        if D_2(x,y) ≠ 0 then
            D_m(x,y) := D_2(x,y);
        else
            D_m(x,y) := D_3(x,y);
        end
    end
end
```

Algorithm 1: Procedure for merging three depth maps.

3.5 Visual SLAM

RGB-D data, consisting of a merged depth map, the corresponding RGB-image and complimentary camera parameters was passed as ROS messages (see section 3.6) at a rate of approximately 1-3 Hz to RGB-D SLAM. From the RGB image, features were extracted using one of the combinations of detectors and descriptors presented in table 3.1 and assigned depth data from the depth map. The pose was computed frame by frame in the front-end and the entire pose trajectory was continuously optimized in the back-end. For each frame, an OctoMap was incrementally built from the pose and the corresponding point cloud either online or offline.

Using the online approach, the points were added to the OctoMap grid in real time. Besides being more computationally expensive, this approach has the disadvantage that once the frame has been added to the OctoMap, a retroactive change of the map is not possible. Optimizations of the pose trajectory, such as those occurring at loop closures, will therefore only affect the current pose and not correct the location of the point clouds that has already been projected to the OctoMap. After the frame has been added, the corresponding point cloud can be cleared to free memory.

If instead the offline approach was used, the OctoMap was constructed after the mapping session was finished using the optimized pose trajectory and the set of stored point clouds. This will generate a more accurate OctoMap, especially if loop closure has occured, but storing the point clouds for each frame will for obvious reasons require an extensive amount of memory.

3.6 Implementation

The entire system was implemented in C++ as executables (nodes) using the ROS framework. ROS, Robot Operating System, is a collection of libraries and tools aimed at simplifying the process of creating robot applications. It is an open source, pseudo-operating system which provides features such as hardware abstraction, visualization tools and message-passing functionality. Recorded data can be stored as so-called bag files, allowing for offline data evaluation and simulation [26]. There are also a large number of third party packages available, such as the RGB-D SLAM package or support for relating different robot link coordinate frames through a system of transformations.

The computer vision-related tasks (including e.g. feature extraction and stereo correspondence) in the ROS nodes and the evaluation executables were implemented with the help of the OpenCV library, an open source library written in C and C++. Furthermore was MATLAB R2014b with the Computer Vision Toolbox used for rapid prototyping and data analysis.

3.7 Evaluation setup

As the objective of this thesis is to construct a system, the primary result is an evaluation of how well this system and its submodules perform on their respective tasks. Thus, a series of evaluations were performed and the experiment setups are explained below.

3.7.1 Depth estimation

The depth reconstruction accuracy was evaluated by comparing the distance given by the depth maps from each of the three camera pairs with a known ground truth distance determined by a high precision laser rangefinder. Distance was measured to a planar and texture rich target, which was placed perpendicular to the camera planes. Measurements were collected at discrete distances, ranging from 1-20 meters from the target. For each camera pair and distance, ten measurements were collected. Each measurement consisted of the mean of nine quadratically positioned measured points which in turn consisted of the mean intensity (distance) of a 10×10-pixel patch in the depth map, as illustrated in figure 3.6. Since a zero indicates that either the target is out of range or that no correspondence was found for that pixel, only non-zero values were considered.

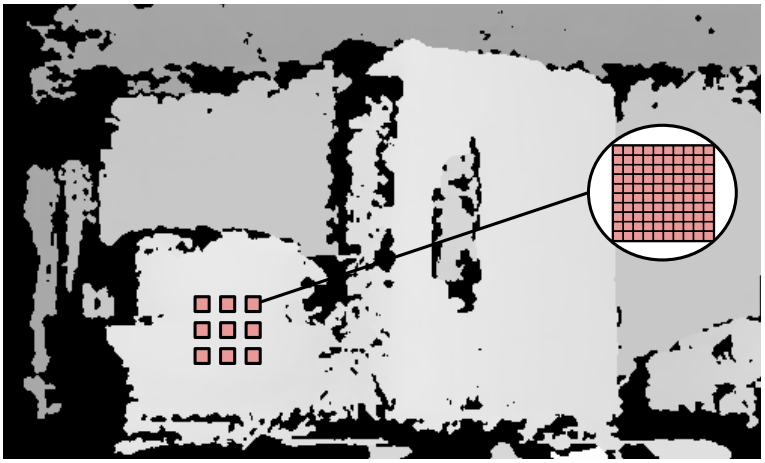


Figure 3.6: *Schematic view of one distance measurement, with nine patches each consisting of* 10×10 *pixels.*

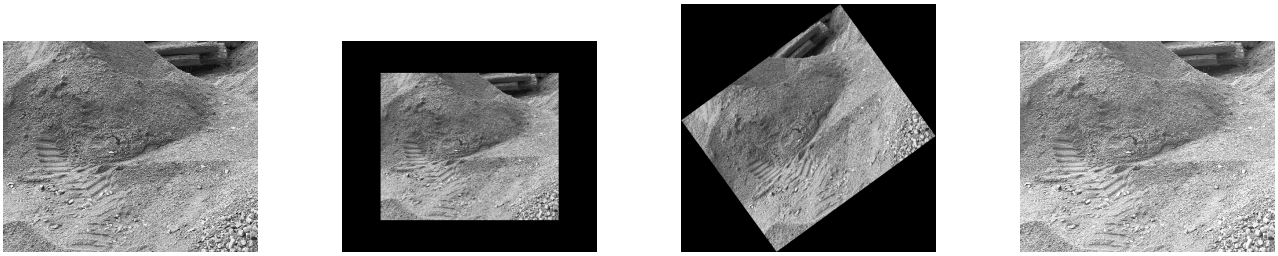
3.7.2 Feature extraction algorithms

In order to find a suitable setup for the SLAM frontend, 14 different combinations of feature detectors and extractors were evaluated with the objective to find a suitable combination in some key aspects; rotation-, scale- and illumination invariance as well as computational speed. The OpenCV implementation of the algorithms were used, unfortunately ruling out the use of BRISK as feature detector since this is defective in OpenCV 2.4.6. The used feature detectors were SIFT, SURF and ORB, whereas SIFT, SURF, ORB, BRISK and FREAK were used for extracting the descriptors. SIFT as detector and ORB as descriptor turned out to be an incompatible combination and was thus removed from the experiments.

Table 3.1: Overview of the 14 evaluated combinations of feature detectors and extractors.

ID	Detector	Descriptor	Distance measure
1	SIFT	SIFT	Euclidean
2	SIFT	SURF	Euclidean
3	SIFT	BRISK	Hamming
4	SIFT	FREAK	Hamming
5	SURF	SIFT	Euclidean
6	SURF	SURF	Euclidean
7	SURF	ORB	Hamming
8	SURF	BRISK	Hamming
9	SURF	FREAK	Hamming
10	ORB	SIFT	Euclidean
11	ORB	SURF	Euclidean
12	ORB	ORB	Hamming
13	ORB	BRISK	Hamming
14	ORB	FREAK	Hamming

The evaluation was conducted in the following way: First, a training image portraying a feature-rich scene was taken with the left inner camera. From this training image, the features for the training set were extracted using one of the 14 detector-/extractor combination, specified in table 3.1. Query images were then created by rotating, scaling or adjusting the intensity in the training image, as illustrated in figure 3.7. Rotation was performed in the range 0-180 degrees, scaling between 50-150 % and intensity was increased between 0-50 %. The keypoint locations in the training set were then geometrically transformed (in case of rotation or scaling) the same way as the training image to produce a ground truth reference for the query set. This query set of features was created by utilizing the same detector-/extractor- combination as when the training set was created.



(a) *Training image* (b) *Scaling* (c) *Rotation* (d) *Illumination*

Figure 3.7: *The training image (a) and examples of the three different distortions used to create the query images. The distortions are (b) scaling to 70 % of the original size, (c) rotation by 36 degrees and (d) intensity increased by 10 % to simulate varying illumination conditions.*

A bruteforce matching was then performed, where each keypoint in the query set was assigned a best match by comparing them to every keypoint in the training set. If the location of the query set keypoint was within 3.0 pixels from the actual location of its believed match, it was considered to be a correct match. Since feature redundancy is beneficial to the visual odometry, the absolute number of correct matches was equally important to determine as the percentage of correct matches. Furthermore, the computational time was measured by dividing the processing time for the detection, extraction and matching by the number of found keypoints when running on a dual core, 2.5 GHz Intel Core i5 processor.

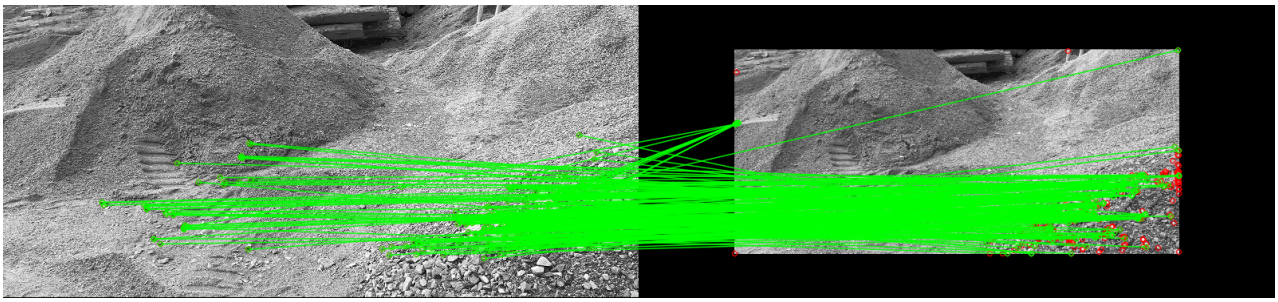


Figure 3.8: *Illustration of the keypoint matches to the training image for a scaled query image. A few obvious incorrect matches can be seen in the upper left quadrant of the query image as well as in the upper right corner.*

3.7.3 Benchmarking the SLAM system

To determine the optimal configuration of the constructed SLAM system, it is necessary to have a performance metric for the entire system. Sturm et al. have proposed a methodology and provided a dataset for benchmarking 3D SLAM applications with RGB-D data input [34]. For a number of reasons, primarily related to the fact that we provide our own data from the stereo camera system, this was not possible to implement. However, their evaluation metrics were used as an inspirational source when designing our methods, which are described below.

When benchmarking the output of a SLAM application against a ground truth sequence, there are two main approaches as there are primarily two types of output; a map from the mapping task and an estimated trajectory from the localization task. If the generated 3D map is subject to evaluation, it could for instance be projected on to a 2D plan of the mapped surroundings. Although there are automated tools for doing this, map errors are often simply determined upon visual inspection.

Sturm et al. instead advocates a method where the estimated trajectory is compared to a precise ground truth sequence. They determine this sequence by monitoring the camera pose with motion capture technology and a set of high speed cameras. As such a system could not be utilized in this project, we had no access to a proper ground truth sequence. This was solved by mounting the camera rig on a trolley and pushing it along a known rectangular course, closely following marked lines in the floor. Since the geometry and extension of these lines could be determined with a high precision laser range finder and the trolley was always moved in a horizontal plane, it is possible to describe the translational component of the ground truth trajectory. We also took advantage of the horizontal movement by assuming that the vertical axis of the true trajectory was always parallel with the z-axis. We thus have a primitive ground truth sequence where five of the six degrees of

freedom (x-, y- and z-position as well as roll and pitch) are known and only the yaw is uncertain. Since the map is registered from the optimized poses in the estimated trajectory, the accuracy of the trajectory estimate is also a measure of how accurate the map will be.

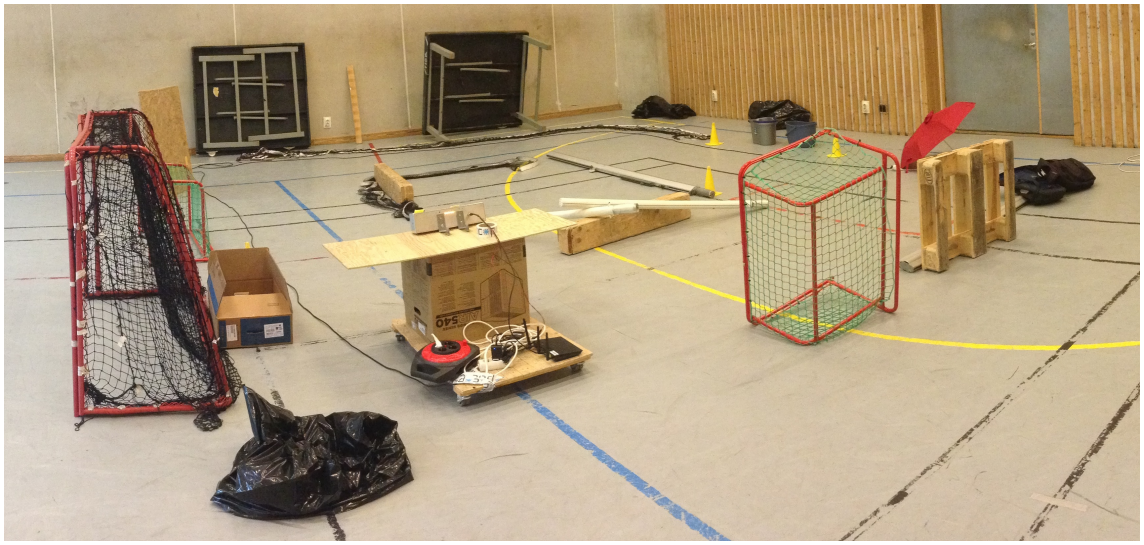


Figure 3.9: *Experiment setup for the evaluation of the SLAM system.*

Furthermore, a number of obstacles were placed on both sides of the course to allow for more visual features to be found. The entire experiment setup, including the course, the obstacles and the trolley with the camera system, is shown in figure 3.9. The processed stereo output, i.e. a merged depth map and an RGB image, was recorded as ROS bagfiles for offline execution of the SLAM application. Four different sequences were recorded, one with camera pairs 1 and 3, one with all three pairs and two with pair 1 and pair 3 alone.

The bagfiles were processed offline with RGB-D SLAM and a (post-optimization) trajectory array was exported, including a timestamp and the camera pose for each added node. The estimated trajectory was then rotated 3.5° around the z-axis and −14.0° around the y-axis to compensate for that the camera was tilted the corresponding amount during the test.

The data was then analyzed in four primary categories; translational error, angular error, lost frames and memory usage.

- The translational error was calculated as the shortest distance between each estimated position and the ground truth course.
- Angular error was the 3D angular deviation (in degrees) between each estimated z-axis orientation and a vertical unit vector.
- Lost frames was determined by comparing how many nodes that were added to the pose graph with the total number of recorded depth map frames.
- The amount of free memory was monitored once every second while only running the RGB-D SLAM application. The decrease of free memory during execution gave an indication of the memory usage for different types of feature extraction configurations.

4 Results and discussion

4.1 Depth estimation

The results from the evaluation of the depth reconstruction are largely in accordance with the expectations. Due to the inversely proportional relationship between distance and disparity as described in section 2.1.5, the camera pair with the widest baseline will have an advantage in long-range depth resolution over the pairs where the cameras are closer together. As can be seen in figure 4.1, camera pair 3 generally has the smallest error fluctuation in the distance readings. As the distance to the target increases, the two camera pairs the narrower baseline struggles to give consistent readings, whereas the third camera pair more closely follows the linear trend and has a lower standard deviation of the error for longer distances.

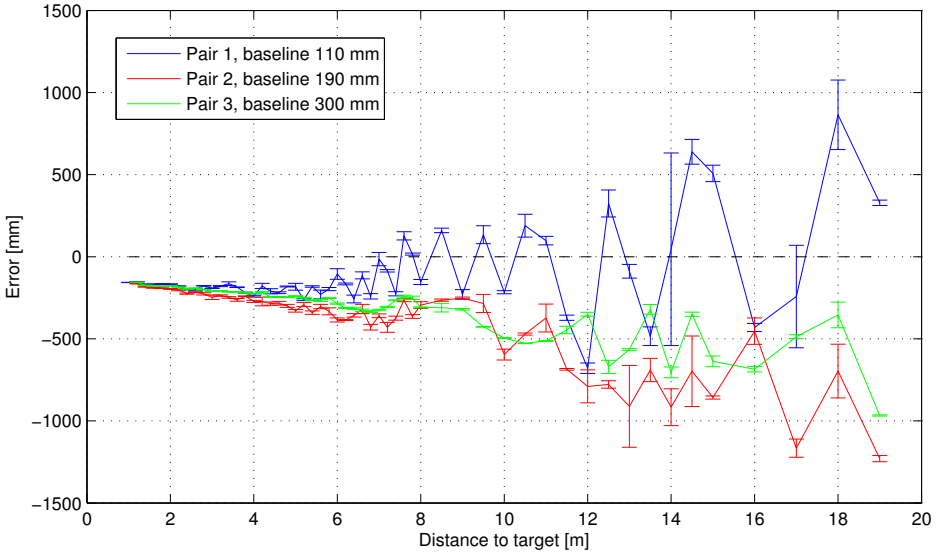


Figure 4.1: *Mean deviation from ground truth distance for the three different camera pairs with baseline of 110, 190 and 300 mm respectively at 1-20 meters distance from the measured object. The horizontal bars above and below each mean data point correspond to one standard deviation.*

In the other end of the trade-off between wide and narrow baselines, pair 2 and 3 fail to return a depth estimation for the shortest distances, suggested by the missing 1.0-meter measurements in figure 4.2. This is due to the fact that close range disparity is too large when a wider baseline is utilized, and the stereo correspondence algorithm thus fails to find a suitable match below the specified maximum disparity. If the maximum disparity was to be set higher, matching would be more computationally expensive.

Although more precise for most distances, camera pair 3 is not necessarily the best choice at all times. Apart from the problem with surpassing the maximum disparity at short distances, other factors also impact the choice of camera pair. The field of view is smaller for camera pairs with wider baseline making the use of camera pairs with short baseline desirable. There will also be more occlusions at short range increasing the portion of the depth map where depth cannot be determined. The stereo correspondence problem will furthermore be more difficult as similarity decreases even for correctly matched image regions since the views are more different with a wider baseline.

Furthermore, it can be noticed that the depth estimations from all three camera pairs deviate from the ground truth and exhibit a drifting behavior, possibly due to the fact that the camera planes are not parallel and coinciding with the face of the camera rig from which the ground truth distance was measured. As is suggested by the least squares approximations shown as dashed lines in figure 4.2, the error drift can be relatively well approximated by linear functions. This indicates that it is possible to compensate for deviation from the true depth by subtracting this linear trend from the raw readings. Hence is a small error variability of higher importance than a small absolute error.

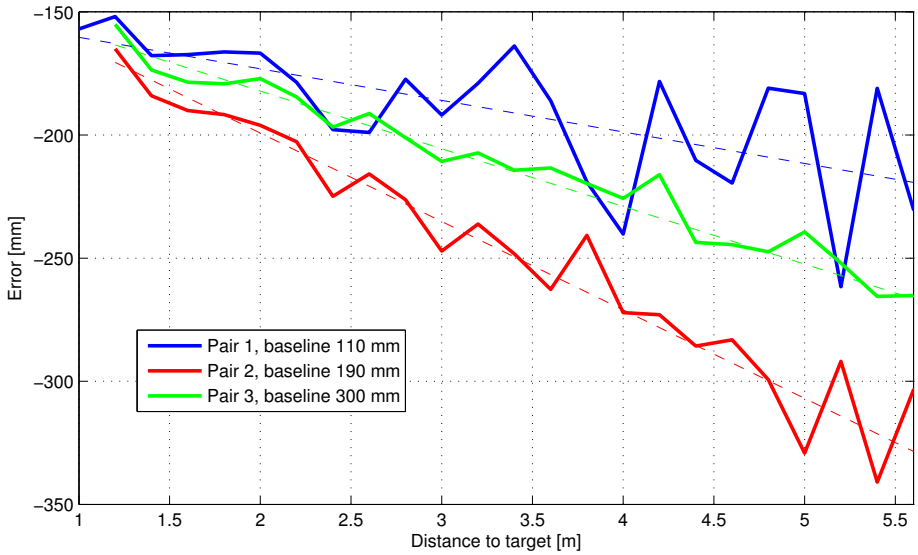


Figure 4.2: *Mean error for shorter distances, with an approximated linear trend plotted with a dashed line for each camera pair.*

In conclusion, it is reasonable to use camera pair 1 on short range and then use camera pair 3 on medium and long range. The results also suggest that the second camera pair should not be used, at least with the current camera calibration. This camera pair is outperformed by the narrow pair on close distance and by the wide pair on long distance, and we fail to find a suitable interval on medium range where the second pair is needed. As it is a computationally very costly operation to perform stereo correspondence and depth map merging with an additional camera pair, it is our belief that any possible minor gains in accuracy do not motivate the drastic increase in computational time.

4.2 Feature extraction algorithms

The total processing time per keypoint for each evaluated detector and descriptor combination is shown in figure 4.3. The processing time is an average of the duration of feature detection, description and matching for both differently scaled, rotated and illuminated images.

Clearly, four combinations are substantially slower relative to the others; SURF/SIFT, ORB/SIFT, ORB/SURF and ORB/FREAK. Using a binary detector compared to a vector-based detector produces much fewer keypoints as is seen in table 4.1. Few detected keypoints in combination with relatively slow extraction time makes ORB/SIFT, ORB/SURF and ORB/FREAK poor detector/extractor pairs, since a high number of keypoints is required to accurately relate the poses between consecutive frames in a SLAM application.

Table 4.1: Total number of found keypoints in the undistorted training image for all combinations of detectors and descriptors. The feature extractors are abbreviated as follows: SI (SIFT), SU (SURF), O (ORB), B (BRISK) and F (FREAK).

Extractor pair	SI/SI	SI/SU	SI/B	SI/F	SU/SI	SU/SU	SU/O
Found keypoints	1924	1924	1716	1504	6440	6440	5670
Extractor pair	SU/B	SU/F	O/SI	O/SU	O/O	O/B	O/F
Found keypoints	5288	3557	500	500	500	254	26

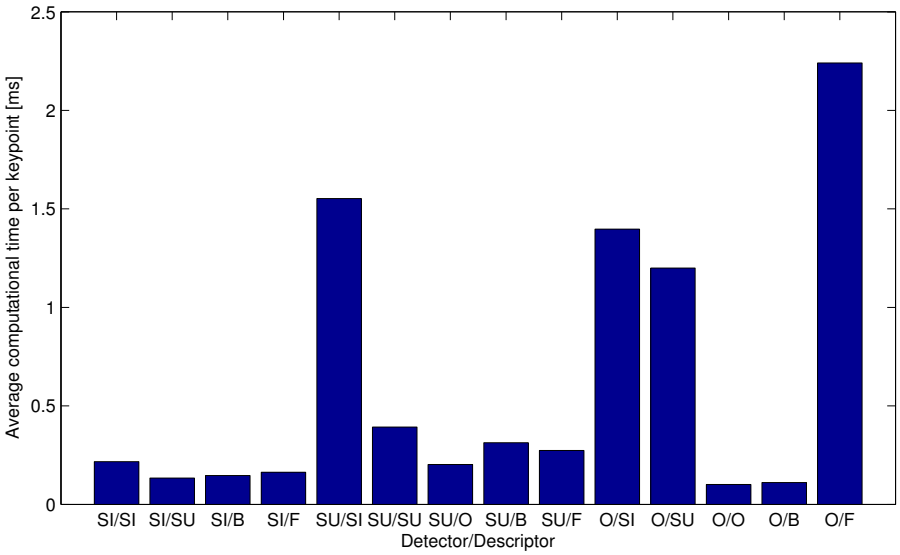


Figure 4.3: *Average computational time per matched keypoint for detection, description and matching with rotated, scaled or increased illumination in the query images.*

When examining processing times only with rotated images, as is illustrated in figure 4.4, it is noticeable that regardless of whether SIFT, SURF or ORB is used as detector, SIFT is always slowest for description and matching. This finding is in line with the expected results, since the calculation of the histogram of oriented gradients in the SIFT-descriptor is known to be a very time consuming process. A more surprising finding however, was that the matching was not significantly faster when binary features were utilized instead of the vector-based features SIFT and SURF. The binary features are evidently faster than SIFT and SURF, but the most noticeable time gains are instead related to a shorter description time for the former feature type rather than a quick matching.

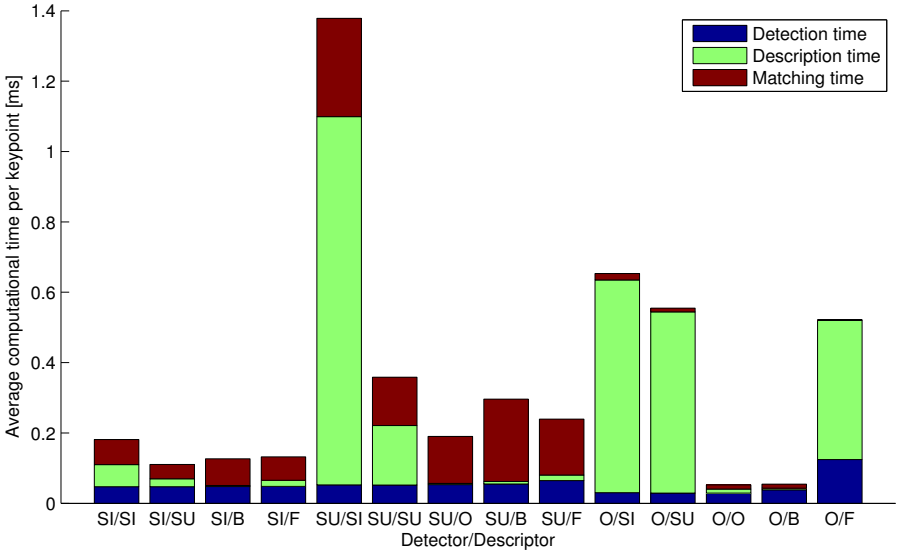


Figure 4.4: *Average computational time per matched keypoint for detection, description and matching with rotated query images.*

As the bruteforce matching compares a keypoint in the query image to every keypoint in the training image, the matching time does not increase linearly with the number of keypoints as the detection or description time

does. Matching time per keypoint is thus not a very accurate metric when the number of found keypoints is not approximately equal. For that reason, comparing the matching time for SURF/SURF with e.g. ORB/BRISK is not very fair, since the former matches 12 times more keypoints. However, if the same detector is used, the results give an indication if matching is done efficiently since approximately the same number of keypoints are generated.

Another negative aspect of the bruteforce matching approach is that every query image keypoint has to be matched to a training image counterpart, regardless if such a keypoint counterpart exists or not. There is thus a risk that a correct match for a query image keypoint is already occupied by an incorrect match, if the incorrect match did not find a better correspondence. This is not not how it is done in RGB-D SLAM, where instead only features that are similar enough according to the comparison metric are considered matches. However, we are also interested in finding feature detectors that have a high degree of repeatability - i.e. they detect the same keypoint if the same scene is reobserved. If the detector repeatability is sufficiently high, correct matches being occupied by incorrect ones is less of a problem.

When analyzing robustness regarding change in rotation, scaling and illumination, the four slowest extractor pairs from figure 4.3 were eliminated from further evaluation. The effect of rotation distortion is shown in figure 4.5. For most extractor pairs, the ratio of correct matches is periodic with peaks in 90° intervals. This is quite natural, since orientation normalization is most accurate around these peaks due to the horizontal-vertical geometry of a 2D image. However, for some extractor pairs, matching is less correct at exactly 0° and 90° when the conditions are presumably ideal. This is likely due to numerical imprecision of the rotation matrix and the inverse transformation utilized to rotate the query image. The same tendencies are observed when scaling the image by a factor 1, as seen in figure 4.6.

Some extractor pairs stand out: SIFT/SIFT and SIFT/BRISK are remarkably robust against rotation whereas SURF/SURF and especially SIFT/SURF are remarkably sensitive to rotation. It should be noted that extractor pairs that use SURF as descriptor exhibited the worst performance, whereas using BRISK resulted in the best performance. Furthermore are the SIFT/SURF (and also SIFT/FREAK) combinations deviating from the periodic pattern, by performing gradually worse as rotation is increased. As they so drastically differ from the behaviour of other extractor pairs, their sub-par performance is likely related to implementation issues.

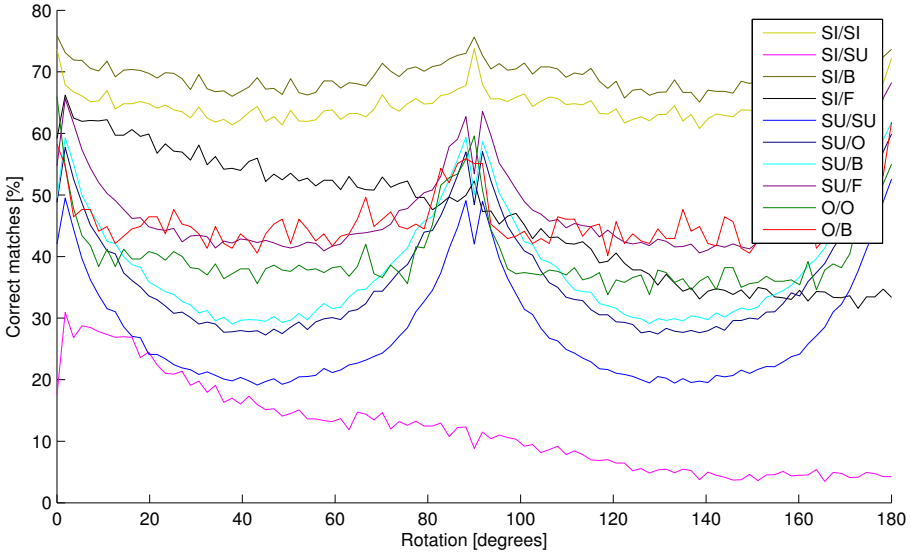


Figure 4.5: *Ratio of correct matches with rotated query images.*

Changing the scale of the query image less than approximately ±15% leads to a performance decline of similar rate for all extractor pairs, as is illustrated in figure 4.6. SIFT/SIFT and SIFT/BRISK turns out to be the most correct and robust extraction pairs, although the latter has a tendency to yield incorrect matches when the query image is heavily scaled. Once again is the SURF/SURF pair not performing as well as its competitors, and extractor pairs using SURF as descriptor exhibits the worst performance up until approximately ±20% scaling. This is especially the case for the SIFT/SURF combination.

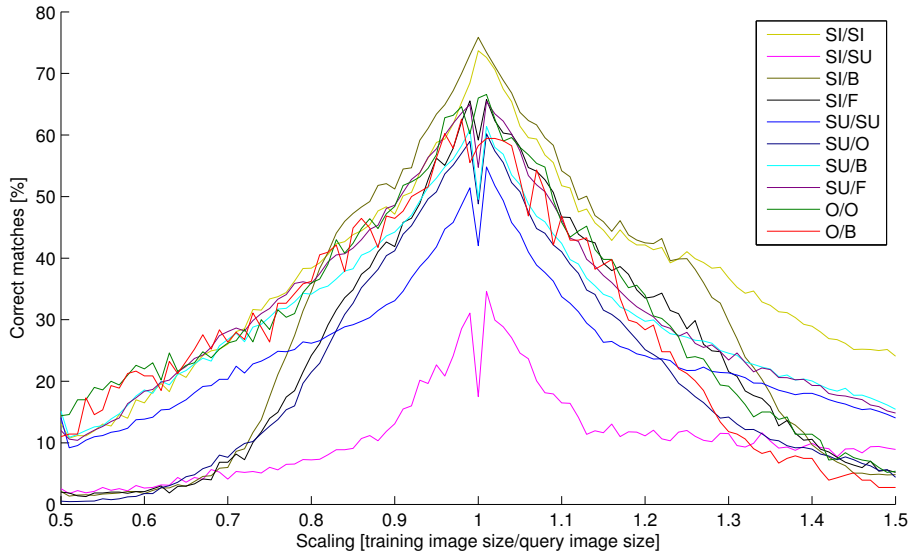


Figure 4.6: *Ratio of correct matches with scaled query images.*

The illumination experiment, where the intensity of the query image was altered, did not suggest any significant difference between the different extractor pairs as is shown in figure 4.7. The most prominent exceptions from this statement are that SIFT/SURF once again appears to struggle more than other extractors and that SURF/FREAK is slightly more robust under the specified conditions.

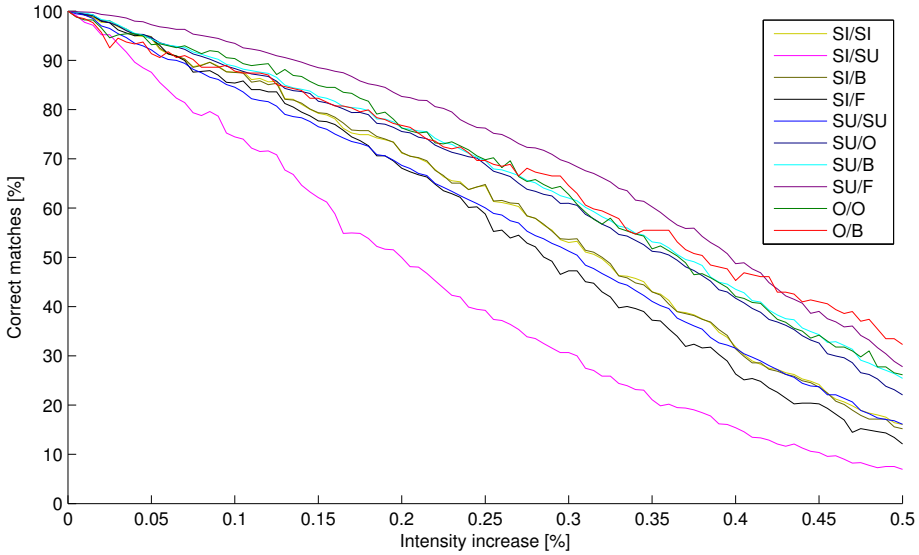


Figure 4.7: *Ratio of correct matches with gradually more illuminated query images.*

In conclusion, our experiments reinforce the widely accepted depiction of SIFT as a very robust but also slow feature extractor. We have furthermore seen that the binary features are extremely fast, but the low number of detected keypoints might have implications for the SLAM front-end. SURF has shown to be a wiser choice for a detector, since it generates the most number of keypoints by far. Using SURF for feature description is not ideal, as its scaling and rotation invariance is not as impressive as its competitors. However, combining SIFT or SURF detectors with BRISK as descriptor could be an interesting approach for a visual SLAM application, since it preserves the high robustness and plentitude of features from the vector-based features but adds the faster extraction and lower memory usage from the binary features.

4.3 Trajectory estimation

The first evaluation was performed to decide which combinations of camera pairs that were most suitable for the SLAM application. The same SLAM configuration was used for all four pair combinations and the result is summarized in table 4.2.

Table 4.2: Key result parameters of trajectory estimation when evaluating camera pairs.

Camera pair	**Pair 1**	**Pair 3**	**Pair 1 & 3**	**Pair 1, 2 & 3**
RMS of translational error	0.4739 m	1.1141 m	0.6008 m	0.3318 m
Added frames	91.52%	87.22%	99.64%	99.34%
Mean angular deviation	11.11°	17.80°	10.95°	11.01°

There is no doubt that there is a correlation between the accuracy of the trajectory estimate and which camera pairs that were used or merged. For instance, camera pair 3 performs significantly worse than the other camera pair combinations. The position error is larger, the heading is more inaccurate and fewer frames are added than with any other combination. This is exemplified further in figure 4.8, where the estimated position of the cameras is shown when only the third camera pair is used. A more successful example is shown in figure 4.9, where depth images from camera pairs 1, 2 and 3 was merged into one depth map.

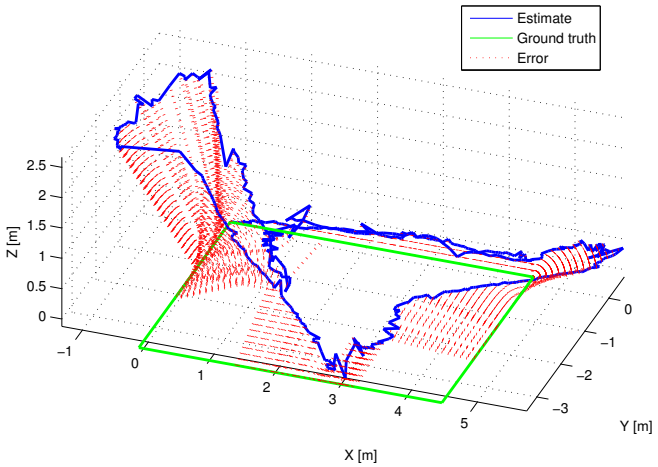


Figure 4.8: *Example of an unsuccessful localization using camera pair 3. The figure shows the estimated trajectory, the ground truth course and the deviation between the estimate and the closest ground truth segment.*

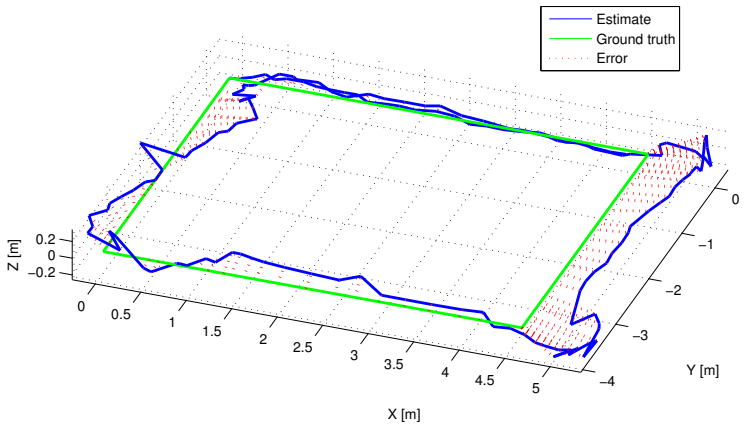


Figure 4.9: *Example of a more accurate localization than in figure 4.8, using camera pair 1, 2 and 3.*

Since the majority of the objects were relatively close to the camera pairs, camera pairs with poor depth accuracy at close range (such as pair 3) suffered in this test, explaining the vast difference in performance between these two camera pair combinations. If no depth is estimated for a feature close to the camera, the feature can not be added to the constraints for that node. If too few features are added so that the inlier threshold is not exceeded when matching the frame against other nodes, the entire frame will be ignored and the node is not added to the graph. It is also more difficult to estimate ego-motion if many features or nodes are ignored, explaining the large RMS of the translational error and the inaccurate heading estimate for camera pair 3.

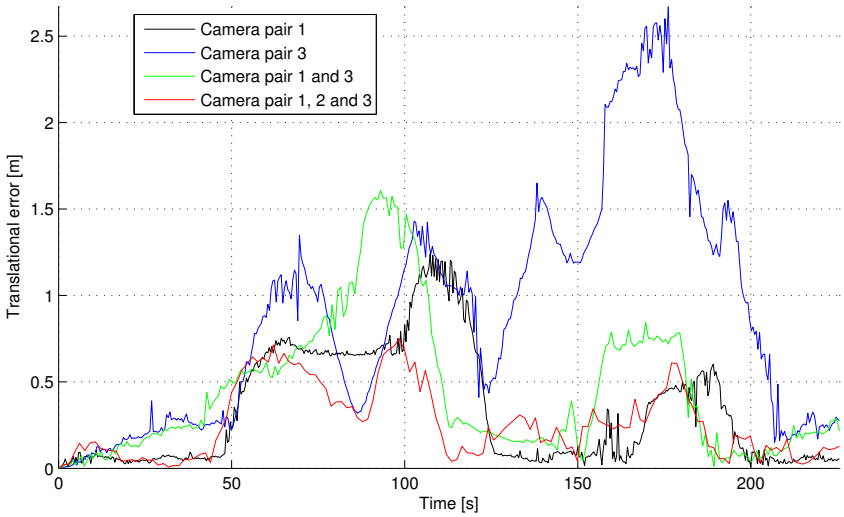


Figure 4.10: *The translational error between the estimated and true position at each time step for different camera pair combinations.*

The deviation between the true and estimated position using different camera pair combinations is shown in figure 4.10. As mentioned above, it is clear that camera pair 3 performs worse than the other camera pair combinations. It is furthermore suggested that the combination of camera pair 1, 2 and 3 outperforms the other camera pair combinations since it has the smallest area under the curve and also the smallest maximum error of approximately 75 cm.

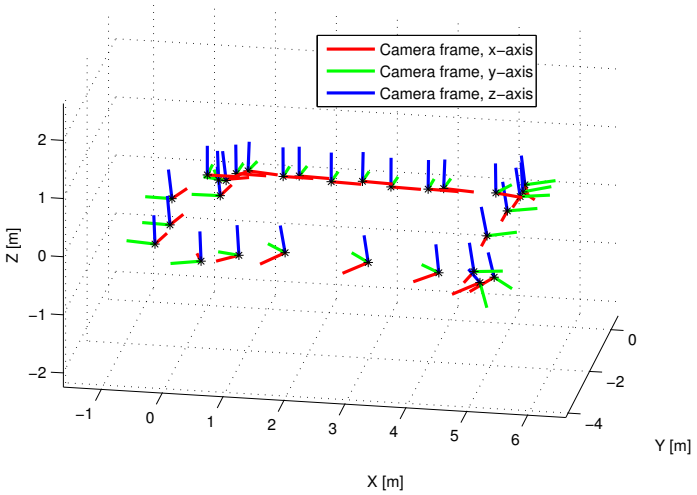


Figure 4.11: *The orientation of the camera frame when moving in the same path as in figure 4.9 using all four cameras.*

When building a 3D map, a good position estimate is not sufficient though. The orientation estimate is of equal or greater importance, since this controls how each point cloud is registered in each node of the graph. To illustrate how the orientation of the cameras is estimated when using all three camera pairs, a set of axes representing the coordinate axis orientation of the camera frame is plotted in figure 4.11. It can be observed that the blue z-axis is vertical or close to vertical for most of the time, which indicates an accurate orientation estimation. This indication is however not sufficient as sole quality metric, since the error of the heading in the xy-plane was not analyzed.

The deviation between the true and estimated heading using different camera pair combinations is shown in figure 4.12. It is worth noting that at approximately the same position (at the third turn, at approximately 110 seconds) all camera pair combinations experience a drastic increase in angular error leading. This might have to do with a lack of visual features at said location in combination with a large change in heading. The errors of all evaluated camera pairs are similar, except for camera pair 3 which has a large persisting angular error up until approximately 200 seconds.

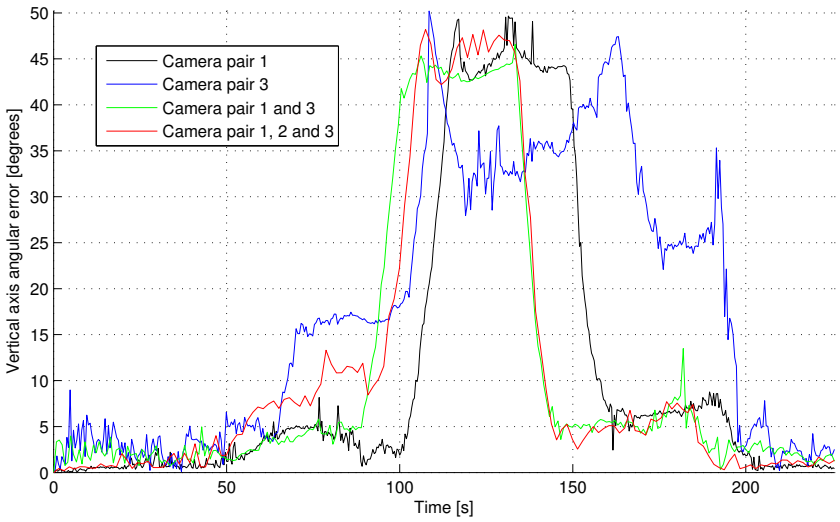


Figure 4.12: *The angular error between the estimated and true heading at each time step for different camera pair combinations.*

While executing the evaluations of the SLAM system, an important discovery was made regarding the use of multiple camera pairs. The frame rate of the depth maps was found to be inversely proportional to the number of camera pairs used. Using one, two or three camera pairs gave a frame rate of approximately 3, 1.5 and 1 FPS respectively. This result is in line with our expectations since the processing of the images and creation of the depth maps was found to be the bottle neck in the system, and hence doubling or tripling the number of depth maps generated will lead to a decrease in frame rate of the same magnitude.

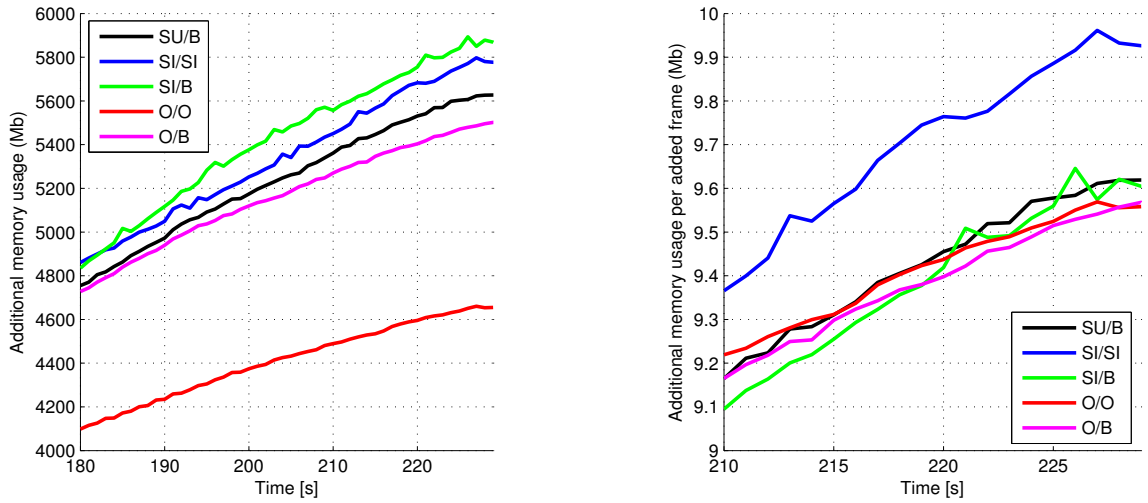
Although not affecting the results when the cameras are manually pushed around an indoor course, a low frame rate might be troublesome when the camera system is implemented in a fast-moving vehicle. It is therefore necessary to thoroughly evaluate the performance in a real vehicle before making the claim that a multi-camera system is always better than a single camera pair. This is however beyond the scope of this thesis.

The second part of this trajectory estimation evaluation concerns another central theme in the visual SLAM process; the choice of feature extraction method. The five most promising combinations from the experiments described in section 4.2 were used on the test data recorded with camera pair 1. The key results are summarized in table 4.3. The most striking result illustrated here is probably that ORB/ORB has slightly smaller mean angular deviation of the z-axis, but performs substantially worse considering the other two quality metrics. There is otherwise no significant difference between the results generated with the SIFT descriptor and those where the binary descriptor BRISK is used. In particular the combinations where SIFT or SURF detectors are paired with BRISK descriptors proves to be very robust.

Table 4.3: Key result parameters of trajectory estimation when evaluating feature extraction configurations using camera pair 1.

Feature detector/descriptor	**SU/B**	**SI/SI**	**SI/B**	**O/O**	**O/B**
RMS of translational error	0.4711	0.4479	0.5082	0.8729	0.5052
Mean angular deviation	10.99°	11.03°	11.63°	10.68°	12.59°
Added frames	91.84%	91.37%	95.92%	76.45%	90.27%

The total memory usage and the memory usage per added frame is shown in figure 4.13a and 4.13b respectively for five different extractor pairs. Comparing figure 4.13a with the bottom row in table 4.3, it is clear that the final memory usage is largely dependent on the number of added nodes. It is thus evident that a large portion of the total memory used by RGB-D SLAM is allocated to storing the depth (cloud) data for each node, to such an extent that the choice of feature descriptor is more or less completely overshadowed. By dividing the total memory allocation with the number of added nodes (which is done with the data in figure 4.13b) we see that extractor pairs with binary descriptors has a lower memory usage compared to the vector-based extractor pair SIFT/SIFT. This confirms that the binary features are indeed more compact than the vector-based features, as was discussed in section 2.2.2.



(a) *Total memory usage.*

(b) *Memory usage per accepted node.*

Figure 4.13: *The increased memory usage when running RGB-D SLAM with different extractor pairs. The binary feature descriptors are equally compact, but more efficient than the SIFT descriptor when considering that more nodes were added.*

5 Conclusions and future work

In this report, we present a complete quadnocular stereo vision system for real time visual localization and mapping of a moving vehicle. Our contributions include hardware and system architecture design, software implementation using open source libraries as well as an evaluation of the system's performance. More specifically, the system computes depth as well as performs SLAM using only the visual information provided by the cameras, all in real time, as was stated in the objective of the thesis.

The objective was furthermore to find a suitable system configuration by evaluating how various system parameters influenced the quality of the output. One such parameter was the camera pair setup. The goal was to take advantage of the established correlation between wider baseline and better depth resolution at long range (and vice versa). We showed that this could be handled by utilizing a multi-baseline camera setup, however at the price of drastically reducing the frame rate of the depth map computation with each extra camera pair. A possible way to balance this trade off is to use only two camera pairs, although the smallest translational error was observed when all three camera pairs contributed to the depth map.

We have furthermore seen that the examined binary feature extraction algorithms, with a few exceptions, are faster and more memory efficient than the vector based without compromising with the robustness. However, the binary feature detector ORB generally generates fewer keypoints than SIFT or SURF which could cause problems for the egomotion estimation, especially when ORB is also used as descriptor. From our experiments we can therefore conclude that combining a vector-based detector with a binary descriptor is the most promising configuration.

Given certain controlled conditions, primarily traveling at low speeds in a feature rich environment, the proposed system is indeed a working proof-of-concept prototype. However, when implemented in a commercial vehicle, no such demands can be made regarding the conditions under which the vehicle operates. The vehicle must be able to move with a decent velocity and the localization system shall not fail if areas with plain texture or repetitive features constitute a large part of the camera's field of view. Furthermore is the achieved accuracy even under optimal conditions probably not precise enough for some purposes, as localization errors up to 50 centimeters were commonly observed. We therefore suggest a set of improvements to increase the robustness and precision of the system which can be used as guidelines for future work in this area.

What first and foremost dictates the maximum speed of the vehicle is if stereo images can be captured simultaneously and if depth maps can be generated at a high rate. If that is not the case, camera motion will lead to corrupt depth maps, incorrect 3D coordinates of the image keypoints and too large movement of keypoints between consecutive frames which will make the egomotion estimation difficult or even impossible. The use of synchronized cameras is a rather simple solution that likely will improve the quality of the resulting depth maps for a moving camera system. It would furthermore be interesting to investigate how parallel computing or utilization of GPU-based methods could accelerate the image processing tasks and thereby increase the frame rate. It might also be worthwhile to investigate for instance the effect of subsampling the point clouds for increased speed.

The use of the same camera both in a short range and long range camera pair was found to be problematic due to conflicting desired camera focal length. An area worth investigating would therefore be possible ways of improving the perspective transform approximation in the depth map merging, or perhaps merge the depth information from separate camera pairs using a completely different method.

Although the accuracy of the mapping and the localization will probably benefit from the measures above or from investing in high-precision hardware such as lenses with less distortion, it is in the end probably necessary to rely on more than visual information to achieve localization which is useful in an autonomous vehicle. We therefore believe that utilizing sensor fusion techniques to combine the visual odometry from stereo vision with other data sources such as wheel sensors, IMUs or satellite positioning systems is the right course of action for future development.

References

[1] E. Ackerman. Google's autonomous car takes to the streets. *IEEE Spectrum* (2010).

[2] A. Alahi, R. Ortiz, and P. Vandergheynst. "Freak: Fast retina keypoint". *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. Ieee. 2012, pp. 510–517.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[4] H. Bay et al. Speeded-up robust features (SURF). *Computer vision and image understanding* **110**.3 (2008), 346–359.

[5] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008, pp. 370–458.

[6] M. Z. Brown, D. Burschka, and G. D. Hager. Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **25**.8 (2003), 993–1008.

[7] M. Calonder et al. "Brief: Binary robust independent elementary features". *Computer Vision–ECCV 2010*. Springer, 2010, pp. 778–792.

[8] S. I.-N. Delhi. When Automakers Invaded a Consumer Electronics Show. *Auto Tech Review* **4**.2 (), 48–51.

[9] M. G. Dissanayake et al. A solution to the simultaneous localization and map building (SLAM) problem. *Robotics and Automation, IEEE Transactions on* **17**.3 (2001), 229–241.

[10] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE* **13**.2 (2006), 99–110.

[11] F. Endres et al. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on* **30**.1 (2014), 177–187.

[12] B. Fan et al. "Learning weighted Hamming distance for binary descriptors". *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 2395–2399.

[13] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM* **24**.6 (1981), 381–395.

[14] G. Grisetti et al. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine, IEEE* **2**.4 (2010), 31–43.

[15] C. Harris and M. Stephens. "A combined corner and edge detector." *Alvey vision conference*. Vol. 15. Manchester, UK. 1988, p. 50.

[16] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **30**.2 (2008), 328–341.

[17] A. Hornung et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots* **34**.3 (2013), 189–206.

[18] B. Jahne. *Practical handbook on image processing for scientific and technical applications*. CRC Press, 2004.

[19] S. B. Kang et al. "A multibaseline stereo system with active illumination and real-time image acquisition". *Computer Vision, 1995. Proceedings., Fifth International Conference on*. IEEE. 1995, pp. 88–93.

[20] K. Khoshelham and S. O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors* **12**.2 (2012), 1437–1454.

[21] R. Kummerle et al. "g 2 o: A general framework for graph optimization". *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3607–3613.

[22] S. Leutenegger, M. Chli, and R. Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints". *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2548–2555.

[23] C.-K. Liang, Y.-C. Peng, and H. Chen. "Rolling shutter distortion correction". *Visual Communications and Image Processing 2005*. International Society for Optics and Photonics. 2005, pp. 59603V–59603V.

[24] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* **60**.2 (2004), 91–110.

[25] K. D. Mankoff and T. A. Russo. The Kinect: A low-cost, high-resolution, short-range 3D camera. *Earth Surface Processes and Landforms* **38**.9 (2013), 926–936.

[26] A. Martinez and E. Fernández. *Learning ROS for robotics programming*. Packt Publishing Ltd, 2013.

[27] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**.10 (2005), 1615–1630.

[28] M. Montemerlo et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". *AAAI/IAAI*. 2002, pp. 593–598.

[29] *OpenCV 2.4.11.0 documentation - Camera Calibration and 3D Reconstruction*. 2015. URL: `http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html` (visited on 05/26/2015).

[30] S. Riisgaard and M. R. Blas. *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*. 2003.

[31] E. Rublee et al. "ORB: an efficient alternative to SIFT or SURF". *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2564–2571.

[32] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of Interest Point Detectors. *International Journal of Computer Vision* **37**.2 (2000), 151–172.

[33] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. "Cengage Learning", 2008, pp. 546–605.

[34] J. Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 573–580.

[35] S. Thrun and J. J. Leonard. "Simultaneous localization and mapping". *Springer handbook of robotics*. Springer, 2008, pp. 871–889.

[36] R. Triebel, P. Pfaff, and W. Burgard. "Multi-level surface maps for outdoor terrain mapping and loop closing". *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE. 2006, pp. 2276–2282.