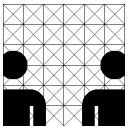
Diplomarbeit
im Studiengang Informatik

Development of a 3D Simultaneous Localization and Mapping Exploration System

am Arbeitsbereich für
Technische Aspekte Multimodaler Systeme,
Universität Hamburg

vorgelegt von
Gregor Michalicek

4. Oktober 2010



Gutachter
Prof. Dr. Jianwei Zhang
Dr. Werner Hansmann

T|A
M|S

Zusammenfassung

Innerhalb der letzten Jahre wurden viele Beispiele für autonome Robotersysteme demonstriert. Einige dieser Systeme profitieren stark davon, dass für sie eine Karte der Umgebung verfügbar ist. Es existieren allerdings auch Anwendungen, für die eine vorherige Erfassung einer Karte nicht möglich ist. In derartigen Situationen muss das autonome System selbständig eine Karte generieren. Diesbezüglich sind viele Beispiele für Kartographierungssysteme, die zweidimensionale Karten herstellen, bekannt, wobei dreidimensionale Karten der Umgebung mächtiger wären und als Grundlage für weitere Anwendungen genutzt werden könnten. Systeme, die autonom die Umgebung erkunden und eine 3D Karte generieren, sind hingegen rar.

Das Ziel dieser Diplomarbeit ist die Entwicklung eines Robotersystems, das in der Lage ist, eine Innenraumumgebung autonom zu erkunden und zu kartographieren. Hierzu wenden wir einen Algorithmus zur *Simultanen Lokalisierung und Kartographierung* (SLAM) an, um eine 3D Karte zu generieren, die zur Planung der Navigation des Roboters genutzt wird, um die Umgebung weiter zu erkunden. Wir realisieren dies als verteiltes System und implementieren weiterhin Programme zur Simulation des Roboters und zur Visualisierung der erfassten Karte. Zum Abschluss evaluieren wir das System als Ganzes und schlagen Ideen für Weiterentwicklungen in diesem Zusammenhang vor.

Abstract

Within recent years many examples for autonomous robot systems were demonstrated. Some of these systems strongly benefit from the availability of a map of their environment, although there exist applications in which a prior acquisition of a map is not feasible. In such situations the autonomous system has to build up the map on its own. In this regard, many examples for such map building systems exist for two dimensional maps, although three dimensional maps of the environment are more powerful and can be used as a basis for further applications. Systems that autonomously explore the environment and build up a 3D map are rare.

The objective of this diploma thesis is the development of a robot system capable of autonomously exploring and mapping an indoor environment. For this we apply a *simultaneous localization and mapping* (SLAM) algorithm to build up a 3D map and use it to plan the navigation of the robot to further explore the environment. We realize this as a distributed system and additionally implement programs to simulate the robot and visualize the acquired map. Finally, we critically evaluate the system as a whole and propose ideas for further developments in this context.

Contents

Chapter 1

Introduction

Within the last years progress in robotics, computer science, performance improvements of computer hardware as a consequence of Moore's law [1], and cheaper and cheaper sensors bring autonomous robotics for several applications within the range of realization. Meanwhile, several excellent examples for autonomous systems exist which are very promising with respect to anticipated future developments. One popular example for such systems are autonomous vehicles as they were demonstrated in the DARPA grand challenge events [2].

In this diploma thesis we focus on another application of autonomous robotics which is, however, equally promising. The objective of this work is the autonomous 3D exploration of indoor environments. This means that we develop a system that lets a robot move through an indoor environment, take 3D scans of its surroundings, and finally build a 3D map of the environment. We build up an integrated system that generates a map in each exploration step and uses this map to determine the future exploration behavior of the robot. Of course, many applications for such a system can be imagined. First of all, the system autonomously generates a map of an indoor environment. Such a map is useful for many purposes, beginning with the planning of the interior accessories of a building and ending with the planning of the jobs for a household robot. Often, it is not feasible to acquire such a map in the run-up of some task that needs it. This can be due to technical reasons or due to the costs of acquiring such a map. We develop a system that offers a cheap way to automatically generate a 3D map of an indoor environment.

The proposed system consists of several components where each component is an own program within a distributed system architecture. The connections between these programs are visualized in figure 1.1. The central part of the system is the control program *RobotController*. This program controls the communication between the other components. It gathers data from some of the components and sends the data together with commands to other components. The second main component is the connection to the robot which is interchangeable with a simulation program for the robot in an indoor environment. The robot controller sends commands to these programs to trigger robot movements or measurements by the robot's sensors. In exchange the robot connector program or the simulation send the scanner data or the robot's odometry data back to the central

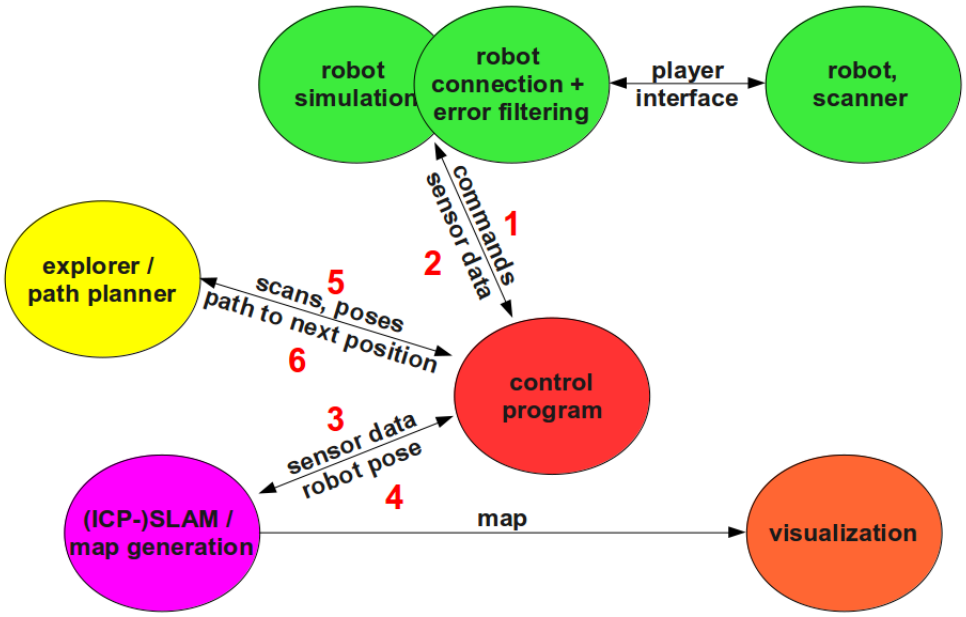


Figure 1.1: Overview of the exploration System. The system incorporates several components that form a distributed system architecture.

control program. The next step in the exploration process is the construction of a 3D map of the environment. For this, the control program sends the data from the robot to another program that applies a so-called Simultaneous Localization and Mapping (SLAM) algorithm on it. Such an algorithm uses the odometry and sensor data to accurately determine the robot pose and build up a map. The program sends this new information back to the central control program. After this, the following step in the exploration process is the determination of the next scan location for the robot. For this, the robot's pose and the map are sent to the exploration program *Explorer*. On the basis of this data the exploration program decides, where the robot should take the next scan. It calculates a route to this position and sends it back to the central control program. Now, the control program has enough information to generate a sequence of commands that moves the robot to the next location. These commands are sent to to robot connection program or to the simulation and this closes the exploration cycle. Finally, we also develop a program *SurfaceProcessor* that visualizes the generated map.

In this thesis we describe the main components of the exploration system, although we start with a general introduction to the SLAM problem in chapter 2. Here, we discuss several approaches to solve this problem and select one of these approaches for the SLAM component of our system. This is an approach on the basis of the Iterative Closest Point (ICP) algorithm. We continue in chapter 3 by describing the physical component of the system and the *RobotConnector* program. Due to the fact that we need very high quality data as a basis for the exploration algorithm, one important aspect of the *RobotConnector* program is an outlier filtering for the scan data. Chapter 4 then discusses

the details of the ICP SLAM algorithm and its implementation. The most important component of the system, the exploration program, is then described in chapter 5. To develop this component we put together several low-level algorithms to form a high-level approach to the exploration problem. The basis for the development of the simulation program *RobotSim* is then given in chapter 6. The discussion of the simulation program mainly covers an overview of the applied movement and sensor models. Outside of the exploration system a visualization program for the map is developed in chapter 7. This program visualizes the map in two ways. First of all the map is simply visualized by a representation of the sampled scan points. The other visualization technique is more sophisticated. Here, we reconstruct the surface of the environment by the Ball Pivoting algorithm. The chapter mainly discusses this surface reconstruction algorithm. After presenting the visualization, we show and discuss results for the exploration of a room in chapter 8. Finally, we summarize the work and give an outlook on possible consecutive developments in chapter 9.

Throughout the work we assume that the reader has a background in computer science and mathematics. Therefore, we make use of common technical terms from these disciplines without explicitly introducing them. Detailed discussions on such terms can be found in common text books.

Chapter 2

Simultaneous Localization and Mapping (SLAM)

For an autonomous mobile robot that has to interact with its environment it is advantageous to know its pose with respect to the surroundings. This implies that it has to have a map of the environment. In general, there are several ways a robot can build up knowledge about its pose. We present three different scenarios in which we assume that the robot has sensors to scan the local environment. Laser range finders, sonar sensors, radar sensors or cameras are instances of such sensors. We also assume that the robot has internal sensors to measure its odometry.

The first scenario is a pure mapping scenario in which the robot has explicit sensors to directly estimate its location and the direction in which it is oriented. This can be a GPS sensor for the estimation of the location and a compass for the orientation measurement. There is no previously recorded map available but the robot can use its sensors to estimate the local environment, combine this information with pose measurements and build up a map by collecting and fusing data from different locations.

In the second scenario the robot has to do pure localization. It has no sensors to directly estimate its pose but there is a previously acquired map available. The robot can then use its local environment scanning sensors, compare the measured data with the map and thereby estimate its pose.

Lastly, there is the Simultaneous Localization and Mapping (SLAM) scenario in which the robot neither has a previously acquired map nor sensors to directly measure its pose. In this scenario the robot scans the environment, moves to another location, scans the environment again and fuses the data with the first scan. To do this the robot uses the odometry sensors to get a first guess of its pose. Unfortunately no sensor measurement has unlimited accuracy. In the case of SLAM the errors of all odometry measurements sum up and after a few steps a robot pose that is purely acquired with the odometry data is unusable. The robot therefore uses the environment scans to correct the odometry pose guesses by matching the scans to each other. In this scenario the map is also represented by the fused environment scans.

There are many applications for autonomous mobile robots in each of the three scenarios.

However, we focus on SLAM. Examples for this scenario are indoor scenarios in which it is not feasible to previously acquire a map. Underwater scenarios or robots on other planets are also instances of the SLAM class of scenarios.

In this chapter a general overview of the SLAM problem is presented that follows the detailed discussion by Thrun et al. [3]. We give a definition of the problem in section 2.1 and a presentation of the challenges for SLAM algorithms in section 2.2. An overview of algorithms that approach the SLAM problem is then given in sections 2.3 to 2.6. Finally we choose one of these algorithms for the exploration system in section 2.7 and conclude the chapter in 2.8.

2.1 The SLAM problem

When doing Simultaneous Localization and Mapping the central problem is the inaccuracy of scanner data. There is a time-independent inaccuracy in the local environment scans but the inaccuracy of the odometry measurements is more crucial. The limited accuracy of these measurements leads to a growing error in pose estimations depending on the actions the robot has done up to some point in time.

The SLAM problem can easily be expressed in a probability distribution

$$p(x_{t_0:t}, m | z_{t_0:t}, u_{t_0:t}) \tag{2.1}$$

where $x_{t_0:t}$ are the poses from time t_0 to time t, m is the map and $z_{t_0:t}$ and $u_{t_0:t}$ are the actions (odometry measurements) and environment measurements the robot has made between t_0 and t. In this expression we assume a fixed and known pose for the robot at t_0. Algorithms that try to handle the SLAM problem now have a well defined task. They have to find the poses $x_{t_0:t}$ and the map m that maximize expression 2.1.

A formulation of the SLAM problem that is based on expression 2.1 is known as the *Full SLAM* problem. Another probability distribution

$$p(x_t, m | z_{t_0:t}, u_{t_0:t}) \tag{2.2}$$

is used to formulate the *Online SLAM* problem. In the Online SLAM problem, measurements in the future ($t > t_1$) do not affect the probability for a pose at time t_1.

The two SLAM problems have different applications. If the goal is to get an accurate map after the robot has followed some path, the Full SLAM problem should be solved. On the other hand, if the map data has to be used while the robot is in action, it is reasonable to only solve the Online SLAM problem as an approximation to the Full SLAM problem. Of course, Full SLAM is computationally more demanding than Online SLAM.

In this diploma thesis we use the acquired map data in action as a basis for an exploration algorithm. Therefore we approach the Online SLAM problem.

Mathematically, it is easy to calculate the Online SLAM distribution 2.2. This is done by integrating out past poses from the Full SLAM problem. An expression for this is given by

$$p(x_t, m | z_{t_0:t}, u_{t_0:t}) = \int \cdots \int p(x_{t_0:t}, m | z_{t_0:t}, u_{t_0:t}) dx_{t_0} \dots dx_t \tag{2.3}$$

where we assume discrete time steps from t_0 to t. To get a practical approach for the Online SLAM problem we further reformulate equation 2.3 and get

$$p(x_t, m|z_{t_0:t}, u_{t_0:t}) = p(z_t|x_t, m) \cdot \int p(x_t|x_{t-1}, u_t)p(x_{t-1}, m|z_{t_0:t-1}, u_{t_0:t-1})dx_{t-1}. \quad (2.4)$$

In this stepwise approach we calculate the posterior for x_t on the basis of the distribution for x_{t-1}. This is computationally a tremendous improvement with respect to equation 2.3. In expression 2.4 we only have one integration at each discrete time step in comparison to a number of integrations that grows linearly with the number of past time steps. Assuming constant time for each integration, expression 2.4 may in principle lead to algorithms that also calculate the position at each step in constant time.

Although equation 2.4 is a real step forward, it is not yet the final step in making the SLAM problem computationally solvable. So far, we did not yet say anything about the representation of the different quantities in it. This is a problem for which every SLAM algorithm has its own solution and we address it when we discuss each algorithm in detail in later sections of this chapter.

2.2 Challenges for SLAM algorithms

In order to further specify the problem to solve, we mention some of the possible challenges in localization and mapping and determine which challenges we have to deal with.

2.2.1 The localization problem

In the pure localization scenario there are three different types of problems which we may have to deal with:

- **Position tracking**. In this scenario the initial robot pose is known and the robot has to keep track of its location and orientation at each time step. The odometry data connects each step with the next one.
- **Global localization**. In global localization a map is given, but the initial pose is unknown. This leads to pose posteriors that are not unimodal since there are in general several poses that fit to the initial environment measurements. In this scenario different steps are also connected by the robot actions which manifest in odometry data.
- **The kidnapped robot problem**. The kidnapped robot problem finally breaks the odometry connections between each step. In this scenario the robot has to incorporate the possibility that it is moved from on pose to another without noticing it by its odometry measurements.

In SLAM we clearly have to approach the *Position tracking* problem. The initial pose in SLAM is the origin of the coordinate system that represents all possible locations and orientations.

2.2.2 The environment

We classify the environment with respect to two aspects. The first aspect is the static or dynamic behavior of the environment:

- **Static environment**. In static environments nothing besides the robot moves. This means that the environment at time t_1 is equal to the environment at any other point in time.
- **Dynamic environment**. Dynamic environments change over time. This could be due to moving people or other aspects of the surroundings. Dynamic environments make the mapping more difficult. Since the map is acquired step by step it has to be taken into account that scans at some point in time show dynamic aspects of the environment which are not present in other scans.

The SLAM problem can be solved for both, static and dynamic environments. In the context of this diploma thesis we deal with static environments.

The second aspect considers the flatness of the environment.

- **Flat environment**. In flat environments the robot has three degrees of freedom, such that its pose can be represented by a vector

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \theta \end{pmatrix} \tag{2.5}$$

  where x_1 and x_2 represent the location and θ the orientation.

- **Uneven environment**. In uneven environments the robot has six degrees of freedom. In such scenarios we need three coordinates for the location and another three coordinates for the orientation of the robot. Therefore, the SLAM problem becomes more complex, although its fundamental aspects are not changed.

In this diploma thesis we consider indoor environments which are flat. This is mostly predetermined by the laser range finder which is used to scan the surroundings. This laser scanner only has a very limited range and is therefore not adequate for the use in uneven outdoor environments.

2.3 SLAM with an Extended Kalman Filter

To handle equation 2.4 one method is to use an Extended Kalman Filter (EKF). In this section we show how this can be done by first describing the Kalman and Extended Kalman Filters and then the application of the EKF to the SLAM problem. This yields to the EKF SLAM algorithm.

A Kalman filter is a special Bayes Filter that makes the so called *Gaussian Noise Assumption* for the quantities taken under observation. This means that the belief for each quantity is represented by a Gaussian. Now assume that some process transforms a quantity, e.g. if a robot moves, its pose is transformed. The arising question is how the corresponding belief is transformed. The Kalman filter is only applicable to processes that *linearly* transform the quantities. In this case, a Gaussian belief is transformed into another Gaussian belief.

To be more precise, if x_{t-1} is the state at time $t-1$, and the action u_t transforms this state like

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2.6}$$

where A_t and B_t are matrices and ϵ_t is a Gaussian random vector with zero mean and covariance R_t, the probability distribution for x_t is given by

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{1/2} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\}. \tag{2.7}$$

Unfortunately, the transformations due to robot motion are not linear. Instead, the robot pose is transformed by

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{2.8}$$

where g is a nonlinear function. Therefore, the Kalman Filter approach for calculating and representing robot poses as Gaussians has to be modified. This is done by linearizing the transformations with the first two terms of a Taylor expansion of g at the mean of the original Gaussian for x_{t-1}.

So far, we have only sketched the idea for the Kalman Filter algorithm and its extension to nonlinear transformations. The final algorithm also incorporates the measurements z_t at time t and its posterior $p(z_t|x_t)$. Of course, the original Kalman Filter also makes the linearity assumption

$$z_t = C_t x_t + \delta_t \tag{2.9}$$

where C_t is a matrix and δ_t is a Gaussian random vector. In the nonlinear extension, the dependence between z_t and x_t is given by

$$z_t = h(x_t) + \delta_t \tag{2.10}$$

where h is a nonlinear function that is linearized by using a Taylor expansion and δ_t has the same meaning as in equation 2.9. Putting all of this together, we obtain the final EKF approach as it is given in algorithm 1. The algorithm's input is the state at $t-1$, represented by the mean μ_{t-1} and covariance Σ_{t-1} of the Gaussian belief of x_{t-1}. The action u_t and the measurements z_t are also given to the algorithm as input. The output of the algorithm is the belief of x_t in terms of its mean μ_t and covariance Σ_t.

Algorithm 1 Extended Kalman Filter

1: **procedure** EXTENDEDKALMANFILTER(μ_{t-1},Σ_{t-t},u_t,z_t)
2: $\hat{\mu}_t = g(u_t, \mu_{t-1})$
3: $\hat{\Sigma}_t = G_t \Sigma_{t-t} G_t^T + R_t$
4: $K_t = \hat{\Sigma}_t H_t^T (H_t \hat{\Sigma}_t H_t^T + Q_t)^{-1}$
5: $\mu_t = \hat{\mu}_t + K_t(z_t - h(\hat{\mu}_t))$
6: $\Sigma_t = (I - K_t H_t)\hat{\Sigma}_t$
7: **return** μ_t , Σ_t
8: **end procedure**

In detail, the EKF algorithm first calculates the predicted belief $p(x_t|x_{t-1}, u_t)$ and represents it by its mean $\hat{\mu}_t$ and covariance $\hat{\Sigma}_t$. Due to the linearization of g the Jacobian $G_t(\mu_{t-1}, u_t)$ of this transformation function is needed in this calculation. The random vector ϵ_t is incorporated by its covariance R_t. Consecutively, the so called *Kalman gain* K_t is computed in terms of the Jacobian H_t of h and the covariance Q_t of δ_t. This quantity represents the effect of the measurements z_t on the belief for x_t. Finally, the belief $p(x_t|x_{t-1}, u_t, z_t)$ is calculated and returned by its mean μ_t and covariance Σ_t.

Assuming that the state vector x has a rather small dimension n, the computationally most demanding step in this algorithm is the matrix inversion for the calculation of the Kalman gain. The matrix to be inverted has a size of $k \times k$, where k is the dimension of the measurement vector z_t. Since todays best matrix inversion algorithms have a complexity of $\mathcal{O}(k^{2.4})$ [3], it is worthwhile to reduce the dimension of z_t by applying feature extraction methods on the raw measurements and use the measured data of these features for the construction of z_t. All in all the computational complexity of this algorithm is $\mathcal{O}(k^{2.4} + n^2)$.

After illustrating the Extended Kalman Filter algorithm, we take the next step and apply this method to the SLAM problem. This can be done in a rather straightforward way. The main point to realize here is that the state at time t does not only incorporate the robot pose x_t, but also a representation of the map m. In the EKF SLAM algorithm the map is represented by a set of point landmarks and therefore the state vector has to be extended with this information. It is then given by

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix}. \tag{2.11}$$

Of course, this new definition of the state vector increases its dimension enormously. Hence, to keep the EKF SLAM algorithm fast, it is required to use as few landmarks as possible to represent the map. To implement the EKF SLAM algorithm it is also required to handle unknown correspondences between landmarks in the map and measured landmarks. However, a detailed discussion of all the challenges of a realization of this algorithm is beyond the scope of the SLAM overview in this work.

2.4 SLAM with a Sparse Extended Information Filter

In this section we sketch the Information Filter and the Extended Information Filter (EIF). Just like the Extended Kalman Filter the EIF linearizes nonlinear transformations. After presenting the EIF algorithm, we proceed and illustrate the application of this approach to the SLAM problem. This yields the Sparse Extended Information Filter (SEIF) SLAM approach.

In principle, an Information Filter makes the same approach as a Kalman Filter. This Bayes Filter also makes the Gaussian Noise Assumption. The difference between the two Filters is the representation of the different quantities, which leads to different computational complexities. Although the difference is not huge, it is interesting to compare the two approaches to see their strengths and weaknesses with respect to each other.

The central quantities used in the Information Filter to represent Gaussians are the *information matrix* $\Omega = \Sigma^{-1}$ and the *information vector* $\xi = \Sigma^{-1}\mu$. A representation by these quantities is called a *canonical parameterization* in contrast to the *moments parameterization* used by the Kalman Filter. It is easy to see that both representations are equivalent, since they can be expressed in terms of the other. Therefore, it is also easy to formulate a Bayes Filter in terms of the new quantities Ω and ξ. We do this once again for the extended version and keep the notations of section 2.3. The result is algorithm 2.

Algorithm 2 Extended Information Filter

1: **procedure** EXTENDEDKALMANFILTER(ξ_{t-1},Ω_{t-t},u_t,z_t)
2: $\quad\Sigma_{t-1} = \Omega_{t-1}^{-1}$
3: $\quad\mu_{t-1} = \Sigma_{t-1}\xi_{t-1}$
4: $\quad\hat{\Omega}_t = (G_t\Sigma_{t-1}G_t^T + R_t)^{-1}$
5: $\quad\hat{\xi}_t = \hat{\Omega}_t g(u_t, \mu_{t-1})$
6: $\quad\hat{\mu}_t = g(u_t, \mu_{t-1})$
7: $\quad\Omega_t = \hat{\Omega}_t + H_t^T Q_t^{-1} H_t$
8: $\quad\xi_t = \hat{\xi}_t H_t^T Q_t^{-1}\left[z_t - h(\hat{\mu}_t) + H_t\hat{\mu}_t\right]$
9: $\quad$**return** ξ_t , Ω_t
10: **end procedure**

Assuming that the Gaussian random vector δ_t from equations 2.9 and 2.10 is represented by its information matrix Q_t^{-1}, the only matrices that are inverted by this algorithm have a dimension of $n \times n$. This yields a computational complexity of $\mathcal{O}(n^{2.4} + k^2)$. Comparing this with the computational complexity of the Kalman Filter, shows that both have advantages over the other algorithm depending on the ratio between n and k.

Of course, the EIF can be applied to the SLAM problem just like the EKF by using the state vector 2.11. Examining the information matrix then shows that it naturally is nearly sparse. There are some elements in it that have far larger values than most of the

other elements. This fact can be used to greatly enhance the efficiency of this approach. In an approximation the small elements of the information matrix can be set to zero, such that the matrix inversions can be done more efficiently. This leads to the Sparse Extended Information Filter (SEIF) SLAM approach. If done in a smart way, the sparsification of the information matrix yields a SLAM algorithm that needs constant time for each iteration. However, the sparsification approximation leads to inaccurate results.

2.5 The FastSLAM approach

The EKF SLAM and SEIF SLAM algorithms both make use of the Gaussian Noise Assumption and approximate the robot movement by linear transformations. The so called FastSLAM [4] approach does not use these approximations. Instead, it uses a completely different way to describe the posterior for the robot pose and the transitions. While the preceding algorithms use Kalman and Information Filters to describe the movement and the poses of the robot, FastSLAM uses a so called *Particle Filter*. This is also a realization of a Bayes Filter. However, instead of using a Gaussian to describe probability distributions, the Particle Filter describes these Distributions by a set of representative samples. Of course, this is a description which is compatible with every error and odometry model for the robot movement. Nonlinear movement of the robot results in a nonlinear transformation of the samples and it is also no problem to apply non-Gaussian error models to the robot movement.

Algorithm 3 Particle Filter

1: **procedure** ParticleFilter(χ_{t-1}, u_t, z_t)
2: $\hat{\chi}_t = \chi_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: sample $x_t^m \sim p(x_t|u_t, x_{t-1}^m)$
5: $w_t^m = p(z_t|x_t^m)$
6: $\hat{\chi}_t = \hat{\chi}_t + \langle x_t^m, w_t^m \rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9: draw pair i from $\hat{\chi}_t$ with probability $\propto w_t^i$
10: add x_t^i to χ_t
11: **end for**
12: **return** χ_t
13: **end procedure**

Algorithm 3 shows the Particle Filter in a pseudo code notation. In this algorithm χ_t is the set of samples at time t while the action u_t and measurement z_t have the same meaning as in the other algorithms. M is the number of samples (or particles) used. The particle Filter consists of two steps. In the first step the old poses are transformed according to $p(x_t|u_t, x_{t-1}^m)$ and weights $w_t^m = p(z_t|x_t^m)$ are associated with each of the new poses. In

the second step we resample the posterior of x_t. This is a critical step since we get rid of bad samples.

The particles in the FastSLAM approach do not only contain the robot pose. They also contain a representation of the map so that $p(z_t|x_t^m)$ can be calculated. This is a very different approach to the preceding algorithms where we only had a single representation of the map. In the FastSLAM algorithm every particle has its own map, which can be represented in different ways. One common approach is to represent the map as a set of features and to use Kalman Filters for each of these features. Another approach is the representation by an *Occupancy Grid Map*. This map representation is not feature based, instead it subdivides the space into a regular grid and saves the probability of something being in a grid cell for each cell. Although this representation takes a lot of space, it has some advantages. The most compelling one is the fact that so called *Negative Information* is retained. This means that information from the robot's sensors which indicate that some area in the space is not occupied by anything is also integrated into the map.

From a performance standpoint the FastSLAM algorithm offers very desirable options. Its runtime is proportional to the number of particles and the algorithm can easily be parallelized over the particles. This means that accuracy can be traded against runtime and it is even possible to implement FastSLAM as an anytime algorithm. If FastSLAM is implemented including the Occupancy Grid Map option, accuracy can also be traded against runtime by varying the resolution of the real-space grid. All in all there are many options to adapt the algorithm to particular requirements.

2.6 SLAM with the Iterative Closest Point algorithm

A SLAM solution can also be realized by applying the *Iterative Closest Point* (ICP) algorithm [5]. This approach does not use an error model for the movement to keep track of the robot motion.

In ICP SLAM the map m is represented by a sequence of robot poses with associated point clouds of measured points on surfaces within the environment. For every step of the robot, the algorithm makes an initial estimate for its pose x_t. The algorithm then compares the current measurements z_t with the map and minimizes

$$E(R,t) = \sum_{i}^{N_m} \sum_{j}^{N_d} w_{i,j} \left\| m_i - Rd_j + t \right\|^2 \tag{2.12}$$

with respect to to robot pose x_t. In this expression R is a rotation matrix and t is a translation vector. Both, R and t, are applied to the robot pose x_t or, as a consequence, directly to the measurements z_t. The current measurements are represented as a point cloud with points d_j and the map is represented as a point cloud with points m_i. $w_{i,j}$ is a weight function that indicates correspondence between points in $\{m_i\}$ and $\{d_j\}$. Equation 2.12 is an expression for the sum of the quadratic distances between corresponding points in the two sets. ICP SLAM iteratively minimizes equation 2.12 and thereby corrects the initial

estimate for x_t. It terminates when some convergence criteria are met.

The advantage of this approach is that no complicated error model for the robot movement has to be developed. A feature extraction on the measurements is also not required.

2.7 Choosing a SLAM algorithm

In this work we develop an integrated exploration system that uses data from a 3D laser range finder. The system uses a SLAM algorithm to keep track of the robot motion and build up a consistent map that is used as a basis for the exploration.

Although there are works like [6] that use feature extraction techniques in a similar context, we notice that features like lines, corners and so on depend a lot on the environment the robot is situated in. We want to keep our approach as general as possible, such that feature extraction is not an option. The EFK SLAM algorithm as well as SEIF SLAM require feature extraction. Therefore, we will not choose one of these two approaches for the SLAM solution of the exploration system.

The remaining options are the FastSLAM algorithm in combination with Occupancy Grid Mapping and the ICP SLAM algorithm. Both algorithms have advantages. For our prototype system we choose the ICP SLAM option since it is already proven to work for 3D SLAM, even with 6 degrees of freedom [5]. Another advantage of this approach is the fact that it does not rely on a sophisticated error model for the robot movement and the scanner measurements. A realization of an exploration system on the basis of the other option would also be interesting, but the feasibility of such a solution would have to be proven first. This would shift the focus of this work into another direction, although future derived works are encouraged to explore this option.

2.8 Summary

In this chapter we gave an overview over the Simultaneous Localization and Mapping problem and its solutions. The problem was defined in section 2.1 and we gave an overview of the challenges associated with this problem in section 2.2.

Next, we sketched some general approaches to the Online SLAM problem in sections 2.3 to 2.6. We identified advantages and disadvantages of each algorithm. The Full SLAM problem was not covered since it is difficult to apply it to the objective we address in this work. Note that we excluded most of the implementation details and several other important aspects that have to be discussed if one of these algorithms has to be implemented. This applies to the problem of finding corresponding features as well as to the problem of closing loops in cyclic environments. A detailed discussion of these aspects is beyond the scope of this work.

Finally, we chose the ICP SLAM algorithm in 2.7 as basis for the exploration system which is developed in this work. An explanation for this choice was also given.

Chapter 3

The Robot Platform

The physical entity of the exploration system designed in this work is a Pioneer 2 DX robot by Mobile Robots Inc. [7]. A rotation platform for a 2D laser range finder is constructed and mounted on top of the Pioneer in a preceding work [8]. By registration of scans from the 2D laser range finder into a three dimensional, local coordinate system, the rotation platform effectively creates a 3D laser range finder that provides the measurements used to map the environment.

In this chapter we discuss the physical properties of the robot platform and the laser range finder. The low-level control of these two components is also described in this topic, as well as the low-level preprocessing of the acquired data.

Section 3.1 describes the Pioneer robot, while section 3.2 introduces the range finder rotation platform and section 3.3 shows the control for both components. We end the chapter by describing the preprocessing of the scanner data in section 3.4 and give a summary in 3.5.

3.1 The Pioneer 2 DX robot

The physical basis for the exploration system is the Pioneer 2 DX mobile robot shown in figure 3.1. The robot's locomotion is based on two independently driven wheels in the front and a single unpowered castor wheel in the rear. This is a typical construction for an indoor robot, although [9] also shows other locomotion principles in this context that have differing properties.

The construction of the robot incorporates several sources for inaccurate movement. For example, the castor wheel has some disadvantages if certain movements have to be performed. If the robot rotates and then moves forward, the castor wheel is misaligned in the beginning of the forward movement. This leads to a small swivel and a slightly wrong direction of the forward movement. There are also other systematic errors that make the locomotion of the robot inaccurate. First of all, slightly differing wheel radii lead to a curvature if the robot should straightly move forward. The robot also makes a small rotation at the end of a forward movement. On top of these internal error sources,

Figure 3.1: The Pioneer 2 DX robot. The original robot is slightly modified by extending the platform on its top. The picture also shows the sonar sensors mounted on the robot's front. (Image from [8])

the movement depends a lot on the structure of the ground.

Practical experiences with the robot also revealed that the odometry information obtained by internal sensors is also inaccurate. The error sources in the locomotion actuators, the interaction with the floor, and the errors in the odometry sensors lead to difficulties when a good error model for the robot movement is to be designed. However, our choice to use the ICP SLAM algorithm implies that we don't need such a model.

Lastly, we note that one of the modifications done to the robot is the integration of a second accumulator. This enables the robot to be under autonomous operation for about 24 hours. Therefore, the robot is able to do long-running explorations of the environment.

3.2 A rotation platform for a 2D laser range finder

The rotation platform for the laser range finder and its control was constructed and built in the work of Peter Breuer [8]. The goal of developing this platform was to get an effective 3D laser range finder for the cost of a 2D range finder. In this section we give an overview of the construction of the rotation platform.

The platform mainly consists of the laser range finder itself, a stepper motor, the mechanical build-up and a microcontroller board which is used to control stepper motor and range finder. Figure 3.2 illustrates the construction in detail.

The central component of the rotation platform is the Hokuyo URG-04LX 2D laser

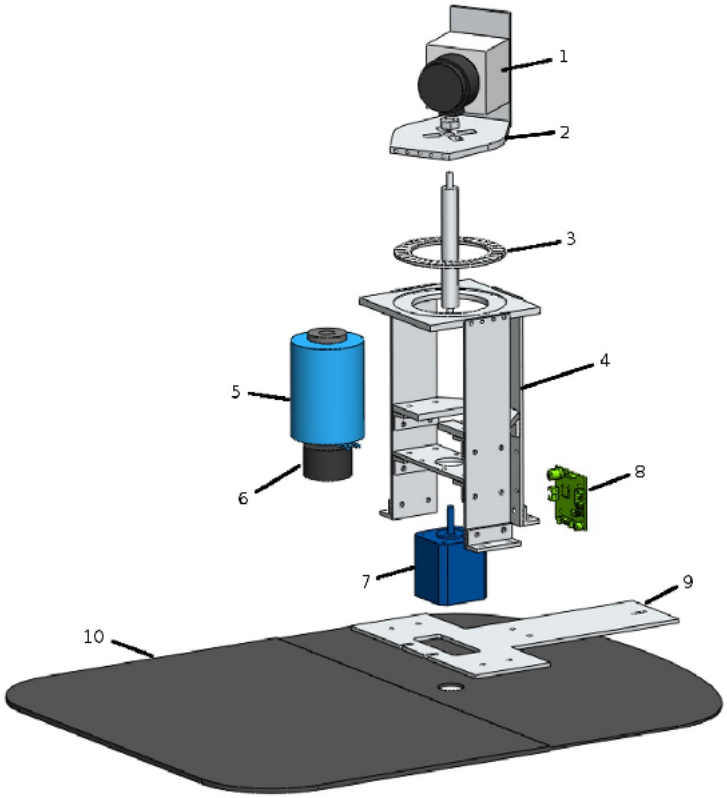


Figure 3.2: Illustration of the rotation platform. The components of the rotation platform are **1** the laser range finder, **3** a needle bearing, **5** a slip ring, **6** a coupling, **7** the stepper motor, and **8** the control for the stepper motor. Component **10** is the plate on top of the robot and components **2**, **4**, and **9** are aluminium parts that hold the construction. (Image from [8])

Figure 3.3: The Hokuyo URG-04LX 2D laser range finder. (Image from [8])

range finder [10], as shown in figure 3.3. The range finder scans the distances in an angle of 240 degrees, where the resolution is about 0.35 degrees. For such a scan step the range finder needs 66.7 ms and another 33.3 ms to complete the rotation to be in the starting position for the next scan. Without any break the range finder performs one scan after the other.

The maximal scan range of the URG-04LX is 5600 mm. Therefore, it is clear that this sensor is only applicable to indoor environments. The accuracy of the laser range finder is stated to be 10 mm if the scan point is nearer than 1 m and 1% of the measured distance if the point is further away. However, there are additional systematic errors in the measured distance. Some surfaces disturb the measurements. On the one hand highly reflective surfaces lead to measured distances that are smaller than the real distance, on the other hand highly absorbing distances do not reflect enough of the laser beam to make valid measurements possible. Additionally to these surface effects, there are absolutely random measurements.

There are different options to construct a mechanism that effectively transforms a 2D range finder into the 3D counterpart by registering the 2D scans into a 3D coordinate system. Breuer constructed a mechanism with a rotation axis in up-down orientation as it is shown in figure 3.4. The resulting angular scan point density is also shown in the figure. The construction has two main advantages over other possible realizations of such a 3D sensor, where the first advantage is regarded to the points missing due to the 2D range finder's limited scan angle. There are only points missing below the construction. Since the robot is located at that position, this is no valuable information about the environment at all. The second advantage is the angle needed to scan the whole space around the robot. The construction only needs a 180° rotation to do this.

Due to the limited scan speed of the URG-04LX the whole construction also has performance limitations. Typically, the 180° rotation needed to scan the whole space is subdivided into 400, 200, 100, or 50 scan planes. Since the URG-04LX needs 100 ms for a single scan plane, capturing the whole space takes up to 40 s. There is also some preprocessing like error filtering taking place on the microcontroller board which consumes a comparable amount of time. Lastly, the transmission of the data from the microcontroller

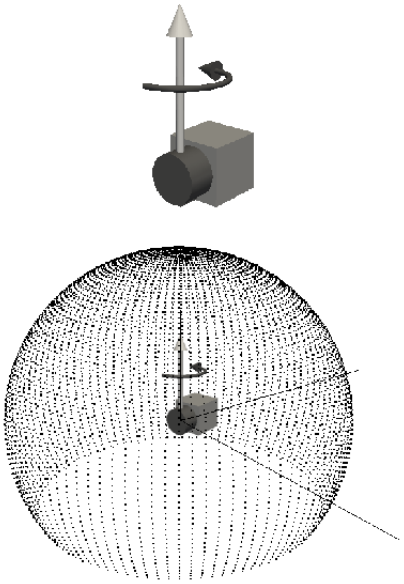


Figure 3.4: Rotation axis and resulting angular scan point density. (Image from [8])

board to a notebook also consumes a lot of time.

The performance considerations of this basic capturing mechanism yields a limited type of operation for the exploration of the environment. The capturing speed does not allow a continuous movement of the robot. Therefore, we will use a so-called *scan-stop strategy* in this context. This is a strategy in which the robot scans the environment while it is standing on a certain point. It then decides where to move from there, then moves there, and finally makes another scan at the new position. This is repeated until the exploration is finished.

3.3 Robot and rotation platform control

In this section we do not describe the control of the rotation platform on the microcontroller board. Instead, we put the focus on the program *RobotConnector* that connects to the robot and to the control program for the rotation platform.

The connection to the robot is provided by the Player/Stage interface [11]. This interface defines methods to set the forward movement speed and the angular velocity of the robot. Methods to read out the data from the odometry sensors are also provided, as well as methods to read out the data from the sonar sensors. Since we follow a scan-stop strategy, we only need to define very basic movement methods. These are a method to rotate the robot by a given angle and a method to move the robot forward by a given distance.

We realize these methods by setting the rotation speed or the forward movement speed to a fixed value. Next we keep track of the data from the odometry sensors and as soon as

the odometry indicates that the angle or the distance is reached, we set the corresponding velocity to zero. In practice it turns out that the robot reacts rather slowly and that the data from the odometry sensors is inaccurate and sometimes erroneous. This makes the control of the robot, in the way we do it, difficult. However, the RobotConnector program is realized in a way that overcomes most of these problems. After moving the robot, the program acquires the odometry data from the robot, compares it with the starting position, and calculates and returns the traveled distance or the rotation angle.

The connection to the rotation platform is realized in a different way. The notebook is connected to the microcontroller board by a serial connection via a USB to serial converter. We open the communication channel to the board over a given port. The program running on the microcontroller board offers us the possibility to set the number of scan steps in the rotation, to start a scan, to acquire the current status of the rotation platform and to acquire the data from the last full space scan. Of course, it is easy to realize a corresponding method on the notebook that sets the scan resolution, triggers a full scan and collects the data from this scan. We will not describe this method in detail.

The RobotConnector program runs on its own and connects itself to the central RobotController program which is responsible for the high-level control of the exploration process. The connection between these two programs is realized within a distributed software architecture. Therefore, the two programs could be located on totally different computers. While the RobotConnector program is realized in C++ and uses *Boost Asio* [12] to establish the connection, the RobotController is a Java program that uses the Java standard library to accomplish this task.

3.4 Outlier filtering in scan data

One important function of the RobotConnector program is the filtering of outliers out of the scan data. These are scan points that are obviously not located on surfaces. To accomplish this task we make a straightforward approach.

The basis for our approach is an approximation to the local surface around a scan point. We assume that the surface is locally flat, at least in one direction. If this is the case, each 3 neighboring scan points that are on a line in the scan mesh are located on a line in 3D coordinates. Figure 3.5 shows this situation in detail. The figure shows a situation with one outlier. Another point is highlighted and the lines on which the corresponding sampled 3D point lies are shown.

Of course, 3 neighboring points will never exactly lie on a line. Therefore, we introduce a tolerance that depends on the distance of the sampled points to the robot. We assume that three points lie on a line if

$$\epsilon > \left| \frac{a+b}{2} - c \right| \tag{3.1}$$

is fulfilled. Here a and b are the distances between the scanner and the end points of the line while c is the distance between the scanner and the center point. The tolerance is

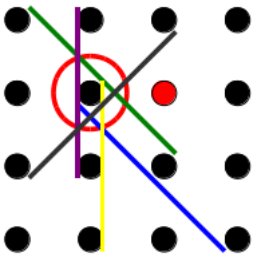


Figure 3.5: Methodology for the outlier filtering. The figure shows points on the angular scan mesh and it is assumed that the red point is an outlier. One of the other points is highlighted and the possible lines on which the corresponding sampled 3D point lies are shown.

defined by $\epsilon = 0.01 \text{ m} + 0.015 \cdot c$. Note that the tolerance is not equivalent to the distance dependent accuracy of the laser range finder, but the difference is not large.

The RobotConnector program checks for each 3 neighboring points that lie on a line in the angular scan mesh if they fulfill equation 3.1. If this is the case, all three points are accepted to lie on a surface. Each sampled 3D point that lies on no line is filtered out of the data set. The complexity of this algorithm is $\mathcal{O}(n)$, where n is the number of sampled points, although, in comparison to the scanning process, the runtime of the outlier filtering is negligible.

Figure 3.6 shows a sample result of the outlier filtering algorithm. Note that the algorithm appears to be very effective. In the example all outliers are filtered out. To demonstrate how many outliers are filtered out by the algorithm we also present total numbers: If a full scan with 400 scan steps is performed, theoretically there should be 272800 sampled points. But there are two filtering steps that reduce this number. The first step filters erroneous transmissions from the laser range finder to the microcontroller board. In this step whole scan planes are taken out of the set of sampled points. This reduces the number of valid points to about 200000. The other filtering step is the outlier filtering which further reduces the number of valid points to 180000. So, about 10 percent of the sampled points are outliers and all in all about one third of the scanned points are invalid.

3.5 Summary

In this chapter we described the physical component of the exploration system. This is the Pioneer 2 DX robot which was introduced in section 3.1 together with a rotating 2D laser scanner shown in section 3.2. We discussed the properties of the robot locomotion and named some sources for inaccurate behavior. We also discussed the rotation platform which effectively transformed the 2D laser range finder into its 3D counterpart in detail. We motivated the choice of the rotation axis and noted that we have to use a scan-stop strategy for the exploration system.

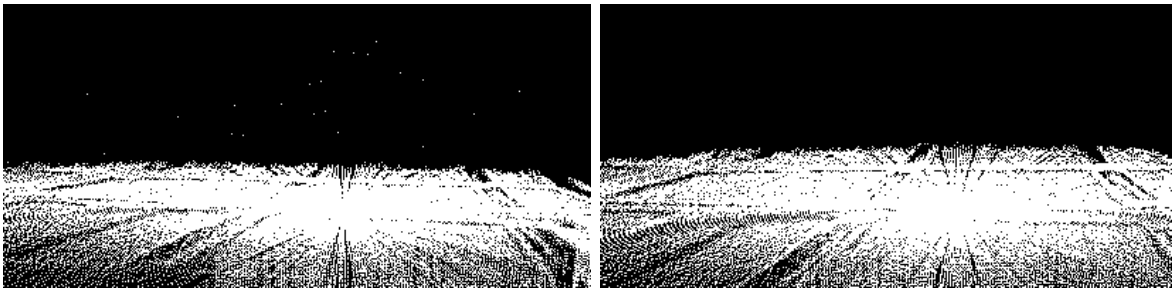


Figure 3.6: Result of the outlier filtering. The images show the sampled points above the ceiling of a room. Left: result including the outliers. Right: result after applying the outlier filtering.

We continued by describing the program that connects the robot and the rotation platform to the rest of the system in section 3.3. The role of this program in the system was defined and we gave a short sketch of the realization. Lastly, we discussed the outlier filtering of the scan data in section 3.4. In this section we used a local approximation to a surface to define a very effective filtering algorithm. Results of an application of this algorithm were also shown.

Chapter 4

Using the ICP Algorithm for 3D SLAM with 3DoF

The choice of the SLAM algorithm for the autonomous exploration system was made in section 2.7. We chose the ICP SLAM approach since, for our purposes, it is a proven concept and the obstacles associated with it are limited. The other promising approach was the FastSLAM algorithm in combination with Occupancy Grid Mapping, although for this algorithm it is difficult to estimate the computational requirements for the case of 3D SLAM. The number of particles as well as the required resolution of the occupancy grid map are difficult to estimate in the run-up to the algorithm implementation.

We sketched the idea of the ICP SLAM approach in section 2.6. In this chapter we make a more sophisticated and detailed approach in discussing ICP SLAM. We start in section 4.1 by introducing the ICP algorithm in detail. On the basis of this introduction we show how to build up an ICP SLAM algorithm in section 4.2. The next point in this chapter is the presentation of the implementation details for the ICP SLAM realization made within this work in section 4.3. Finally, the chapter finishes with a discussion of the parametrization of the algorithm in section 4.2 and a summary of the chapter in section 4.5.

4.1 Theoretical background of the ICP algorithm

The Iterative Closest Point (ICP) algorithm [13, 5] rotates and translates a point cloud $\{d_j\}$ with respect to another point cloud $\{m_i\}$ to minimize the sum of the quadratic distances between corresponding points in both sets. Formally this can be expressed as the minimization of the function

$$E(R,t) = \sum_{i}^{N_m} \sum_{j}^{N_d} w_{i,j} \left\| m_i - R d_j + t \right\|^2 \tag{4.1}$$

where R is a rotation matrix and t is a translation vector. In this equation $w_{i,j}$ is a weight factor that expresses correspondences.

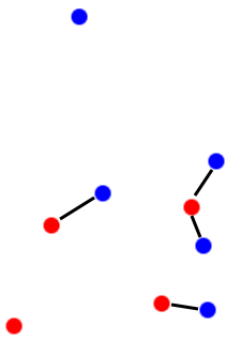


Figure 4.1: Corresponding points. In this figure the point cloud $\{d_j\}$ is shown in blue while $\{m_i\}$ is shown in red. For each point in $\{d_j\}$ the nearest point in $\{m_i\}$ is determined. If the distance between the two points is smaller than c_{max}, they form a corresponding point pair. The point on top of the image has no partner that fulfills this requirement.

One of the first questions arising when looking at equation 4.1 is how to determine $w_{i,j}$. A simple approach to this problem is to fuse nearest points in the two sets to a corresponding point pair. Figure 4.1 shows the approach we take to define correspondences. For each point in $\{d_j\}$ we search for the nearest point in $\{m_i\}$ and form a corresponding point pair if the distance between the two points is smaller than some maximal distance c_{max}.

Note that there may be points in $\{m_i\}$ that have no correspondences in $\{d_j\}$, although points in $\{d_j\}$ are nearer than c_{max}. We set $w_{i,j}$ to 1 if m_i and d_j form a corresponding point pair. If this is not the case $w_{i,j}$ is 0.

If the point cloud $\{d_j\}$ is rotated or translated, the correspondences change. The ICP algorithm therefore works iteratively. First it chooses correspondences, then it minimizes the cost function 4.1 by translating and rotating $\{d_j\}$. The algorithm repeats these two steps until some convergency criteria are met. These convergency criteria are maximal translations and rotations for a converged correction of the position and orientation of $\{d_j\}$ allowed in an iteration. It is assumed that the correspondences in the last iteration step are correct.

The central step in the ICP algorithm is the minimization of 4.1. To do this, the algorithm first determines the required translation and then the rotation. To calculate the translation, we first form the subset of points of $\{d_j\}$ that have corresponding points in the other set. We denote the subset as $\{d_j^c\}$ and the corresponding points as $\{m_{f(d_j^c)}\}$. The centroids of the two subsets are then given by

$$c_d = \frac{1}{|\{d_j^c\}|} \sum_{j}^{|\{d_j^c\}|} d_j^c \tag{4.2}$$

and

$$c_m = \frac{1}{|\{d_j^c\}|} \sum_{j}^{|\{d_j^c\}|} m_{f(d_j^c)}. \tag{4.3}$$

After calculating the centroids c_d and c_m the translation is given by $t = c_m - c_d$. We continue by translating $\{d_j^c\}$ and $\{m_{f(d_j^c)}\}$ by there centroids and define the new sets

$$D' = \{d'_j = d_j^c - c_d\} \tag{4.4}$$

and

$$M' = \{m'_{f(d_j^c)} = m_{f(d_j^c)} - c_m\} \tag{4.5}$$

which are centered in the origin. The minimization of 4.1 then reduces to the minimization of

$$E(R) = \sum_{j}^{|\{d_j^c\}|} \left\| m'_{f(d_j^c)} - Rd'_j \right\|^2, \tag{4.6}$$

such that only the rotation matrix R is left to be calculated. To do this, we set up the correlation matrix H as

$$H = \sum_{j}^{|\{d_j^c\}|} d'_j m'^T_{f(d_j^c)} \tag{4.7}$$

and perform a singular value decomposition $H = U\Lambda V^T$ of this matrix. The rotation matrix is then given by $R = VU^T$. Although this is difficult to see, there exists an algebraic proof [14] that the rotation matrix can be calculated in this way.

The calculation of the rotation matrix completes the theoretical background for the ICP algorithm. The result is summarized in algorithm 4.

4.2 SLAM on the basis of the ICP algorithm

Nüchter et al. [5, 15] propose a 3D SLAM algorithm for an uneven environment on the basis of the ICP algorithm. Although the ICP algorithm is the central part of the proposal, there are also other important parts in it.

Nüchter makes a multiple step approach to estimate the correct robot pose. The first of these steps is the extrapolation of the robot pose on the basis of the last known robot pose and the odometry data. The next step is the construction of an octree in which the scan points of the current pose are stored. Nüchter constructs such an octree for multiple poses near to the estimated pose of step one. Of course, this can only be done for a few sample poses. Another octree is built-up by the map data and finally the map data octree

Algorithm 4 The ICP algorithm

procedure IcpAlgorithm($\{m_i\}$, $\{d_j\}$, c_{max})
  repeat
    for $j = 1$ to $|\{d_j\}|$ **do**
      find corresponding point $m_{f(d_j)}$ to d_j in $\{m_i\}$ if it exists
    end for
    calculate centroids c_d and c_m of subsets $\{d_j^c\}$ and $\{m_{f(d_j^c)}\}$
    $t \leftarrow c_m - c_d$
    calculate sets D' and M' defined in equations 4.4 and 4.5
    calculate correlation matrix $H = \sum_j^{|\{d_j^c\}|} d'_j m'^T_{f(d_j^c)}$
    perform singular value decomposition $H = U\Lambda V^T$ of H
    calculate rotation matrix $R = VU^T$
    translate and rotate $\{d_j\}$ by t and R
  until convergency is reached
end procedure

is compared with the octrees for the possible current poses. Nüchter then takes the pose which agrees best with the map and uses the corresponding pose as the input pose for the ICP algorithm. The scan points are translated and rotated according to the choice of the robot pose. The rotations and translations calculated by the ICP algorithm are then applied not only to the scan points but also to the current robot pose.

In the ICP algorithm, Nüchter chooses not to compare the current scan points with the whole map but only with the scan points from the last known pose. This has the advantage that the computational requirements for the ICP algorithm do not depend on the size of the map.

The algorithm is completed by a mechanism for loop closing of the map in cyclic environments. For this, Nüchter also builds up octrees of the current pose and poses near to this pose. However, this happens after the ICP algorithm has finished. The octrees are compared to the map again and if an octree different to the robot pose better agrees with the map the pose is corrected according to this agreement. Then, the correction is passed through the history of robot poses and all poses are refined such that a consistent map is produced.

To speed up the runtime of the algorithm Nüchter uses a point reduction mechanism. The scanned points are also stored in a kd-tree. This considerably enhances the performance of point search operations.

4.3 Implementation details

We realize an ICP SLAM algorithm in the programs *IcpSocketSlam* and *IcpFileSlam*. The latter of these programs does not work in the exploration context but reads files of recorded robot histories. These histories consist of the actions and measurements of the robot at

each time step. The IcpSocketSlam program instead connects to the RobotController program within the distributed software architecture that we use for the exploration system. Of course, it can be run on a dedicated computer, just like any other program in the collection of programs that form the exploration system. It can also be easily replaced by another program implementing another SLAM algorithm since the communication with the RobotConnector program only incorporates data that is independent of the SLAM algorithm.

Since the data acquisition itself is so time-consuming that a continuous movement of the robot is not feasible, we do not have to restrict the implementation of the ICP SLAM algorithm with respect to some runtime constraints. Therefore, we focus the implementation on accuracy and make any design decision that trades accuracy vs. runtime in favor of accuracy. This implies that we do not use any point reduction technique to speed up the execution of the program. The other main aspect affected by this principle is the part of the map that is compared with new scan data. While Nüchter compares new scan data only with the last scan, we compare it with all the scan data collected so far. We assume that these two design decisions yield a better accuracy in the estimation of the robot poses.

The focus of our exploration system lies on indoor environments. This type of environment is associated with rather good robot pose estimations after odometry extrapolation. Therefore we do not need the coarse robot pose estimation step with the octrees that Nüchter's algorithm performs. We also abandon the option of the loop closing step, although this may be implemented in a succeeding work. So far, our work basically incorporates the central ICP algorithm.

We represent the map in two ways. The first representation is a list of robot poses with corresponding scan data. The second representation is a subdivision of the space into a regular grid where each grid cell contains the scanned points that lie in this region of space. We use this data structure for search operations like the search for corresponding points.

One important detail of our realization of the ICP algorithm refers to the number of degrees of freedom that our robot has. While Nüchter realizes a SLAM algorithm for a robot with 6 DoF we only have to deal with 3 DoF. This affects the correlation matrix H and the rotation matrix R. In Nüchter's case these are 3×3 matrices while we have to deal with 2×2 matrices. This means that the z coordinate is not represented in H, although it plays a role when corresponding point pairs are searched. Note that this variation of the ICP algorithm means that the converged result does not necessarily minimize expression 4.1, but empirically we obtain very good results.

Of course, the performance of our ICP SLAM algorithm can be further optimized. Future modifications of this algorithm may also use the very promising kd-trees that Nüchter uses. However, for our current requirements this would be an unnecessary complication of the algorithm.

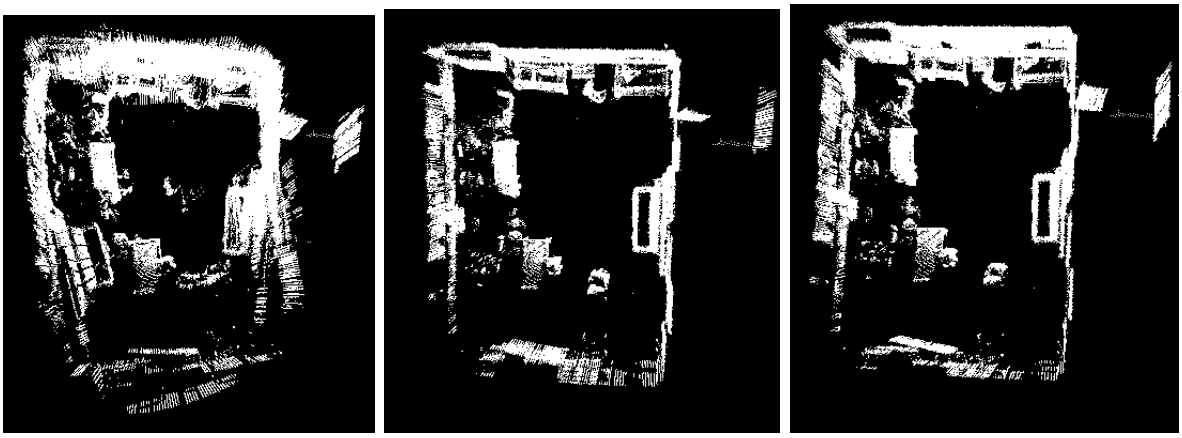


Figure 4.2: ICP SLAM for 14 scans with different parametrization. **Left:** $c_{max} = 0.300$m, $t_c = 0.0050$m, $r_c = 0.170°$, complete runtime: $84min$. **Center:** $c_{max} = 0.100$m, $t_c = 0.0005$m, $r_c = 0.020°$, complete runtime: $59min$. **Right:** $c_{max} = 0.085$m, $t_c = 0.0002$m, $r_c = 0.010°$, complete runtime: $74min$. There are no visual quality differences for the SLAM results shown in the center and right image.

4.4 Parametrization of the ICP SLAM algorithm

The ICP SLAM algorithm has 3 parameters that have to be optimized to get the best results. These are the maximal distance between two corresponding points c_{max} and the convergency criteria for the translation t_c and the rotation r_c. These parameters are not independent of each other. For example, the two convergency criteria of the translation and the rotation are tied together. Interestingly, it makes sense to also make c_{max} dependent on the convergency criteria. A c_{max} that does not fit to the convergency criteria may, for example, lead to no convergency at all. This happens if c_{max} is so small that too few corresponding point pairs are found. If the convergency criteria are too large in this case, the resulting translation and rotation in the first iteration may fulfill the criteria, although we get a very bad result. Neither it makes sense to make c_{max} too large and the convergency criteria too small. In this case we assume a very good result in the end. This would be inconsistent with large distances between corresponding points. Therefore, we only have to optimize one of the parameters and choose the other parameters such that they fit to the optimized parameter. Such an optimization is shown in figure 4.2.

The figure shows the SLAM result for 14 scans in a room of about 10×8 meters. To get a better visualization of the quality of the results the floor and the ceiling are cut out of the images. Note that the runtime of the algorithm depends on the parameters in a complicated way. A large c_{max} and large convergency criteria yield convergency in very few iterations. On the other hand a large c_{max} also yields a larger required time for a single iteration since corresponding points have to be searched in a larger region around a given point. The figure shows that the result in the center is optimal with respect to accuracy and runtime. We choose the corresponding parameter set for the ICP SLAM algorithm.

4.5 Summary

In this chapter we introduced the ICP SLAM algorithm. The basis of such an algorithm is the ICP algorithm for registering two point clouds with respect to each other. We presented the theory behind this algorithm in section 4.1. After the theory was presented, we sketched the ICP SLAM algorithm by Nüchter et al. in section 4.2. This algorithm enhanced the central ICP part with a coarse alignment step under usage of octrees. It also incorporated a loop closing mechanism and several performance optimizations that traded a better runtime to the cost of worse accuracy.

Our realization of an ICP SLAM algorithm was sketched in section 4.3. We took a route that was stronger oriented on the basic ICP algorithm and we chose to design the algorithm to get optimal accuracy. Finally, we discussed the parametrization of the algorithm in section 4.4. It turned out that there is an optimal parametrization that gave us excellent results while minimizing the runtime. Of course, this optimal parameter set depends on the environment, the robot, and the type of operation. But if these circumstances are known, it can easily be estimated in an empirical way.

Chapter 5

Exploring the Environment

Environment exploration may or may not be based on a map that the robot possesses. Approaches that do not rely on a map are based on a few rules that let the robot move on the basis of its current measurements. For example, one of these approaches is the so-called *wall-following*. A robot that performs wall-following moves along a wall which he is aware of due to his current measurements.

The objective of this work is the autonomous exploration of the environment on the basis of 3D SLAM. This is a more sophisticated idea that uses the 3D map produced by the SLAM algorithm to choose the next robot location and navigate to it.

In this chapter, we present our exploration idea, which is realized in the program *Explorer*, in detail. The general idea is introduced in section 5.1 while the implementation details are shown in sections 5.2 to 5.5. Finally, we finish the chapter with a summary of the presented algorithm in section 5.6.

5.1 An idea for an exploration algorithm

Since there are relatively few approaches to 3D SLAM, there also exist only a few approaches to developing an exploration algorithm on this basis. We have some additional constraints to the movement of the robot so that our exploration approach is probably unique. The main constraint relates to the scan-stop strategy that we use for the exploration system. We want to take measurements at relatively few points and the movement between these points should be as simple as possible. We intend to move on a straight line between the scan locations as often as possible. We also have a special environment. This is an indoor environment that has a flat floor. Therefore we have to plan the movement on this flat floor, which is not directly accessible through the map data. We have to extract the accessible floor from the map data and this is the first step in the exploration algorithm.

The next step is the setting of landmarks on the floor. We do that to have some special points between which the robot moves on a straight line. Of course, we also want to decide where the robot should move to in the next step on the basis of the acquired map. We build up two 2D maps to make this decision. The first map stores the costs associated

with the movement to a certain location. The second map stores the knowledge that we have already gathered about a given point on the map. The decision where to move the robot then is an optimization problem on the basis of these two maps. The last step in the exploration algorithm is the actual path planning. We will apply a simple algorithm that moves the robot between the starting point, the landmarks, and the destination point.

Algorithm 5 The exploration algorithm

1: **procedure** EXPLORE(x_t, m)
2: Extract the floor from map data m.
3: Perform a distance transform of the floor.
4: Erode the floor by the robot radius.
5: Generate a homotopy preserving skeleton of the accessible floor.
6: Select landmarks on the skeleton.
7: Generate a graph of *direct reachability* between the landmarks.
8: Generate the cost map for the accessible floor on the basis of the current pose x_t.
9: Generate the knowledge map on the basis of scan locations in the map m.
10: Select a point on the accessible floor with low knowledge and low cost.
11: Plan path to selected point.
12: **return** path
13: **end procedure**

Our approach for a single exploration step is summarized in algorithm 5. After extracting the floor, we erode it by the radius of the robot to obtain the accessible floor for a robot that is reduced to a point. To do this, we first generate a map that stores the distance to the floor boundary for every point of the floor. We then delete all points from the floor that feature a distance smaller than the robot radius, such that the robot would hit something if it moved there. The next step in the exploration algorithm is the selection of landmarks on the floor. To do this, we first select the local maxima of the distance transform and use these points as anchors for the generation of a homotopy preserving skeleton. Additionally to the anchor points we choose special points like junctions in the skeleton as landmarks. If one special point on the skeleton is not directly reachable from a neighboring special point, we additionally select points in between. We assume that we can reach every point on the accessible floor on a direct line from one of the anchor points, although we can think of special situation in which this is not the case. For example, a floor that looks like a perfect donut in 2D has no local maxima in the distance transform from which another point on the floor is directly reachable, at least if the space would not be discretized. We assume that such a situation will not occur in the exploration. Therefore the anchor points let us reach every point on the floor while the special points on the skeleton assure that we can reach each anchor point from another anchor point if such a connection is physically possible. On the basis of the landmarks on the skeleton and the current position x_t we continue and build up a cost (distance) map of the floor. We also generate a knowledge map on the basis of past scan locations. These two maps are used to select the optimal location for the next scan. After selecting this point, we

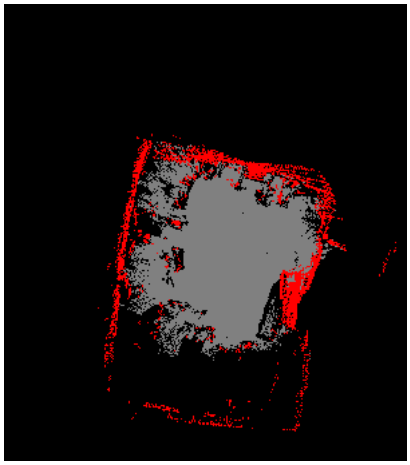


Figure 5.1: Extracted floor. The image shows the floor map after the basic floor extraction step. Gray pixels mark cells that only contain the floor flag, red pixels mark cells with the obstacle flag, and black pixels mark cells with unknown status. Note that the underlying 3D map is chosen to be suboptimal in this example.

finish by planning a path from x_t to the new scan point on the basis of the connections between the landmarks. The next sections present the algorithm in detail.

5.2 The accessible floor

The starting position for the extraction of the floor is the map m. This map consists of a collection of robot locations associated with the corresponding measurements at those position. These are point clouds. We want to have a 2D representation of the accessible floor. Therefore, we subdivide the relevant 2D area into a regular grid. For each grid cell, we determine if there is floor in that region, if there is an obstacle in the region, or if we do not know anything about the region. To do this, we define a thin layer of thickness d_{floor} around the floor. For every scanned point in the map we check if the point is located in this layer and if this is the case we mark the corresponding cell in the floor map as a *floor cell*. Above the floor layer we define another layer of thickness d_{robot} which has the thickness of the robot's height. For every scanned point in the map which lies in this layer, we mark the corresponding grid cell in the floor map as an *obstacle cell*. After doing this, cells that do not feature the floor cell or obstacle cell flag are cells for which we do not have enough knowledge to decide what they contain.

Figure 5.1 shows the extracted floor after performing the procedure described above. The example uses a suboptimal 3D map as basis to show the importance of very well aligned scans. The bad alignment of the scans yields many obstacle cells in the floor map. This ultimately limits the movement of the robot.

The next step in the determination of the accessible floor is the calculation of a distance transform of the floor-only cells in the floor map with respect to any other cell type. There

exist many approaches for distance transforms [16, 17, 18, 19, 20]. We choose to implement the approach by T. Saito and J. Toriwaki [20] since it shows very promising properties in surveys like [21].

The algorithm's input is a 2D image f with width W and height H that contains foreground (1) and background (0) pixels. In our application, floor-only cells are foreground pixels while every other cell is a background pixel. The algorithm calculates the square distance of each foreground pixel to the nearest background pixel. To do this, the algorithm first calculates the image g given by

$$g_{ij} = \min_x \{(i-x)^2 | f_{xj} = 0, 1 < x < W\}. \tag{5.1}$$

This is a transform of the image in i-direction. g_{ij} contains the square distances of foreground pixels to background pixels in this direction. Since the calculation can be done by scanning f once from left to right and once from right to left, this is an $\mathcal{O}(n)$ operation where n is the number of pixels.

The next step is a transformation in j-direction to calculate the image h which is given by

$$h_{ij} = \min_y \{g_{iy} + (j-y)^2 | 1 < y < H\}. \tag{5.2}$$

This new image contains the square of the desired distance transform. If done in a smart way, this last step can be calculated in $\mathcal{O}(n \cdot Average\{\sqrt{g_{ij}}\})$. We assume that the average value of $\sqrt{g_{ij}}$ does not depend on the image size. In this case the whole algorithm has a runtime of $\mathcal{O}(n)$. For our application on maps of indoor environments, this assumption is true for large maps since the size of a room typically does not depend on the size of the map.

After calculating the distance transform we remove all pixels from the floor that feature a distance to the boundary smaller than the robot radius. By this the robot is effectively reduced to a point and the remaining floor represents the accessible floor for this point-like robot. An example showing the distance transform of the remaining floor pixels is shown in figure 5.2. Note that we stick to the same situation already shown in figure 5.1.

The remaining floor pixels are then used to determine landmarks on the floor.

5.3 Landmarks on the floor

To choose the landmarks on the floor, we generate a homotopy preserving skeleton of the floor. There exist a couple of skeletonization algorithms [22] from which we choose to implement the algorithm by L. Vincent [23] since it features homotopy preservation and several other good properties. Vincent's algorithm works on a hexagonal grid. Since we work on rectangular grids, we modify his approach to work on such grids. We assume 4-connectivity on these grids. This is an important decision since it affects the homotopy. Actually, it even turns out that an implementation of Vincent's algorithm on a rectangular grid with 8-connectivity does not work.

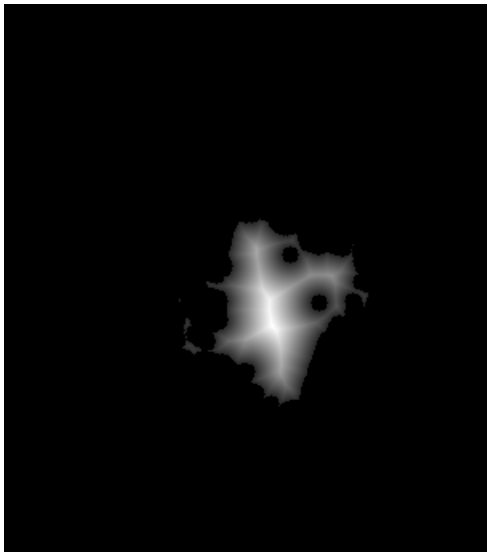


Figure 5.2: Distance transform of the floor. The image shows the distance transform of the floor. The non-accessible pixels due to vicinity to the boundary are already removed.

Vincent's algorithm consists of three steps. The first step is an initialization step in which the anchor points are set and the data structures that are temporally used by the algorithm are created and filled. The second, so-called propagation step generates an initial skeleton and the last step prunes the extremities of the skeleton branches to create the final skeleton.

Of course, the algorithm has to have knowledge about the important pixels for the homotopy of the skeleton as well as knowledge about the pixels that can be pruned. For this it uses tables. For the hexagonal grid these tables can be found in [23] and [24]. Since we use a rectangular grid, we adapt these tables to be used with such grids. The resulting table for homotopy relevant pixels is shown in figure 5.3. If one of the homotopy relevant pixels would be removed from a skeleton, the homotopy of the skeleton would change. The table showing the pixels that can be pruned is shown in figure 5.4.

As an input, Vincent's algorithm anticipates an image I with foreground (1) and background (0) pixels. Another image I_a incorporates the anchor points for the skeleton. These points are also marked with a 1 in the image. Any other point in it is 0. As anchor points we use the local maxima of the distance transform calculated in section 5.2. The algorithm's output will be generated in the input image I.

The algorithm begins with an initialization step. In this step the anchor points are marked with a 1 in I. Every other foreground pixel in I is marked with a 2. Next, we fill a FIFO queue with the border pixels and set these pixels to 3 in I. The border pixels are foreground pixels that have background pixels in their direct neighborhood.

After initializing the FIFO queue the algorithm continues with the propagation step. This propagation pulls every pixel p from the FIFO queue and performs the following steps on it. It first determines the neighborhood configuration of the pixel and puts

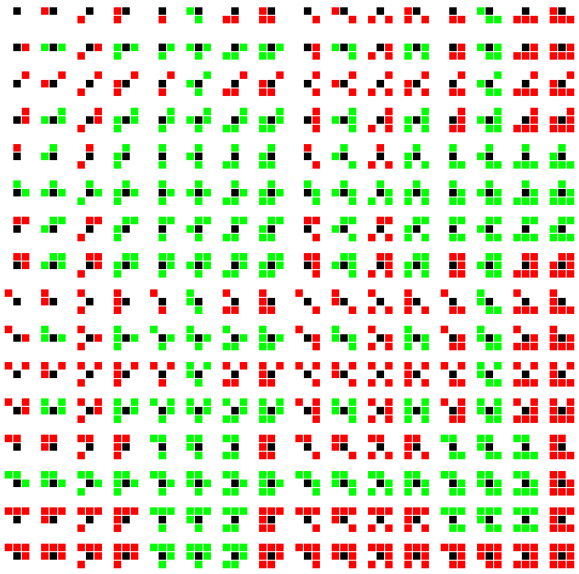


Figure 5.3: Homotopy relevant pixels. For every possible combination of set and unset pixels in the neighborhood of a black reference pixel, it is shown if the reference pixel is relevant for the homotopy. This is the case if the set neighboring pixels are colored in green. If the neighborhood is colored in red the reference pixel is not relevant for the homotopy.

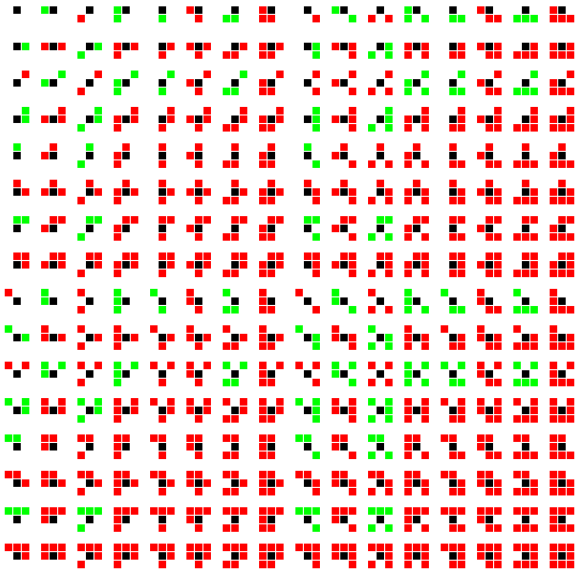


Figure 5.4: Pruning relevant pixels. For every possible combination of set and unset pixels in the neighborhood of a black reference pixel, it is shown if the reference pixel can be pruned from the skeleton. This is the case if the neighborhood of the pixel is colored in green.

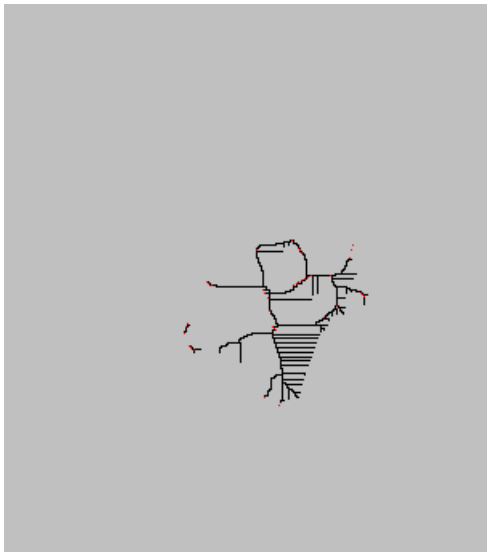


Figure 5.5: Skeleton after propagation step. The skeleton after the propagation step still features several extremities which are unnecessary and have to be pruned to get the final result. The red points mark the anchor points for the creation of the skeleton.

each neighboring pixel p_n into the FIFO queue for which $I(p_n) = 2$ is true. For such a pixel $I(p_n)$ is then set to 3. Next, the propagation step checks with the neighborhood configuration and the table in figure 5.3 if the current pixel is relevant for the homotopy. If this is the case the pixel is put into a data structure TAB. In the opposite case $I(p)$ is set to 0. This removes p from the foreground region and therefore from the skeleton. The result after the propagation step is a skeleton with several unnecessary extremities. An example for such a skeleton is given in figure 5.5. Note that we still stick to the same situation already taken under scrutiny in the other examples in this chapter.

The final step of Vincent's algorithm is the pruning step. For this, we check every pixel in TAB if it can be pruned. This is determined by the pruning table in figure 5.4. If a pixel p can be pruned, we set $I(p)$ to 2 to mark it as a boundary pixel and add it to the FIFO queue.

Next we eliminate the branches. For this we check for every pixel p in the FIFO queue if it can be pruned. If this is the case we set $I(p)$ to 0 and put every pixel p_n in the neighborhood of p for which $I(p_n) = 3$ into the FIFO queue and set $I(p_n)$ to 2. If p cannot be pruned we set $I(p)$ to 3.

After doing this we set $I(p)$ to 1 for every pixel with $I(p) = 3$. Now, the extremities are pruned from the skeleton and the final result is given by the pixels marked by 1 in I. For our example situation the final skeleton is shown in figure 5.6.

The figure also shows the special points that are selected as landmarks on the skeleton. The selection of these points is also based on a table shown in figure 5.7. Additional special points are selected on the skeleton between two special points A and B which are directly connected by the skeleton if the robot cannot move from A to B on a direct line.

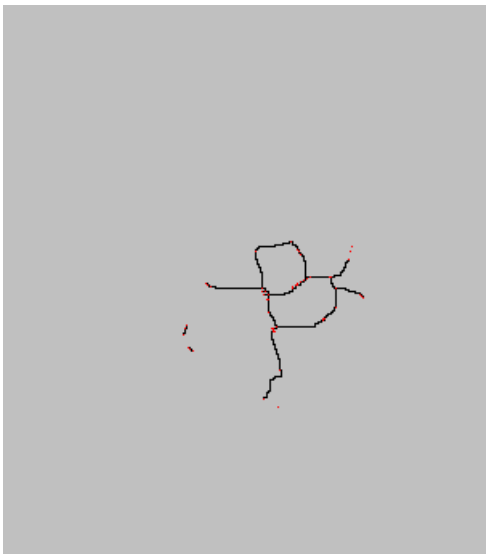


Figure 5.6: Final skeleton with additional landmarks. The image shows the final skeleton after the pruning of its unnecessary extremities. Special points on the skeleton that are used as landmarks, as well as the anchor points, are colored in red.

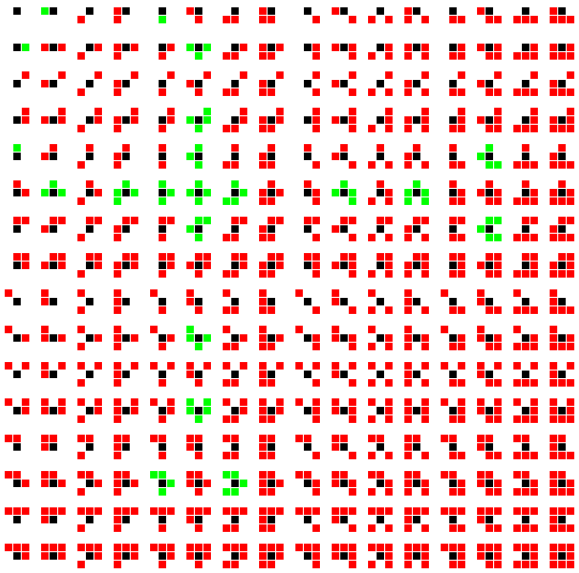


Figure 5.7: Special points on the skeleton. For every possible combination of set and unset pixels in the neighborhood of a black reference pixel, it is shown if the reference pixel is a special point in the skeleton. This is the case if the neighborhood of the pixel is colored in green.

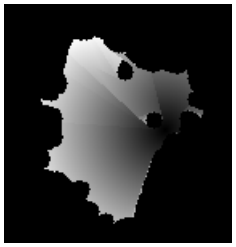


Figure 5.8: Cost map. The brighter a point on the map is the higher are the costs of moving there. It is easily visible that some points are only reachable by an indirect traveling route over certain landmarks. The current robot position is also easily detectable in this image.

After calculating the skeleton and the landmarks on it we proceed in the next section by defining cost and knowledge maps.

5.4 Cost and knowledge maps

To decide where the robot should move to, we construct a cost and a knowledge map. While the cost map reflects the costs of moving to a certain point on the floor, the knowledge map shows the knowledge that the robot already has gathered about a point. The definition of both maps is in principle arbitrary. The same applies to the algorithm that determines the next scan point on the robot's agenda. The only important thing is that the combination of the maps and the algorithm result in a reasonable behavior of the robot. We start this section by defining the cost map.

We define the cost map on the basis of the distance the robot has to travel to reach the point. The point can either be reached directly or by moving over a sequence of landmarks on the skeleton and finally to the point. We additionally define a maximal distance the robot travels on a line. The cost map is now created as follows. The first step is the calculation of the distance to certain landmarks which can be reached directly or by moving over a sequence of other landmarks. We first check which landmarks can directly be reached from the starting point and then use a simple shortest path algorithm like *Dijkstra's algorithm* on the graph that reflects the connectivity between the landmarks to estimate the distance to all landmarks. When we have these distances we check for every point in a local environment of each landmark and of the starting position if the point is directly reachable from that location. If this is the case, we check if there is already a shorter path to this point known and if this is not the case we associate the point with the given landmark or with the starting position. The point is also associated with the total traveling distance needed to get there. In the calculation of these distances, the local environment is given by the maximal direct movement distance. The final cost map for our example situation is shown in figure 5.8.

The cost map algorithm's runtime is proportional to the square of the maximal direct movement distance which is a constant. It is also proportional to the number of landmarks.

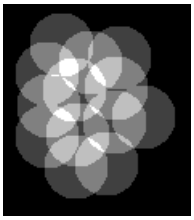


Figure 5.9: Knowledge map. The brighter a point on the map is the higher is the knowledge already obtained for this point. The scan positions of the robot are easily detectable in this image.

We assume that the number of landmarks grows linearly with the map size. Therefore, the algorithm is very fast. This good performance behavior would be broken if the robot was allowed to move without any limit on a direct line. But this is not a desirable behavior of the robot after all. We want to be able to make relatively good predictions of the robot position by extrapolating the last position on the basis of odometry data. This only works well if the traveled distance is not too large.

The knowledge map is defined in such a way that every scan adds knowledge to its environment. We define a knowledge enhancement radius to determine this environment. If the robot makes a scan it increments the knowledge inside an area with that radius around its current position by 1. The knowledge map for our example situation is shown in figure 5.9.

We end this section by defining the next point the robot should move to. We choose the point that maximizes

$$b_{xy} = \frac{1}{k_{xy} + k_0} \cdot \frac{1}{c_{xy} + c_0} \tag{5.3}$$

where k_{xy} is a point on the knowledge map, c_{xy} is a point on the cost map and k_0 and c_0 are constants that are used as knowledge and cost offsets.

The selection of the maximal movement distance d_{max}, the knowledge enhancement radius k_r, as well as k_0 and c_0 is more or less arbitrary. The behavior of the robot can be tuned by varying these parameters. We choose these parameters to be $d_{max} = 2.0$m, $k_r = 1.0m$, $k_0 = 1.0$ and $c_0 = 3.0$. However, the choice of these parameters should depend on the properties of the laser range finder, the robot, and the environment.

5.5 Moving to the next robot location

The last step to be defined for the exploration algorithm is the path planning to the next selected scan point. Implicitly, this is already done by determining the cost map. The cost map consists of traveling distances that are needed to reach a given point. To calculate these costs the path to each point has to be known. We only have to extract this implicit information from the cost map and the connectivity graph for the landmarks.

While calculating the cost map we associated each point on the map with the landmark from where we would move to it or with the starting position. We also associated each landmark with the landmark from which we would move to it. If we can reach a landmark directly from the starting position the landmark is associated with this position. So, all information to plan the path is already available. We extract it by starting at the target position and moving back to the current robot location over the associated points between.

When the path is extracted, the robot moves to the new position. Since we do not want to accumulate large errors while the robot follows the path, the robot makes a full scan at each intermediate landmark and performs the SLAM algorithm to correct its pose.

5.6 Summary

In this chapter we developed the exploration algorithm which is the core of the system we build. We explained the general idea in section 5.1. In this section we started by defining the constraints for the exploration algorithm. Since the input data, as well as the constraints are special, only very few comparable algorithms exist. At least, we are not aware of any exploration algorithm that applies to the situation we deal with.

The algorithm's approach is rather intuitive and straightforward. It combines several low-level algorithms into one high-level algorithm as a whole. The first steps of this high-level algorithm are the extraction of the accessible floor and the generation of an Euclidean Distance Transform (EDT) of this floor. These steps were taken under scrutiny in section 5.2. The next steps had the objective to define special landmarks on the accessible floor. For this we took the local maxima of the distance transform and used them to create a homotopy preserving skeleton of the floor. The anchor points, as well as special points on the skeleton, then defined the landmarks. We described these steps in section 5.3. The third group of steps of the algorithm had the goal to define cost and knowledge maps. This was presented in section 5.4. Finally, we completed the algorithm by presenting the path planning mechanism which was described in section 5.5.

The resulting algorithm is a robust approach to the exploration task we have to solve. It fulfills the constraints we have defined for the algorithm and to some extend the algorithm is also tunable for differing exploration tasks. This can be done by varying the parameters used to select the next scan point.

Chapter 6

Simulating the Robot

For testing purposes, evaluation purposes, and due to temporary unavailability of the real robot we develop a simulation program *RobotSim* for the robot. This is important since the robot could behave in an unintended way if something in the exploration system does not work as it should. We use the simulation to test the robot's behavior, as well as the visualization of the result. In a simulation we can create basic environments for which errors in algorithms are easily detectable and trackable. Another reason for the development of the simulation is that the rotation platform for the laser range finder was not available in the beginning of this work.

We start this chapter in section 6.1 by describing the basic functionality of the simulation. Next, we introduce the configurable error models for the movement of the robot and for the laser range finder in sections 6.2 and 6.3. The chapter closes with section 6.4 by giving a summary of the simulation program.

6.1 General overview

We develop a robot simulation program for an indoor environment. Although the environment is three-dimensional, we assume a flat floor. Therefore, the robot in the simulation has three degrees of freedom. We also simulate a 3D laser range finder on top of the robot that gathers data from the 3D environment.

The simulation program loads the environment data in terms of data files in the STL format [25]. This is a very common file format for storing sets of triangles together with their normals. Although this file format allows the storage of arbitrary sets of triangles, we make further assumptions about this set. In detail, we assume that the set describes an indoor environment with a floor in the $z = 0$ plane. The coordinates of the triangles are assumed to be given in meters. We construct such environments using the free 3D modeling program *Blender* [26] which features an option to export STL files.

An example image showing the simulation program with a loaded environment and a robot after some movements and measurements is shown in figure 6.1. The triangles containing to the floor are displayed in gray while every other triangle is only shown as a

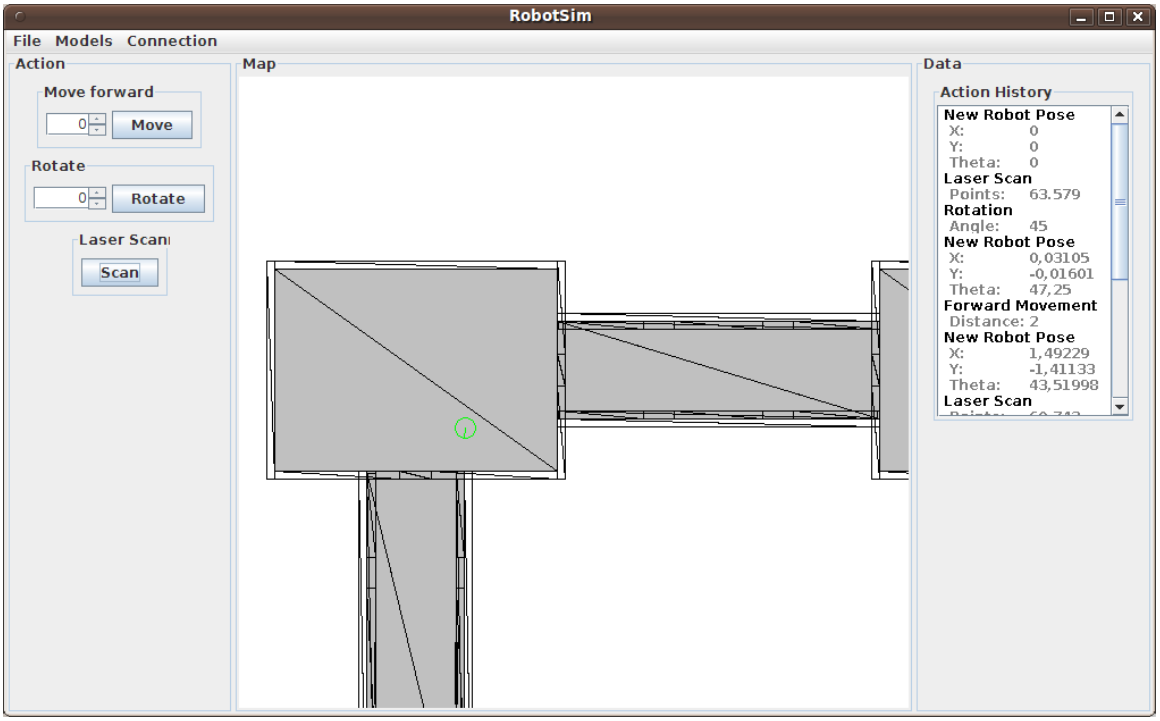


Figure 6.1: Screenshot of the RobotSim program. The robot is displayed as a green circle and its orientation is marked by a green line. On the left pane the action commands for the movement and the sensor measurements can be invoked, while the right pane shows a history of the robot's past movements and measurements.

wireframe. The robot itself is represented as a green circle with a line inside that marks its orientation. The pane on the left side of the program incorporates the control panel for invoking special movement commands for the robot and starting a measurement with the robot's sensors. Of course, the robot can also automatically move if the simulation connects to the central RobotController program. The right pane shows a list of the past movement commands, measurements, and poses of the robot. In this example, it can easily be seen that the movement commands and the robot poses are not completely consistent to each other. This is the case since the movements and measurements are also affected by error models.

The error models can be configured inside the graphical user interface. The action history can be saved in a simple file format that consists of a sequence of identifiers for each action or measurement class together with its parametrization or measurements. The measurements with the 3D laser range finder themselves are simulated by using a simple variant of *ray casting*.

The simulation at the moment only features basic functionalities that are required for our special purpose. A collision detection is not implemented, although the program is easily extendable with this or other functionalities.

6.2 A model for the robot movement

The movement model for the robot is subdivided into a model for the forward movement and a model for rotations. This is adequate since the movement commands coming from the exploration system are also subdivided into these two classes.

The forward movement model assumes systematic errors in the movement and additionally some random deviations. The first systematic error is a systematic curvature. We assume that the robot does not move on a straight line but on a curve. This is realistic since the radii of the two wheels will be slightly different. We also define a random deviation from this curvature. Therefore, we use a Gaussian with a configurable standard deviation and randomly sample a value from this distribution. The value is then added to the systematic curvature of the forward movement.

The second systematic error is a relative distance error. We assume that the radii of the wheels do not exactly match the assumptions of the robot's odometry sensors. This clearly leads to a relative distance error that is proportional to the traveled distance. We also define an additional deviation from this relative distance error by a Gaussian.

The last systematic error in the forward movement model is an error in the finishing angle. We assume that the two independently drivable wheels are not completely synchronized which leads to a small rotation at the end of a forward movement. Empirical observations support this assumption. Additionally to this systematic error in the finishing angle, we define a random error for this quantity.

The rotation model is chosen to be more arbitrary. We define a Gaussian for the change of the position due to the movement and relate the standard deviation of the Gaussian to the rotation angle. Although the position clearly changes due to a rotation, it is questionable how this should be modeled. For the development of a realistic rotation model this should be examined.

We also define a systematic error in the relative angle for a rotation. This is a deviation from the rotation angle that is proportional to the angle. Of course, we also add a random error to this quantity.

The movement model defined in the simulation is not completely consistent with the real robot behavior. For example, we do not model the small swivels that we observe at the beginning of a movement of the real robot. We also do not model any interactions with the floor that lead to additional errors. Note that this is acceptable for our purposes, since the applied SLAM algorithm does not rely on such a realistic error model. If we had to extend the exploration system with another SLAM algorithm that uses an error model for the robot movement we would have to reconsider this issue. The simulation program is easily modifiable with respect to the movement model. If a realistic model had to be implemented this is doable without any problem as soon as this model is known and if interactions with the environment are not part of the error model.

Finally, note that the configurability of the error model features the option to simulate a large set of such models. We can test how the exploration algorithm or other algorithms react to certain movement models and compare it to a perfect movement as a reference. This implies the option to evaluate algorithms with respect to certain robot behavior. If an

algorithm does not work very well due to the behavior of the robot, this can be detected in the simulation. The usage of the real robot does not feature such evaluation options since its movement properties cannot be configured.

6.3 A model for the laser range finder

After defining the movement model for the robot, the only thing that is left to perform the simulation is the definition of a model of the 3D laser range finder. We will define this model in this section.

The 3D laser range finder model used in the simulation is geometrically analog to the real laser scanner and the rotation platform. The simulated rotation axis is conform with the axis of the rotation platform and the height of the laser scanner is configurable. We set it to the height of the real range finder. The number of scan points in a scan plane is also configurable and the same applies to the number of rotation steps used for a full scan of the environment. Finally, the maximal range of the laser range finder is also a configurable parameter.

The error model for a scan differentiates between three cases. The first case is a normal scan in which we calculate the intersection point of the laser beam with the surface of the environment. This is done by applying *ray casting*. If the intersection point is within the range of of the laser scanner we take the distance to the surface and add a distance dependent error to it. We choose this error to be consistent with the specifications of the real laser range finder, although this is a modifiable quantity. If the distance to the intersection point is not within the range of the laser scanner we set the measured distance to the maximal scan range. The real laser scanner acts in a similar way if it cannot detect anything and sets the distance to zero.

The second case is a random scan in which we select a random value for the measured distance which is taken from a uniform distribution over the whole scan range. Random scans are also observed for the real range finder. Although, it is not clear if they are uniformly distributed.

The last case is a maximum scan in which we set the measured distance to the maximum scan distance. Such a case can also be observed in the data taken from the real laser scanner, although the real laser scanner sets the distance to zero.

All cases are associated with configurable relative probabilities that are used to select one of them. Note that this laser range finder model is similar to the model defined in [3], although the other model is more sophisticated and features an additional case to deal with dynamic environments.

Of course, the real laser scanner shows a more complicated behavior. The most important difference is the fact that its measurements depend on the surface which is hit by the laser beam. If this surface is highly reflective the distance is underestimated. If it is absorbing the distance is overestimated. Some surfaces cannot even be detected. The form of an object is also relevant. The measurements are reliable if the surface is large and flat. If the distance to a small object is measured the distance is not very reliable and

can depend on the surroundings of the object.

Note that the evaluation options arising for the movement model due to the configurability also arise for the scanner model. We can simulate the behavior of many different laser range finders and we can also simulate perfect measurements. Therefore, we can also evaluate algorithm behavior with respect to the sensor models.

6.4 Summary

In this chapter we presented the simulation used to test the exploration system. The simulation program is connectable to the RobotController program in the same way the *RobotConnector* program connects to the exploration system. Therefore, the simulation and the real robot can be replaced with each other depending on the purpose of a run of the whole exploration system.

We started the chapter by giving a general overview of the simulation program in section 6.1. We developed a graphical user interface for the program that can be used to control the robot. However, the robot can also be controlled by the exploration system to simulate autonomous exploration of the environment. We continued the chapter in section 6.2 by defining the movement model which incorporated systematic and random inaccuracies in the movement. The movement model is not defined to be completely realistic, although we noted that the model is adequate for our purposes. The same conclusion can be drawn for the laser scanner model defined in section 6.3.

Note that the algorithm evaluation options related to the configurability of the error models, together with the fact that the simulation can also run on its own, transform the simulation program into a valuable tool beyond the purpose of this work. The program can be adapted to provide important data for the evaluation of other algorithms or other sensors and actors as well.

Chapter 7

Visualizing the Map

To evaluate and show the map data generated by the exploration robot we visualize it by two different visualization algorithms. The first visualization is just a 3D-representation of the collected data points. This very basic data visualization is important since it does not imply any data interpretation. The quality of the map can very well be evaluated with this visualization.

The second visualization technique is based on a surface reconstruction. We use the map data and extract a surface from it. Such a reconstructed surface can be used as a basis for more sophisticated visualizations of the map that are beyond the scope of this work. It is, for example, possible to mount a camera on the robot and project the camera image onto the reconstructed surface. After generating the surface in terms of a set of triangles we use OpenGL [27] to visualize it with hardware acceleration. This is very important since the number of triangles is too large for a pure software visualization that features immediately reacting navigation capabilities.

In this chapter we discuss the surface reconstruction based visualization. For this we start in section 7.1 by giving a short overview over the visualization program *SurfaceProcessor* which is developed within this work. To reconstruct a surface from the map data we implement the so-called Ball Pivoting algorithm. This method is described in detail in sections 7.2 and 7.3. While section 7.2 describes the general idea in two dimensions, section 7.3 applies this idea to three dimensional data. Section 7.5 then shows and discusses first results obtained by applying the Ball Pivoting technique. We finish the chapter with a summary in section 7.6.

7.1 A program for the map visualization

The visualization program developed in this work offers two independent visualization modes. We can visualize the map directly by the recorded data points or after performing a surface reconstruction on the map. In the latter case, a set of triangles is visualized. Figure 7.1 shows a screenshot of the visualization program that demonstrates these two modes.

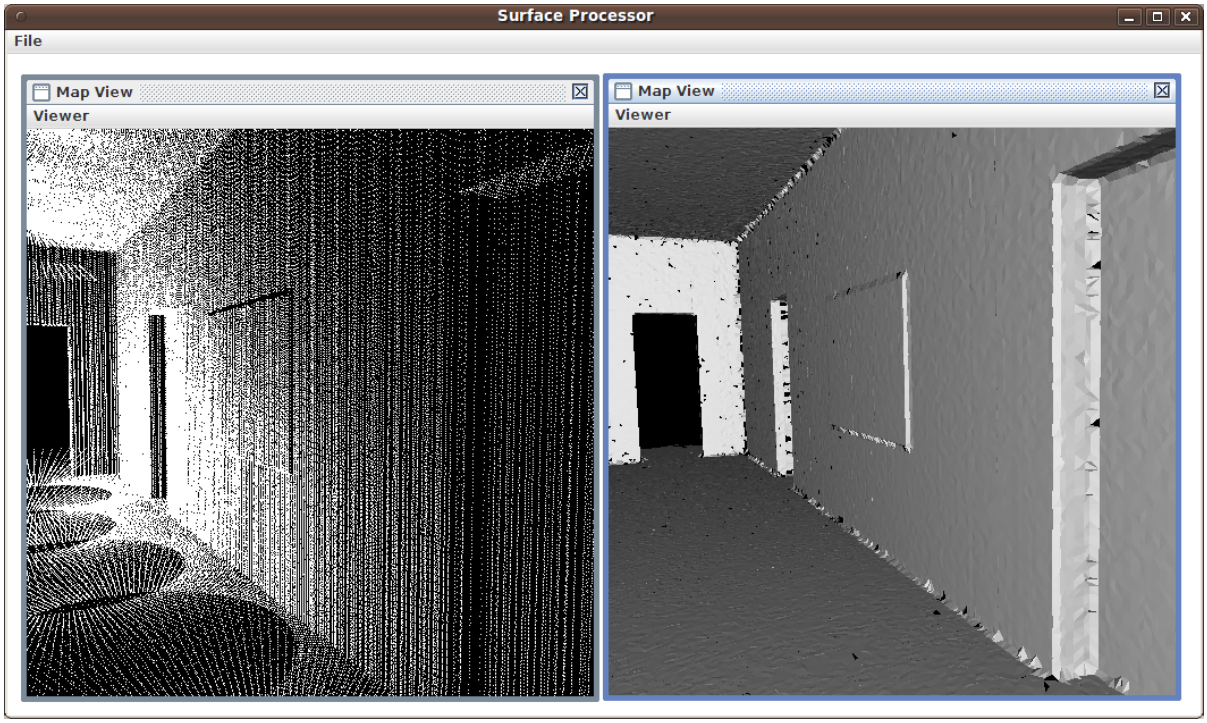


Figure 7.1: Screenshot of the *SurfaceProcessor* program. In this screenshot a simulated environment is visualized. The program has two visualization modes. The left frame shows direct visualization of the recorded data points. The right frame shows the visualization that is based on a surface reconstruction by the Ball Pivoting algorithm.

The program can load map data in a simple format that incorporates a collection of robot poses associated with the measurements taken at these locations. Since the surface reconstruction is rather time-consuming, the program also offers the option to save and load a reconstructed surface. For this, the STL file format [25] is used.

The program offers the possibility to continuously navigate through the visualized map. It is possible to slide in different directions using the keyboard and the orientation of the camera can also be rotated by using the mouse. To provide such a continuous navigation at a reasonable speed, hardware acceleration is required. We use the graphics hardware through the OpenGL [27] libraries. Both visualization modes are hardware accelerated.

There exist several approaches for the surface reconstruction on the basis of scan data from multiple scans [28, 29, 30, 31, 32, 33]. We choose to implement the ball pivoting algorithm [28] for this task since it is a common and state of the art approach that produces very good results. The next sections describe this method in detail.

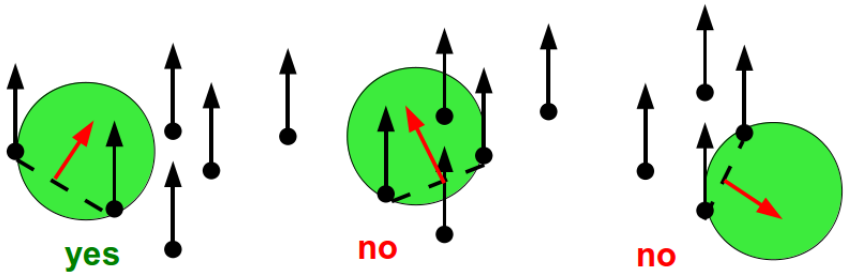


Figure 7.2: Choosing a seed edge. The left image shows a valid seed edge defined by the circle with given radius and two points of the data set. No points lie within the circle and the normals of the edge and the data points point into the same half space. The center image shows an invalid choice of a seed edge. In this case there lies a point from the data set inside the circle. The right image also demonstrates an invalid choice. Here, the normal of the edge points into another half space than the normals on the two ending points.

7.2 Ball Pivoting in two dimensions

To get an easier understanding of the Ball Pivoting algorithm we first introduce the idea for two-dimensional data. This implies some fundamental differences to the 3D version of the algorithm. While the 3D version reconstructs a surface in terms of a set of triangles, the 2D algorithm reconstructs a boundary for data points on a plane in terms of edges. Of course, we also have no ball in 2D but a circle. In two, as well as in three, dimensions the sampled data points are associated with normals on these points. If the data points are produced by a laser range finder it is an obvious choice to choose the normal such that it points towards the laser scanner.

The parametrization of the algorithm is done by choosing the radius of the circle or the ball in 3D. In 2D, the circle will always touch two supporting points from the data set so that the radius should be chosen such that it is consistent with the typical distance between neighboring points on the boundary. In 3D the ball will always touch three supporting points from the map data. Therefore, the radius should be chosen in a similar way.

In 2D, the algorithm starts with the choice of a seed edge in the boundary. This seed edge features some important properties. First of all it is an edge between two points from the data set which are no further apart than twice the radius of the circle. Next, we calculate the position of the circle that touches the two points. Of course, there will be 2 circles that fulfill this property, although only one of the two circles will fulfill our other requirements. The center of the two circles are on different sides of the edge. We define a normal on the edge that points to the center of the associated circle. Now that we have defined a normal on the edge, we can distinguish between the edges associated with each circle. We choose the circle and the edge with the normal that points into the same half space as the normals of the two data points, where the two half spaces are separated by an infinite line through the two data points. If the normals of the two data points do not

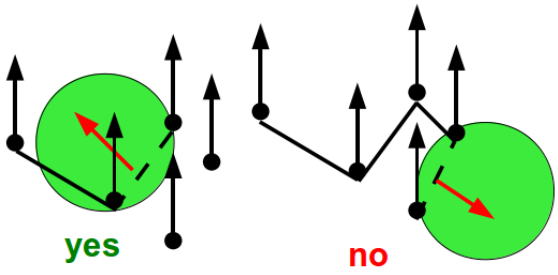


Figure 7.3: Adding edges to the boundary. The left image shows a valid new edge that will be added to the boundary. The right image shows an invalid edge since its normal points into another half space than the normals of its end points.

point into the same half space, we cannot choose one of the edges as seed edge. If one of the edges fulfills all properties up to now, one last requirement also has to be met. No points from the data set, with exception to the ending points of the edge, are allowed to lie inside the circle or touch the circle. Figure 7.2 demonstrates the properties of a seed edge.

After selecting a seed edge, we add it to the boundary. This is just a collection of edges. The boundary also has so-called active points. The end points of the seed edge are the active points we start with. The next step in the 2D Ball Pivoting algorithm is the pivoting step. In this step we select an active point A from the boundary. This active point is associated to an edge $\overline{AB}$ for which a corresponding circle is given. We take this circle and pivot it around the active point until it touches another point C from the data set. With this new point we have a new candidate for a boundary edge. This is the edge $\overline{AC}$. The new candidate has to fulfill the same requirements that had already been fulfilled by the seed triangle. There have to be no other points inside the circle and the normals of the points and of the edge have to point into the same half space. Figure 7.3 demonstrates this situation. If $\overline{AC}$ is a valid boundary edge, we add it to the boundary and also add C to the set of active points. Regardless of the result of the validity check for $\overline{AC}$ we remove A from the set of active points.

The ball pivoting step is repeated over and over again until the set of active points is empty. When this is the case, we cannot expand the boundary selected by the seed edge any further. Of course, there can also be some special cases that have to be treated in a special way. For example, the new edge could add a point to the set of active points that is already in this set. In this case we fuse two ends of the boundary and the new point has to be removed from the set of active points. It can also be the case that the new point already is an inactive point in the boundary. In this case we choose not to add the edge nor the new point to the boundary. If we added the edge to the boundary we would possibly end up in an infinite loop.

In the 2D Ball Pivoting algorithm, the seed edge selection step with its subsequent edge addition steps are repeated until no further seed edge can be chosen. When this point has been reached, the boundary is complete and the algorithm terminates.

7.3 Ball Pivoting in three dimensions

After presenting the Ball Pivoting algorithm in two dimensions we transfer the idea to three dimensional data sets. In this case, we reconstruct a surface in terms of triangles. In principle, the 3D algorithm follows the same approach as the 2D algorithm. We first select a seed element which is a triangle in the 3D case. This triangle is then added to a set of triangles called the surface which is expand as far as possible until no further triangle can be added.

The seed triangle has to fulfill similar requirements as in the 2D case. Although, there are some differences. First of all we use a ball with a fixed radius and not a circle. The ball touches three points from the dataset that are the corners of the triangle. We define the triangle such that its normal points in the direction to the center of the ball that is used to construct it. Of course, there are two possible triangles for each three points with normals in opposite directions. A valid seed triangle has to have a normal that points into the same half space, defined by the plane through the 3 data points, as each normal on its corners. The second requirement for the seed triangle is that no further points may lie inside the ball associated with it.

After constructing the seed triangle, we add it to the triangle set that defines the surface. Further, we add its three edges to the boundary of the surface. This is a set of sequences of connected edges that define the frontiers between the surface triangles and the void. Of course, the sequences are indeed loops since the last edge in a sequence is connected to the first one. Edges in the boundary can be active or inactive, just like the active or inactive boundary points in the 2D case.

The next step expands the surface by pivoting a ball on a surface triangle around one of its active boundary edges until it touches another point. The new point and the two old points from the active edge define a new triangle that is a candidate for addition to the surface. The new triangle has to fulfill the same requirements that the seed triangles have to fulfill. This means that the relevant normals all have to point into the same half space and no other points may lie inside the ball. If the new triangle can be added to the surface this is done. The edge used for pivoting is removed from the boundary and the two new edges are added as active edges to the boundary. If it cannot be added to the surface, the pivoting edge is marked as inactive.

The pivoting step is repeated until no further active edges exist in the boundary. Figure 7.4 shows the selection of a seed triangle with subsequent expansion of the surface. Of course, there exist special cases, the pivoting step has to handle. We assume that the new triangle fulfills all requirements with respect to the normals and other data points inside the ball. If this is given, we have to distinguish between different cases. The first case is that the new data point to be added to the surface is not yet used. This means that this point is not yet part of any surface. This is the case that we have described in the simple example above. We name the operation that removes the pivoting edge and adds the two new edges to the active edges of the boundary a *join* operation. If the new point is already part of the surface, the situation is more complicated. If it is part of the surface but not on the boundary, we cannot add the new triangle to the surface since this

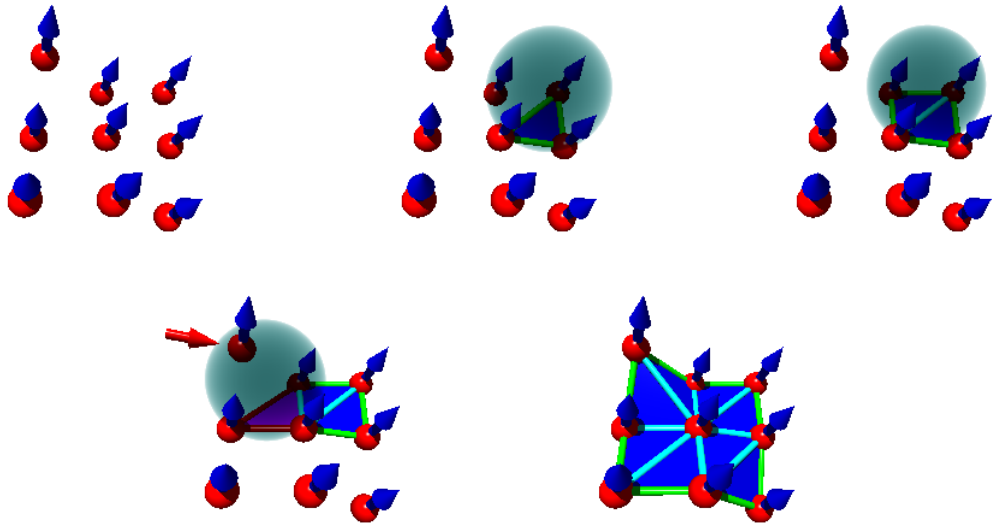


Figure 7.4: Construction of a surface with Ball Pivoting. The upper left image shows the pure sample data set with normals on the data points. In the upper center image a seed triangle is found that fulfills all associated requirements. The upper right image shows a first pivoting step to add another triangle to the surface. In the lower left image, a test triangle fails to fulfill one of the requirements. Here, the associated ball contains another point from the data set that is not part of the triangle. The last, lower right, image shows the completed surface after performing the Ball Pivoting algorithm.

would create a non-manifold vertex which we want to avoid. If the new point already is part of the boundary we also have to check if the addition of the triangle would create a non-orientable manifold. If this is not the case, we perform a simple join operation. This join operation may add edges to the boundary that are already part of the boundary. In this case we have to perform a so-called *glue* operation.

For the glue operation, we define a direction for the edges. This direction is defined such that the edges of a triangle are went through in counter clockwise order if one follows the edge direction. If two edges with the same end points are part of the boundary, they always have different orientation. We have to distinguish four different cases to deal with such pairs of edges. In the first case, a loop of connected edges only consists of two edges with same end points and opposite orientation. In this situation the whole sequence of edges is removed from the boundary. The second case features two adjacent edges with the same end points in a sequence of connected edges. This case differs from the first one by the length of the sequence which here is larger than two. We deal with this situation by removing the two adjacent edges from the loop. In the third case, the two edges are still part of the same loop, although they are not adjacent. Here we have to split the loop into two loops. For this we remove the two edges from the loop. This produces two sequences of edges that are not connected with each other. Although, both sequences form loops on

their own. All the edges in both loops have predecessors and successors. The last case merges two loops. Here, the two edges exist in different loops. Again, the two edges are removed and, if fused together in a correct way, the two resulting edge sequences form a single loop where every edge is connected to two other edges.

These are all problematic cases that have to be dealt with. The approach as a whole, as it is proposed in [28], is summarized in algorithm 6. In this pseudo code algorithm S is the data set of sampled points, while B is the data structure representing the boundary. After presenting the general algorithm we discuss the implementation details in the next section.

Algorithm 6 The Ball Pivoting algorithm

```
1: procedure BallPivotingAlgorithm(S,B)
2:     while true do
3:         while e_{i,j} = getActiveEdge(B) exists do
4:             if σ_k = ballPivot(e_{i,j}) exists then
5:                 if notUsed(σ_k) or onBoundary(σ_k) then
6:                     addTriangleToSurface(σ_i, σ_k, σ_j)
7:                     join(e_{i,j}, σ_k, B)
8:                     if e_{k,i} ∈ B then
9:                         glue(e_{k,i}, e_{i,k}, B)
10:                    end if
11:                    if e_{j,k} ∈ B then
12:                        glue(e_{j,k}, e_{k,j}, B)
13:                    end if
14:                end if
15:            else
16:                removeActiveFlag(e_{i,j})
17:            end if
18:        end while
19:        if (σ_i, σ_j, σ_k) = findSeedTriangle(S) exists then
20:            addTriangleToSurface(σ_i, σ_j, σ_k)
21:            addActiveEdge(e_{i,j}, B)
22:            addActiveEdge(e_{j,k}, B)
23:            addActiveEdge(e_{k,i}, B)
24:        else
25:            return
26:        end if
27:    end while
28: end procedure
```

7.4 Ball Pivoting implementation details

The ball pivoting algorithm does not define the details for its implementation. Questions like how the balls are constructed or how the pivoting is done are not answered by the general algorithm description. In this section we address these questions.

The first question is how to construct a valid seed triangle. For this we select a point from the data set and search for nearby points. The search radius is given by two times the ball radius. To reduce the time needed for the search, we subdivide the space into a regular grid. Each grid cell stores a list of data points which are located in the corresponding region. By doing this, we can focus the search on the relevant region which is a great advance. An even better performance for the search might be achieved by using kd-trees. Of course, we only accept points that are not yet used for the surface. Each triple of points that we find in this way defines a triangle with defined orientation through the ordering of the three points. When we have found such a triangle we first check if its normal points into the same half space as the normals on the corner points. If this is the case we calculate the center point of the corresponding ball. After performing some linear algebra operations this calculation reduces to a quadratic equation which only has real solutions if a ball with the given radius can be constructed that touches the three points. If no real solutions exist we reject the test triangle and search for the next one. If there are real solutions, the two solutions correspond to the two balls on both sides of the triangle. We choose the solution that is in the half space into which the normal on the triangle points. Now that we know the center point of the ball, we search for data points inside the ball. If we find such points we have to reject the triangle. If no point is found that is not one of the corner points, we accept the triangle as seed triangle.

The second important question is how to pivot the ball around an edge until it touches another point. The answer is that this is not done. Instead, we search for valid triangles for which two of the corner points are the end points of the pivoting edge. The validity of the triangle is given by the requirements to the corresponding ball and the normals. For each triangle we find in this way we calculate the pivoting angle for the ball. Finally, we select the triangle that features the smallest pivoting angle.

The last important implementation detail we discuss here is an extension to the approach also proposed by [28]. We apply the algorithm multiple times with different ball radii to the same data set. We start with a small radius and construct a surface. Then we mark the boundary edges as active again and apply the algorithm a second time with a larger radius to the data set. By this procedure, we apply three different ball radii to the data set. This has two main advantages. The first advantage is performance related. A small ball radius results in searches for points in small regions which speeds up the search a lot. The second advance relates to the details visible on the surface. Of course, the minimal size of the visible details is related to the ball radius. By applying different ball radii we get the details with the smallest ball radius and we are also able to construct a surface if the point sampling is not dense enough for a surface reconstruction with a small radius.

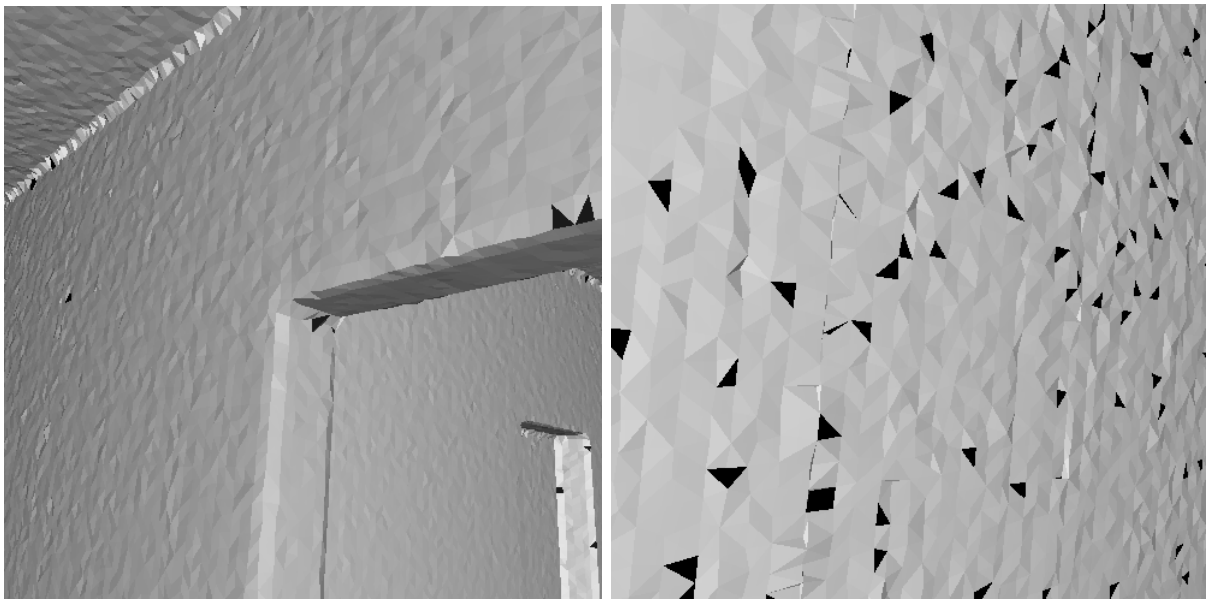


Figure 7.5: Surface reconstruction by ball pivoting for simulated environments. The left image shows a rounding of edges which is a typical property of the algorithm. The right image shows gaps in the reconstructed surface. Due to the functional principle of the algorithm there are gaps that cannot be closed.

7.5 Ball Pivoting results

Finally, we present some results of the surface reconstruction with ball pivoting. Fundamental properties of the algorithm can best be seen on surface reconstructions of simple, simulated environments. Figure 7.5 shows examples for generated surfaces for such environments. The examples show two typical artifacts that can always be seen in surfaces reconstructed by ball pivoting. First of all, the algorithm is not capable of producing sharp edges. This is due to the point sampling not being infinitely dense and the finite ball radius preventing a reconstruction close to an inner edge.

The other artifact are gaps that exist in the reconstructed environment. There are several reasons for such gaps. First of all, if the radius of the ball is given, not all sets of three points can lie on the surface of such a ball, even if the points are close to each other. If, for example, the points lie nearly on a line, the minimal radius for a ball touching all three points is very large. Another reason for a gap is that the normals on the corner points might not point into the same half space. If the resulting gaps are to be closed, a postprocessing of the surface is required that performs this job.

Figure 7.6 shows the surface reconstruction for a single 3D scan on top of a table. Here, we see several artifacts of the scan process. First of all, there are spikes on the ceiling of the room. This is due to inaccurate scan data. Next, there are some gaps in the reconstructed surface that correspond to certain objects. These are objects that feature a high absorption coefficient for the wavelength of the laser range finder beam. Due to

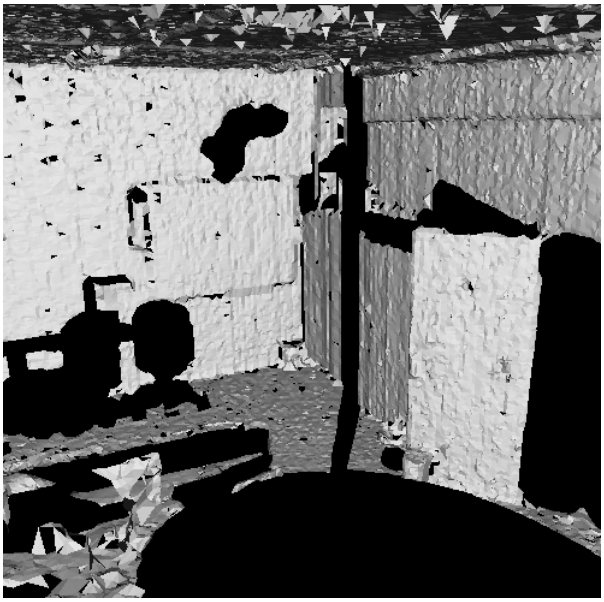


Figure 7.6: Ball pivoting for a single scan. For this scan the 3D laser range finder was positioned on a table. Several artifacts of the scan can be seen: The spikes on the ceiling are due to inaccurate scan data. Additionally, some surfaces with high absorption coefficients are obviously not reconstructed. Nevertheless, some objects are very well identifiable after the surface reconstruction.

the high absorption, the scanner is not able to measure the distance to these objects. Therefore, we do not get any reliable data points for the distance to these objects. There are also some scan planes missing in this image due to bad data transfer from the laser scanner to the microcontroller board.

There are also objects in the surface reconstruction that are very well identifiable. This is the case for the door or for the trash can in front of the door. We also see a board on the wall.

7.6 Summary

In this chapter, we demonstrated two different visualization approaches for the map data. The first approach is a simple representation of the sampled data points. This representation is very well suited for the evaluation of the map. There is no interpretation of the scan data in this method. The other visualization approach is the representation of a reconstructed surface on the basis of the scanner data. Here we interpret the data in terms of surfaces.

We started the chapter in section 7.1 by giving a general overview of the visualization program. Then we discussed the Ball Pivoting algorithm in detail. This discussion included an introduction of a 2D version of the algorithm in section 7.2. On the basis of this introduction we described the algorithm for 3D data in section 7.3. The main

implementation details were discussed in section 7.4 and we finished the chapter by a presentation of surface reconstruction results in section 7.5.

Chapter 8

Autonomous Exploration and Map Building Results

The central objective of this thesis is the development of an exploration system, which was described in the preceding chapters. This chapter now uses the complete exploration system for the exploration of a room of about 10×8 meters. We demonstrate the process of exploring the room and evaluate the final results.

This chapter is organized in two main sections. It starts in section 8.1 by showing the intermediate results for each exploration step. Section 8.2 then evaluates the final result after 21 exploration steps. We conclude the chapter in section 8.3.

8.1 Exploring a room

Here, we show the results for 20 consecutive exploration steps of a room. However, before we start we show a visualization of the final result after 21 exploration steps. This is done to demonstrate the quality of the resulting map. We see the final map in figure 8.1. The two maps with and without SLAM clearly show that we need a SLAM algorithm if we want to perform map based exploration. Without the application of a SLAM algorithm the map has a bad quality that is not adequate for the basis of an exploration system. The SLAM based map instead has a very high quality. This means that all scans are well aligned to each other. There are no wrongly aligned scans in this map that yield a 2D exploration map with non-existent obstacles. Such obstacles, produced by wrongly aligned walls and other objects are the main point why the bad quality map without SLAM is not useful for exploration purposes. The robot would not be able to move a lot on the basis of such a map. Every new scan would limit the possible robot movements even more instead of revealing unexplored floor that is accessible for the robot.

The discussion of the first 20 exploration steps starts with the presentation of the extracted floor from the map data in these steps. This is shown in figure 8.2. Like the final map, the extracted floor shows that the scans are very well aligned to each other. We can also see that each scan expands the area of the extracted floor. This lets us conclude that

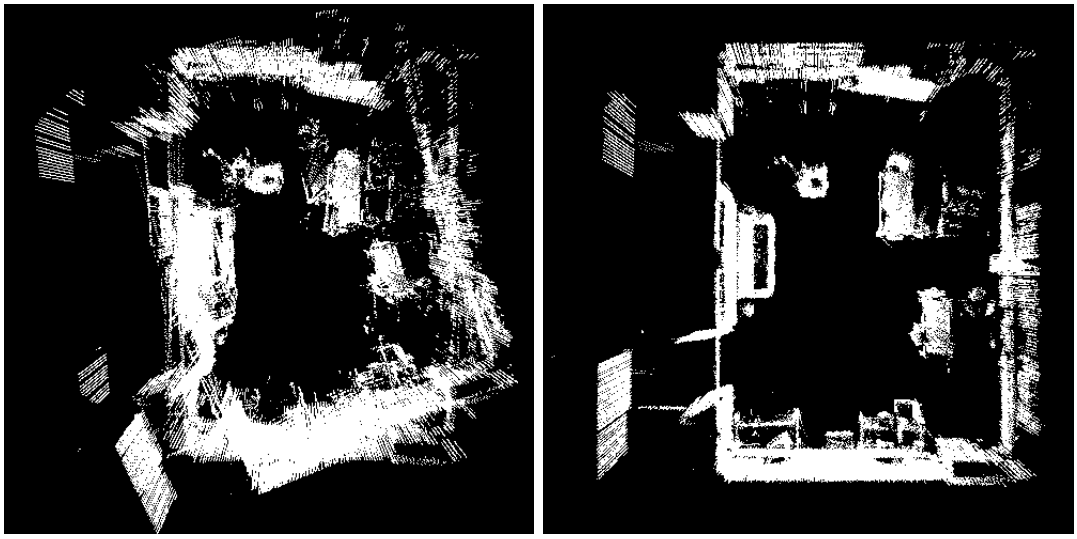


Figure 8.1: Final map for a room after 21 exploration steps. The left image shows the final map with robot poses given by the odometry of the robot. The right image shows the result obtained from the SLAM algorithm. The exploration system builds up the right result by integrating each new scan in each exploration step with the SLAM algorithm.

the exploration algorithm works as it should. We gain knowledge with every new scan and we can use this knowledge to expand the area that is accessible to the robot. The later extracted floor maps even open the path through the door in the lower left corner of the room. If we observed the exploration over more steps the robot would leave the room through this door.

The development of the extracted floor over the exploration steps also shows that the outlier filtering for the raw data works very well. A bad outlier filtering would yield red obstacle pixels in the images showing the extracted floor. We do not observe an increase of such unreal obstacles over the exploration steps. Of course, the robot moves in a static environment. If we had to deal with a dynamic environment with moving objects, people and so on, a more sophisticated approach had to be applied to overcome the problem of temporary obstacles. Otherwise, such obstacles would pollute the static map with temporary data. This would yield the same problems that come with a bad outlier filtering or a bad alignment of scan data.

After showing the extracted floor for the first 20 exploration steps we continue by showing the corresponding distance transform images for the extracted floor in figure 8.3. In these images we observe that small gaps in the extracted floor yield large holes in the accessible floor. This is due to the erosion of the floor by the robot radius. Although it would be possible to perform some preprocessing to artificially close these gaps, we find that this is not necessary. Due to the systematic exploration approach the gaps in the floor are naturally closed over time.

A comparison of the eroded distance transform images with the images of the extracted floor makes some requirements to the map quality obvious. Thin passages, like the door

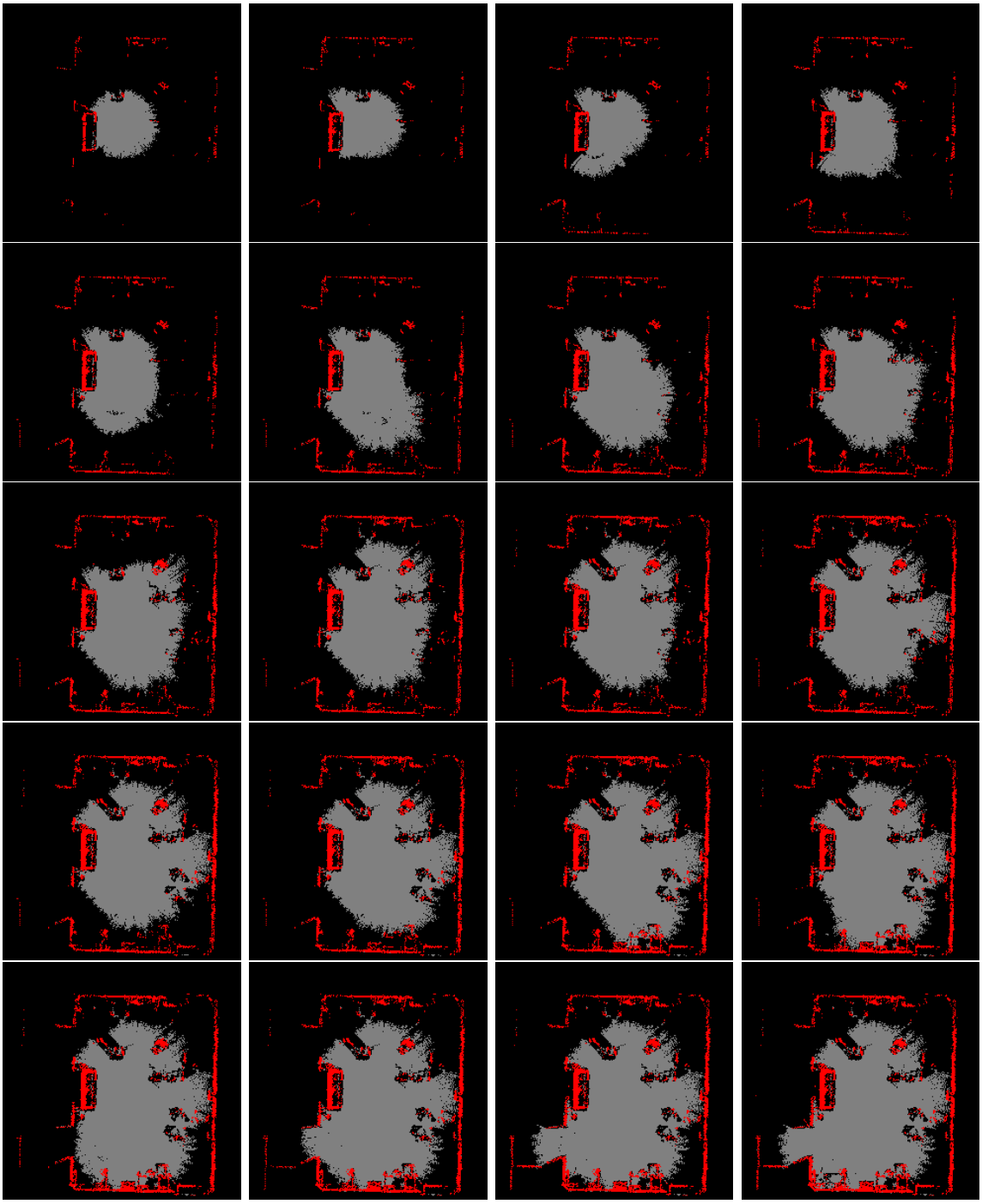


Figure 8.2: Extracted floor during exploration. The images show the extracted floor for 20 consecutive exploration steps. The floor is shown in gray. Black areas mark unexplored regions and obstacles are shown in red. The sequence goes from left to right and top to bottom. It is visible that the SLAM algorithm aligns new scans very well to the map.

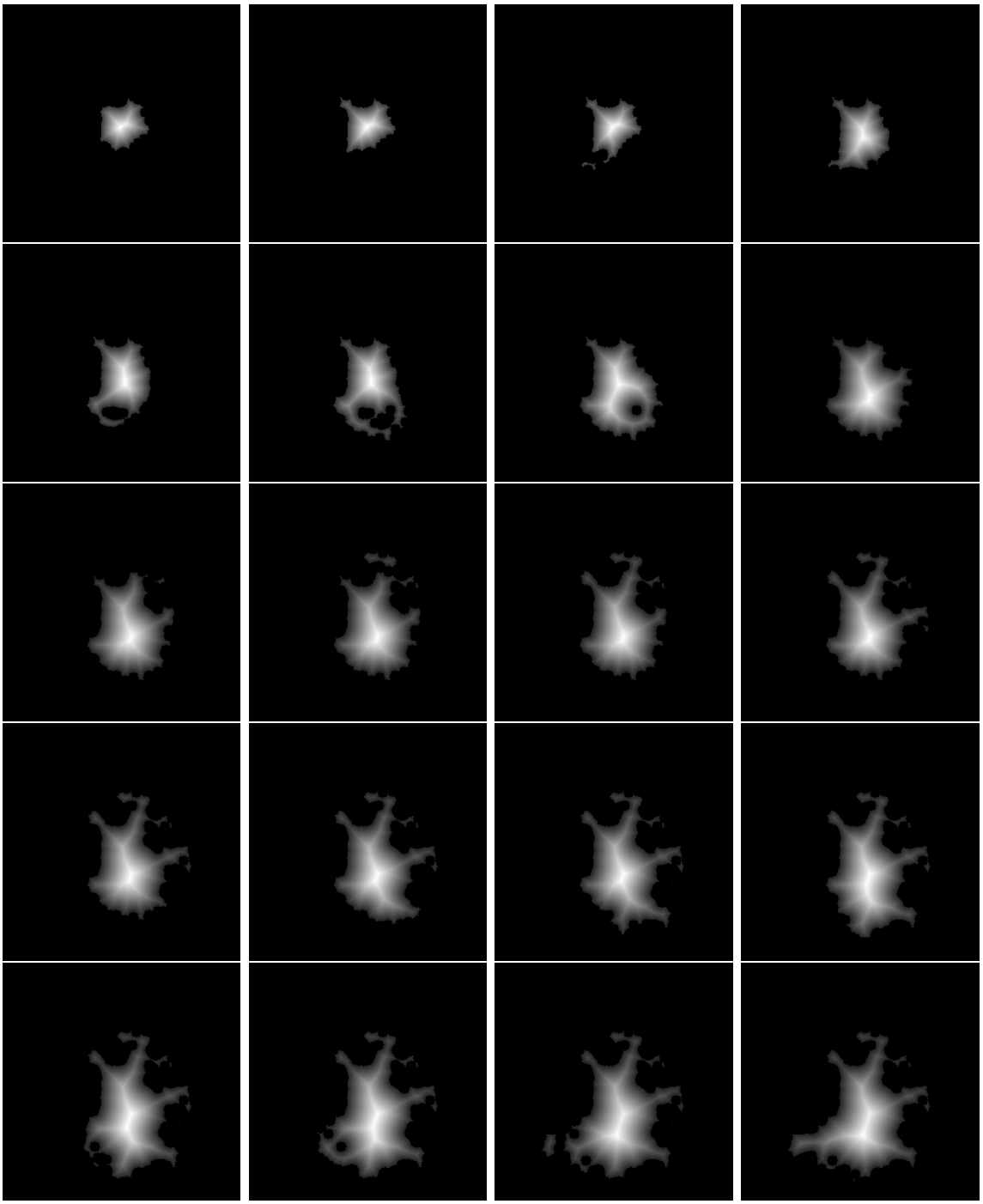


Figure 8.3: Distance transform of the extracted floor during exploration. The images show the distance transform for the first 20 exploration steps. The resulting floor is already eroded by the robot radius such that only the accessible area is visualized.

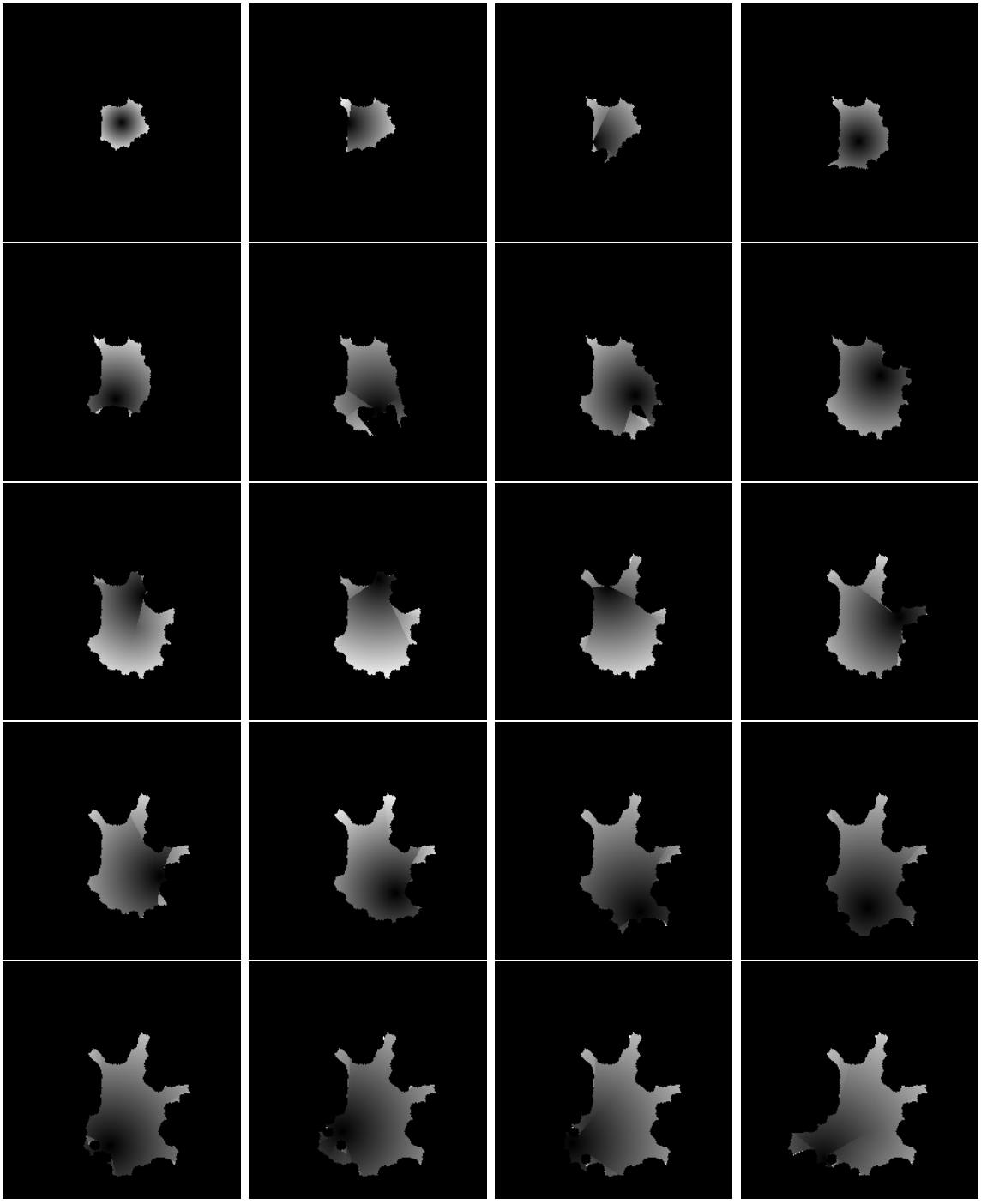


Figure 8.4: Cost map during exploration. The maps show the costs of moving to each position of the accessible floor for the first 20 exploration steps. Brighter pixels visualize high costs while darker pixels represent low costs. Black pixels also represent non-accessible areas.

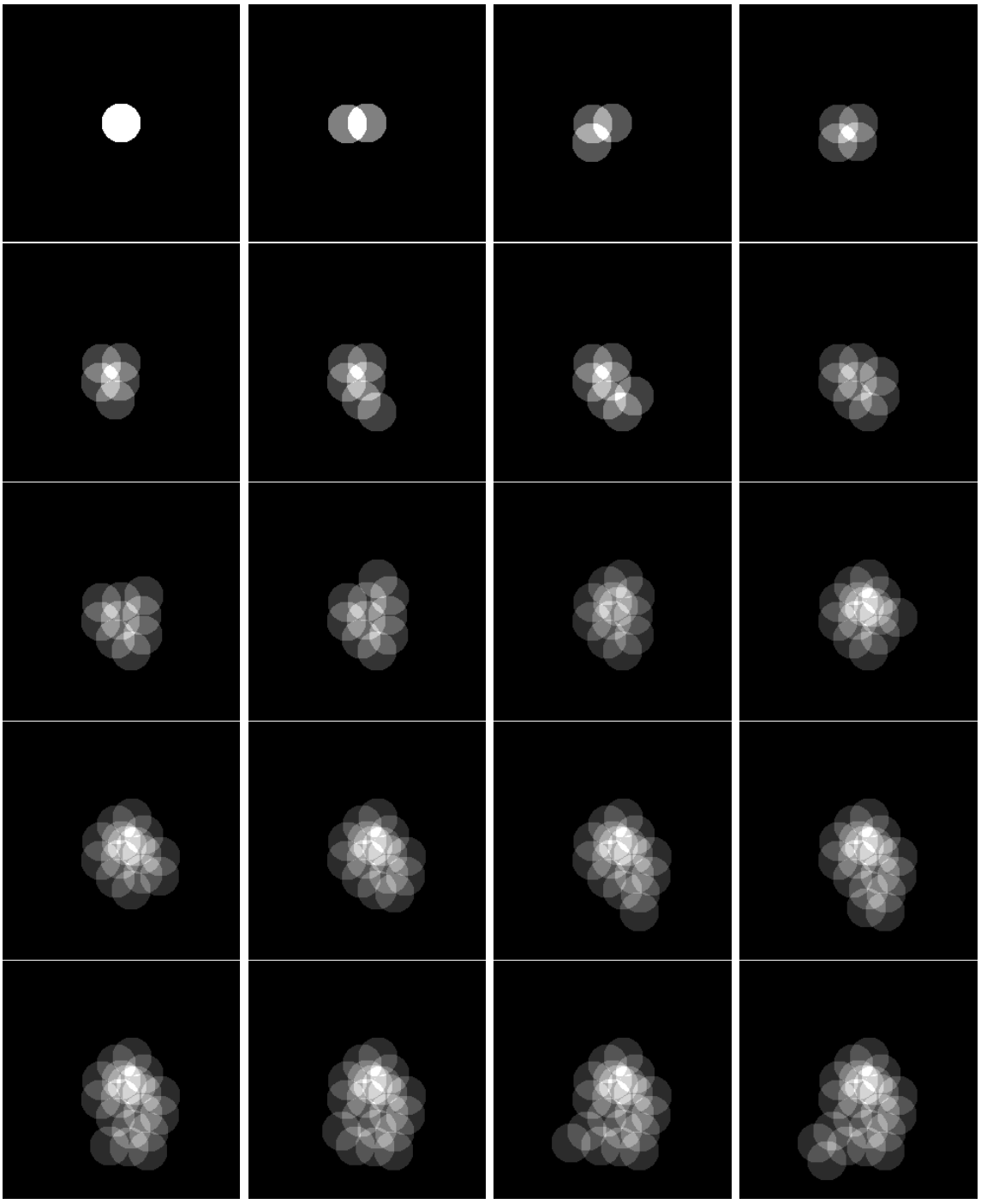


Figure 8.5: Knowledge map during exploration. The maps show the artificially defined knowledge for every location and each of the 20 exploration steps. Each scan generates knowledge within a radius of 1 meter. Note that the brightness that defines the grade of knowledge is not comparable between different images. This is only a measure within a given map.

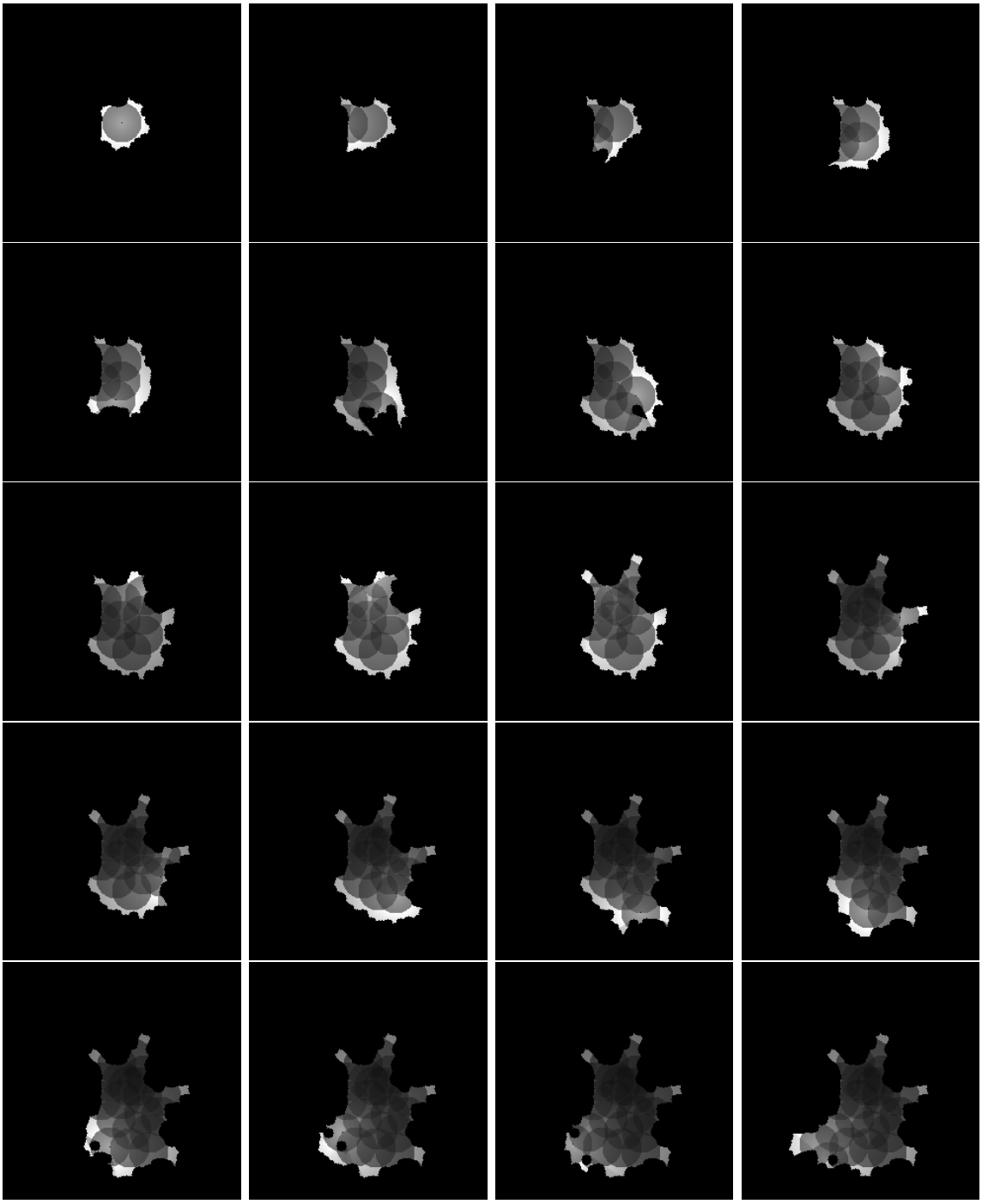


Figure 8.6: Next point evaluation map during exploration. The evaluation for possible next points is shown for the first 20 consecutive exploration steps. After the evaluation, the robot will move to the point that is marked with the brightest point in the maps. It is easily observable that the images combine the cost and knowledge maps.

on the lower left of the room, are made even thinner by the erosion of the floor. Therefore the scans have to be very well aligned to keep these passages open. A passage through a door might become closed if a scan was misaligned by perhaps 20 centimeters. The robot has very limited space to move through such passages. Obviously, the map quality in our case is good enough so that the thin passages stay open. This is an important property of maps used for exploration systems.

The next maps we observe for each exploration step are the cost maps which are shown in figure 8.4. In these images we can easily detect the current robot position. This position is given by a black spot and continuously increasing costs with increasing distance from this spot. We also see which locations can be reached in a direct movement and which positions are only accessible by moving over the artificially generated landmarks. The borders between such directly and indirectly accessible areas are marked by incontinuities in the cost function.

After taking the cost maps under scrutiny we also have to look at the knowledge maps to decide where the robot should move to. The knowledge maps for our example are shown in figure 8.5. In these maps we observe that each scan adds knowledge within the given *knowledge enhancement radius*. The locations for each scan are easily identifiable in the maps. We note that the scan locations are very well distributed. This is evidence for the good behavior of the exploration system. With each new scan the robot adds data on the frontier of the already known area.

Finally, we examine the evaluation maps that combine the cost and knowledge maps on the basis of equation 5.3. After looking at these maps it is very comprehensible where the robot will move to in the next step. Typically this will be a location in the near of the robot and on the accessible floor for which the robot has not yet collected a lot of knowledge.

8.2 Evaluating the final result

The final map produced by the exploration system is an accumulation of single laser range scans that are very well aligned to each other. We already showed that the alignment of the scans works very well. In this section we will have a more detailed look at the result. Of course, there are inaccuracies in each scan and the question is how these inaccuracies are integrated into the map as a whole.

We will first look at scanner artifacts. As already mentioned the scanner is not able to accurately measure the distances to highly reflecting surfaces or to strongly absorbing surfaces. Figure 8.7 shows the accumulation of inaccurate scanner data for a reflector on the leg of a table. It can be seen that each scan measures a distance to the reflector that is about 10 centimeters too short. The result for the accumulation of several scans is that we obtain a point cloud around the reflector instead of points that accurately represent the surface. Of course, these bad data points imply a locally bad quality of the map that also has an effect on the exploration system. For the exploration system, the accumulated scanner data looks like data from a pillar instead of data from a thin leg of a table. In

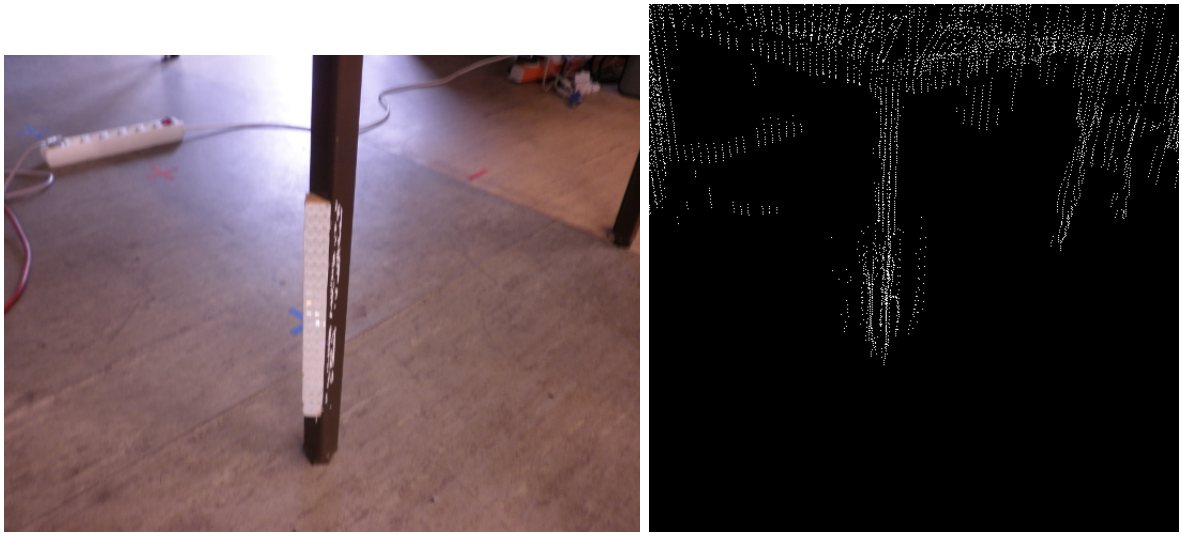


Figure 8.7: Accumulation of inaccuracies for highly reflective surfaces over several scans. The left image shows a reflector on the leg of a table, while the right image shows the corresponding part of the map consisting of 21 scans.

this example the accumulated inaccuracies are not critical, although if such inaccuracies would occur for the measurements of the distance to a door frame, the accessible floor might get eroded and close a passage through the door. The accumulated inaccuracies also have an effect on surface reconstruction techniques. Of course, no meaningful surface can be reconstructed for the reflector.

Another scanner artifact is the total mismeasurement of distances to strongly absorbing surfaces. An example for this can be seen in figure 8.8. Here, a LCD monitor on a table is shown. This monitor features a strongly absorbing surface and is not detected by any of the scans. This is an interesting observation, since one could assume that at least some scans produced acceptable data for such surfaces. This is not the case. There exist surfaces that are absolutely not detectable by the used laser range finder. In this case, the non-detectable surface is not problematic. However, we can imagine situations where non-detectable surfaces correspond to obstacles for the robot that are not detected. If, for some reason, there is a floor detected below such an object the exploration system might let the robot move through the object and produce a collision.

The accumulation of inaccurate and erroneous scanner data yields fundamental problems in the construction of a map. Although these problems are not critical for our purpose so far, a more sophisticated fusion of the scanner data is possible and desirable. For some scanner artifacts, the additional usage of another type of scanner and an integrated multi sensor data fusion also is a solution. This is, for example, the case for non-detectable surfaces.

Finally, we evaluate the surface reconstruction by ball pivoting for the map consisting of 21 scans. An example for a part of this reconstruction is given in figure 8.9. It can be seen that several details in the map are visible. This applies to chairs, monitors, tables and

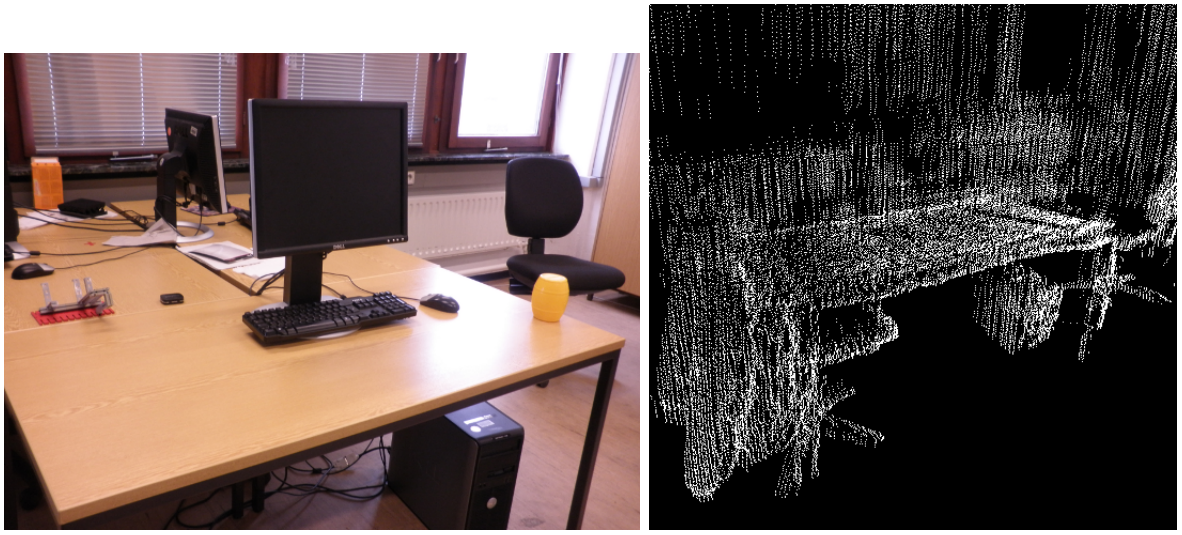


Figure 8.8: Non-detectable objects in multiple scans. The left image shows a LCD monitor on a table, while the right image shows the corresponding part of the map consisting of 21 scans. The monitor is not detected in any of the scans, independent of the angle etc. to the monitor.

Figure 8.9: Surface reconstruction by ball pivoting for a map consisting of 21 scans aligned with the ICP SLAM algorithm. Several details of the map, like chairs or monitors, are visible.

so on. Although, a comparison with the surface reconstruction for a single scan, like it is given in figure 7.6, reveals that the algorithm yields qualitatively better results for single, isolated scans. This is easily understandable since the map accumulates the inaccuracies of all scans. A triangulation for such a representation of the map always has the problem that it has to deal with all inaccuracies in all scans. This does not only apply to the Ball Pivoting algorithm but to all triangulation algorithms. To handle this problem, the scans had to be fused into the map in a more sophisticated way that would not just add all scan points but use these points to correct previous measurements. However, for the proof of concept that is demonstrated in this diploma thesis, the result of the reconstruction is very good.

8.3 Summary

In this chapter we showed the results for the exploration of a room. We demonstrated that the exploration system works the way it is supposed to work. It produces a very high quality map which is, of course, needed since the exploration system itself relies on the map. However, for other purposes the map quality may have to be even higher.

In section 8.1 we demonstrated the exploration process. It became clear that all the components of the exploration system work as they should and that they form a good exploration system as a whole. Finally, we took a critical and detailed look at the final result of the map in section 8.2. We saw how scanner inaccuracies accumulate in the map and found that this is a challenge for visualization purposes.

Chapter 9

Summary and Outlook

In this diploma thesis we developed an autonomous integrated mapping and exploration system. On the hardware side, this system consisted of a robot on which a laser range finder with a rotation platform was mounted. The physical components of the system allowed us to take 3D scans of the whole space from each robot pose. However, the focus of this work was the development of the system's software components. The software system navigated the robot to locations from which it was supposed to take scans of the environment. It then took the scan data to build up a 3D map by applying the ICP SLAM algorithm. Finally, the map was used to determine the next scan location for the robot. Each component of the system was able to run on its own and the system as a whole formed a distributed software architecture.

We motivated the work and gave a general overview of the system in the introductory chapter 1. Next, we proceeded by introducing the theoretical background of the SLAM problem in chapter 2. In this chapter we also motivated the choice of the ICP SLAM algorithm for the exploration system. Afterwards, the physical components of the system were described in chapter 3. The consecutive chapters introduced the software components. This started with a detailed discussion of the ICP SLAM algorithm and its implementation in chapter 4. The second software component we developed was the exploration program in chapter 5. In this chapter we described how the floor was extracted from the scan data, how we set landmarks on this floor, and how we choose the next robot location and move there. Chapter 6 then introduced the simulation program for the robot. This component was interchangeable with the program that connected to the real robot. After presenting the simulation program we also presented the visualization program for the map in chapter 7. This program visualized the map in two ways. The first visualization was a simple representation of the sampled scan points. The second visualization showed the environment's surface reconstructed by the Ball Pivoting algorithm. We discussed this algorithm in detail. Finally, we discussed the results for the exploration of a room in chapter 8. This discussion revealed that the exploration system as a whole worked very well. However, we also discussed the disadvantages of our approach.

Of course, an outlook on possible consecutive developments of the system has to address the disadvantages of the current approach. These disadvantages were primarily visible in

the visualization of the map and had the potential to affect the exploration algorithm. We mentioned that the inaccuracies of all scans accumulate in the map. This is a problem that we can address with another representation of the map. At the moment, the map is just a set of robot locations with associated scan data. However, it is possible to replace this representation by a so-called occupancy grid map [3]. Such a map subdivides the space into a regular grid and stores in each grid cell the probability that the cell is occupied by matter. Occupancy Grid Mapping has several important advantages. First of all, the inaccuracies do not accumulate, instead the opposite is the case. Multiple scans in the same region reduce the inaccuracies in the map. Occupancy Grid Mapping also suppresses wrong data in the map whenever newer scans indicate that this data is wrong. This is an important property if dynamic environments are to be explored. Note that we hold two maps of the environment in the exploration system. One map in maintained by the SLAM program and another map is maintained by the exploration program. Since the SLAM part of the system works very well, we can stick to the present map in this component. However, we have the option to replace the ICP SLAM algorithm by a 3D version of FastSLAM with Occupancy Grid Mapping. It is possible to just replace the present map by an occupancy grid map in the exploration component and use this map also for visualization purposes.

In principle, an occupancy grid map offers many options for the visualization. First of all, *Direct Volume Rendering* is a large field that features many techniques to visualize such a map. If a surface reconstruction is required, the occupancy grid map may also be preprocessed on the basis of the Marching Cubes Algorithm [34].

Another improvement can be applied to the scan acquisition process. We mentioned that the outlier filtering works very well. This is true, but on the other side, the preprocessing of the scan data on the microcontroller board, as well as the outlier filtering, reduce the number of valid data points to two thirds the number of the original scan points. Here, it is possible to replace the error filtering mechanisms with error correction mechanisms. Such an improvement would not drastically affect the exploration, but it may have an effect on the results of the visualization.

Finally, an obvious improvement of the system would be the extension of the approach to uneven environments in which the robot has 6 degrees of freedom. However, this is a complicated task since the floor extraction of the exploration component, as well as the path planning, are designed to work on flat environments.

Bibliography

[1] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[2] The DARPA Grand Challenge. *http://www.darpa.mil/grandchallenge/index.asp*.

[3] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press.

[4] M. Montemerlo and S. Thrun. *FastSLAM - A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*. Springer.

[5] A. Nüchter and H. Surmann. 6D SLAM - 3D Mapping Outdoor Environments. *Journal of Field Robotics*, 24(8,9):699–722, 2007.

[6] H. Surmann, A. Nüchter, and J. Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45:181–198, 2003.

[7] Mobile Robots Inc. *http://www.mobilerobots.com*.

[8] P. Breuer. *Entwicklung eines Systems zur 3D-Rekonstruktion der dynamischen Umgebung mit einem rotierenden 2D-Laserscanner*. University of Hamburg, Diploma Thesis in Mechatronics.

[9] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press.

[10] *Scanning Laser Range Finder URG-04LX Specifications*. Hokuyo Automatic CO. LTD.

[11] The Player Project. *http://playerstage.sourceforge.net/*.

[12] Boost C++ Libraries. *http://www.boost.org/*.

[13] P. Besl and N. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[14] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[15] A. Nüchter. *3D Robotic Mapping - The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*. Springer.

[16] H. Eggers. Two fast Euclidean distance transformations in $\mathbb{Z}^2$ based on sufficient propagation. *Computer Vision and Image Understanding*, 69(1):106–116, 1998.

[17] R. Lotufo and F. A. Zampirolli. Fast multi-dimensional parallel Euclidean distance transform based on mathematical morphology. *Proccedings of SIBGRAPI, XIV Brazilian Symposium on Computer Graphics and Image Processing, IEEE Computer Society*, pages 100–105, 2001.

[18] C. Maurer, R. Qi, and V. Raghavan. A linear time algorithm for computing the Euclidean distance transform in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, 2003.

[19] A. Meijster, J. Roerdink, and W. Hesselink. A general algorithm for computing distance transforms in linear time. *Proceedings of the 5th International Conference on Mathematical Morphology and its Applications to Image and Signal Processing*, 2000.

[20] T. Saito and J. Toriwaki. Newalgorithms for Euclidean distance transformations of an n-dimensional digitised picture with applications. *Pattern Recognition*, 27(11):1551–1565, 1994.

[21] R. Fabbri, L. da Fontoura Costa, J. C. Torelli, and O. M. Bruno. 2D Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys (CSUR)*, 40(1), 2008.

[22] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. Cengage Learning Services, 1998.

[23] L. Vincent. Efficient computation of various types of skeletons. *Proceedings of SPIE*, 1445:297–311, 1991.

[24] L. Vincent. *Algorithmes Morphologiques a Base de Files d'Attente et de Lacets Extension aux Graphes*. l'Ecole Nationale Supérieure des Mines de Paris, These pour obtenir le titre de Docteur en Morphologie Mathematique, 1990.

[25] STL (file format). *http://en.wikipedia.org/wiki/STL_(file_format)*.

[26] Blender. *http://www.blender.org*.

[27] OpenGL - The Industry Standard for High Performance Graphics. *http://www.opengl.org*.

[28] F. Bernardini, J. Mittleman, H. Rushmeir, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[29] N. Amenta, S. Choi, and R. K. Kolluri. The Power Crust. *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, 2001.

[30] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface Reconstruction from Unorganized Points. *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, 1992.

[31] K. Pulli, T. Duchamp, H. Hoppe, J. McDonald, L. Shapiro, and W. Stuetzle. Robust Meshes from Multiple Range Maps. *International Conference on Recent Advances in 3-D Digital Imaging and Modeling, 1997. Proceedings*, pages 205–211, 1997.

[32] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[33] G. Turk and M. Levoy. Zippered Polygon Meshes from Range Images. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, 1994.

[34] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

Danksagung

Hiermit bedanke ich mich bei allen Personen, die mir bei der Anfertigung dieser Diplomarbeit mit Hilfe und Rat zur Seite standen. Dies sind insbesondere...

- ...Prof. Dr. Jianwei Zhang, der es mir ermöglicht hat, diese Arbeit anzufertigen und Erstgutachter der Arbeit ist.
- ...Dr. Werner Hansmann, der sich dazu bereiterklärt hat, diese Arbeit als Zweitgutachter zu bewerten.
- ...Denis Klimentjew, der mich bei der Anfertigung der Arbeit sehr gut betreut hat und mir viele Ideen und Anregungen gab.
- ...Ph.D. Houxiang Zhang, der mir ebenfalls viele Anregungen gegeben hat.
- ...Peter Breuer für die ergiebige Zusammenarbeit und viele interessante Anregungen.
- ...alle weiteren Mitglieder der TAMS Arbeitsgruppe des Fachbereichs Informatik der Universität Hamburg.
- ...meine Familie, sowie Qionghua Ge, die mir den benötigten Halt gegeben haben.
- ...Christoph Dittmann, der mich an seinen Englisch- und LaTeX-Kenntnissen teilhaben ließ.

Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig und ausschließlich mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben.

Ich erkläre weiterhin, dass ich mit einer Veröffentlichung meiner Arbeit in der Bibliothek einverstanden bin.

Hamburg, den 4. Oktober 2010

Gregor Michalicek