A Scalable, Platform-Independent SLAM System for Urban Search and Rescue

Ein skalierbares, plattformunabhängiges SLAM System für urbane Ortungs- und Rettungseinsätze (USAR)

Diploma Thesis by Stefan Kohlbrecher

October 2009

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Simulation, Systemoptimierung und Robotik

A Scalable, Platform-Independent SLAM System for Urban Search and Rescue
Ein skalierbares, plattformunabhängiges SLAM System für urbane Ortungs- und Rettungseinsätze (USAR)

vorgelegte Diplomarbeit von Stefan Kohlbrecher

Prüfer: Prof. Dr. Oskar von Stryk
Betreuer: Dipl. Math. Karen Petersen

Tag der Einreichung:

Erklärung zur Diplomarbeit

Hiermit versichere ich die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 27. Oktober 2009

(S. Kohlbrecher)

Abstract

Robot systems see increasing use outside the field of automation engineering, where they were first introduced. To solve complex tasks in unknown environments, autonomous systems need a model of their environment. An example of such a complex task are Urban Search and Rescue (USAR) scenarios.
In the scope of this work, a platform independent and scalable system for simultaneous localization and mapping (SLAM) for autonomous and semiautonomous robots was developed, enabling these platforms to generate 2D maps of USAR and other environments using a laser scanner.
A particle filter and scan registration approach were implemented. Experiments show that a combination of both approaches yields the best results.
The system described in this work was deployed successfully on an Unmanned Ground Vehicle (UGV) in the Robot Rescue League of the international robotics competition RoboCup 2009.

Kurzzusammenfassung

Robotikysteme werden zunehmend auch ausserhalb der Automatisierungstechnik eingesetzt, wo sie zuerst eingeführt wurden. Um komplexe Aufgaben in unbekannten Umgebungen zu erfüllen, müssen autonome Systeme über ein Modell ihrer Umgebung verfügen. Ein Beispiel für eine solche Aufgabe sind urbane Ortungs- und Rettungseinsätze (eng. USAR - Urban Search and Rescue).
Im Rahmen dieser Arbeit wurde ein plattformunabhängiges und mit der verfügbaren Rechenzeit skalierendes System entwickelt, das einem autonomen oder teilautonomen Roboter unter Benutzung eines Laserscanners ermöglicht, eine zweidimensionale Karte seiner Umgebung zu erstellen und sich in dieser zu lokalisieren (eng. SLAM - Simultaneous Localization and Mapping). Sowohl ein Partikelfilter- wie ein Scanregistrierungs-basiertes Verfahren wurden implementiert. In Experimenten wird gezeigt dass eine Kombination beider Ansätze die besten Ergebnisse erzielt. Das beschriebene System wurde auf einem autonomen Fahrzeug erfolgreich in der Rettungsliga des internationalen Robotikwettbewerbs RoboCup 2009 eingesetzt.

Contents

1 Introduction

1.1 Motivation

Robots see increasing use outside of automation, where they were first introduced. Autonomous robots need to have a consistent model of the environment surrounding them to not be limited to simple reactive behaviors but perform tasks that involve efficient navigation and planning. They will increasingly be deployed for missions deemed too dangerous or even impossible for humans. In many scenarios, a map of the future area of operations of the robot cannot be supplied. An example are USAR (Urban Search and Rescue) scenarios where a robot has to enter and search partially collapsed buildings of unknown geometry.
It is therefore important to enable the robot to create a map of the environment while simultaneously localizing itself in this map. This is commonly referred to as the Simultaneous Localization and Mapping (SLAM) problem and the topic of this work. A SLAM approach is of potential use for a multitude of platforms, so a platform independent approach that scales well with the amount of processing power available is desirable.

1.2 The SLAM Problem

In Simultaneous Localization and Mapping, the robot has to estimate the own position in a map it is building of the environment using internal and external sensors. The difficulty lies in the fact that for localization a map is needed, so sensor data can be compared to that map. On the other hand, to build a consistent map, the robot position has to be known, so sensor data can be integrated into the map consistently. This is an instance of what casually might be called a "chicken and egg" problem and it obviously is not possible to satisfy both requirements easily at the same time, so care has to be taken to integrate noisy system dynamics and sensor measurements in a way that minimizes the overall error.
Revisiting locations already mapped is termed loop closing and gives the opportunity to correct the error built up in a loop. When loop closing fails however, inconsistent maps are often the result. This is especially the case with large scale environments, where the robot can travel long distances and thus build up large errors before encountering a chance for loop closure.

1.3 Requirements

The aim of this work is to provide a SLAM solution for Urban Search and Rescue environments, enabling autonomous systems to perform their mission. Autonomous robotic systems at the TU Darmstadt include wheeled and legged robots, as well as flying and swimming platforms. Two examples are depicted in figure 1.1. These systems vary greatly in the computational power available, so for a future adaptation to different platforms, the system has to be scalable and be able to give meaningful and correct results even if only low computational power is available.
The widespread used sensor type for SLAM applications are laser scanners as they offer fast and

(a) (b)

Figure 1.1: Examples of Autonomous systems used at TU Darmstadt: (a) "Monstertuck" UGV, (b) Quadrotor UAV

very precise measurements of the environment. For this reason, they are adopted as the SLAM sensor for this work.
As the choice of operating system for an autonomous system often cannot be made dependent on the submodules used (in this case the SLAM implementation) the system has to be platform independent, so it can be adapted to platforms where free choice of Operations System is not an option.
Finally, the system has to be able to consistently map USAR scenarios to enable high level behavior like route planning and mapping of victims. It is important that it can be seamlessly integrated into an existing modelling and behavior specification framework.

2 Foundations

2.1 Notation

Throughout this work, the following notation will be used:

x_t		State vector at time t
z_t		Measurement vector at time t
u_t		Control vector at time t
$x_{0:t}$	$:=$	$x_0, x_1, ..., x_t$
$z_{0:t}$	$:=$	$z_0, z_1, ..., z_t$
$u_{0:t}$	$:=$	$u_0, u_1, ..., u_t$
m_t		Map vector at time t
$p(A)$		Probability of A
$p(A,B)$		Probability of A and B
$p(A\|B)$		Conditional Probability of A given B
η		Normalizer for Bayes Rule
P_t		State covariance at time t
Q_t		Process noise covariance at time t
R_t		Measurement noise covariance at time t
A_t		State transition model at time t
B_t		Control-input model at time t
H_t		Observation model at time t
K_t		Kalman Gain at time t
χ_t		Particle Set at time t
${x_t}^{[i]}$		Particle i from set χ_t at time t
W_t		Particle Weight Set at time t
${w_t}^{[i]}$		Particle weight i from set W_t at time t
${}^c v$		Vector given in coordinate system c
${}^a T_c$		Homogeneous Transformation matrix from coordinate system c to a

2.2 Coordinate Systems and Transformations

As data is given in different cartesian coordinate systems, the coordinate systems and their relations have to be defined. All coordinate systems observe the right-hand rule. Figure 2.1 provides an overview:

- C_w is the world coordinate system and global reference frame. For the purpose of this work, the origin of C_w is placed at the robot position on robot startup.
- C_m is the map coordinate system. It is is fixed relative to the world coordinate system.
- C_r is the robot coordinate system. The robot coordinate system is fixed at the robot and thus moves relative to the world coordinate system when the robot moves.

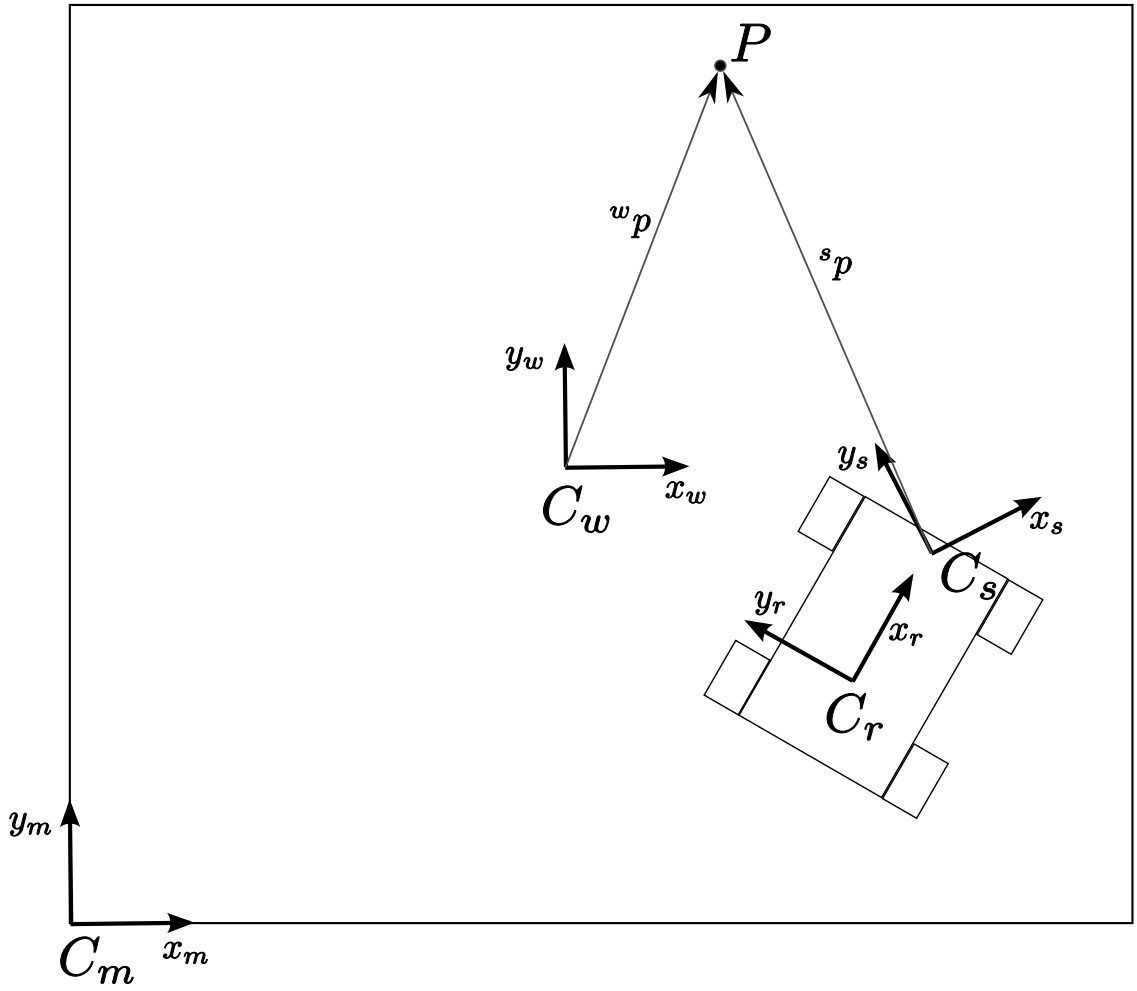


Figure 2.1: Overview of the different coordinate systems used

- C_s is the sensor coordinate system. This is the coordinate system sensor data arrives in.

As an example, the sensor that belongs to C_s senses point P. The measurement is given in the sensor coordinate system as vector ^sp. If we are interested in ^wp, which amounts to the world coordinates of P, we have to use the following transformation:

$$^wp = {}^wT_r\,{}^rT_s\,{}^sp \tag{2.1}$$

Transformations are generally written as homogeneous transformations for clarity.

2.3 Sequential State Estimation

Estimating the robot position and the map of the environment can be modeled as a sequential state estimation problem. While the robot moves through the environment, observations of the state are gathered via sensors and can be integrated into the state estimate. In recent years, probabilistic approaches to solving this problem have become state of the art, as modeling of uncertainty makes these methods very robust. A short overview of the recursive bayesian state estimation techniques will be given in the following section. The interested reader might refer to [1] for a more thorough treatment including proofs.

2.3.1 Bayes Filter

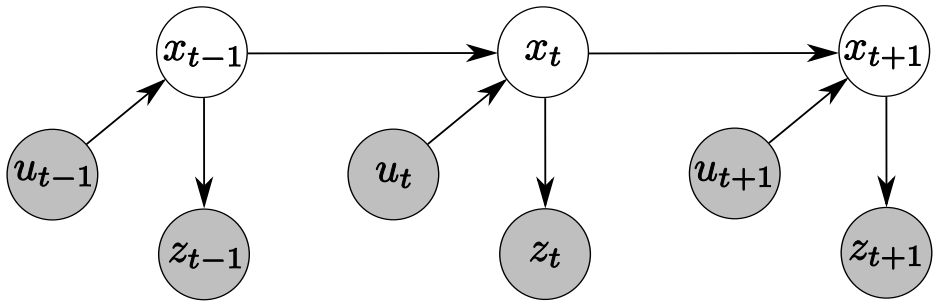


Figure 2.2: Dynamic Bayes network showing the relationship between states x, controls u and observations z. The light grey color indicates controls and observations are known, while the state has to be inferred.

In a system governed by probabilistic laws, the evolution of the state x_t generally might be described by a generative model taking all previous states $x_{0:t-1}$, measurements $z_{1:t-1}$ as well as controls $u_{1:t}$ into account:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) \tag{2.2}$$

For most real world problems, this state estimation problem is intractable. However, assuming the Markov assumption to hold, the problem can be simplified to be a sequential state estimation problem where states are only conditionally dependent on the preceding state instead of all earlier states. This is equal to assuming the state estimation x_t for a given time t to be complete, e.g. to contain all information that is available of the system at that point in time. The Markov assumption is a very strong one and is often violated in practice. Nevertheless, careful probabilistic formulations can be quite robust against these violations. With the Markov assumption we have:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.3}$$

Conditional independence between states also implies that the measurement model $p(z_t)$ is only dependent on the current state:

$$p(z_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.4}$$

Figure 2.2 shows a graphical model of the relationships with the Markov assumption taken into account. It remains to be shown how a sequential filter for state estimation can be inferred. Using Bayes rule we have:

$$\begin{aligned} p(x_t|z_{1:t}, u_{1:t}) &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t}) p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t}) p(x_t|z_{1:t-1}, u_{1:t}) \end{aligned} \tag{2.5}$$

with the Markov assumption follows:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t) p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.6}$$

Using the posterior belief:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.7}$$

and the prior belief:

$$\overline{bel(x_t)} = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.8}$$

Equation 2.6 can be rewritten as:

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel(x_t)} \tag{2.9}$$

It remains to simplify $\overline{bel(x_t)}$:

$$\begin{aligned}
\overline{bel(x_t)} =& p(x_t|z_{1:t-1}, u_{1:t}) \\
=& \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1}, z_{1:t-1}, u_{1:t}) dx_{t-1} \\
=& \int p(x_t|x_{t-1}, u_t) p(x_{t-1}, u_t) dx_{t-1} \\
=& \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1}
\end{aligned} \tag{2.10}$$

Equation 2.10 is the Bayes filter prediction equation and 2.9 is the update equation. The recursive nature becomes clear: The prior belief $\overline{bel(x_t)}$is calculated from the posterior belief $bel(x_{t-1})$ of the preceding time step while the integration of the measurement model $p(z_t|x_t)$ results in the posterior belief $bel(x_t)$.

2.3.2 Kalman Filter

The Kalman filter is a recursive stochastic filter that estimates the mean and covariance of a continuous linear dynamic system from noisy measurements. It is a tractable instance of the Bayes filter and has a closed form solution, at the cost of only supporting linear systems as well as unimodal, gaussian distributions. The family of Kalman filter algorithms is the de facto standard in state estimation for a wide range of different applications such as aerospace engineering [2], computer vision [3] and robotics. A comprehensive introduction might be found in [4].
In the Kalman filter belief is represented by the mean of the estimated state vector $\hat{x}_t$ as well as the corresponding covariance P_t.
The filter alternates a prediction step, in which the estimated system state is projected forward in time according to the system dynamics and a measurement update step which updates the forward projected state by incorporating a new measurement. In the prediction step the predicted state $\overline{\hat{x}_t}$ is determined by:

$$\overline{\hat{x}_t} = A_t \hat{x}_{t-1} + B_t u_t \tag{2.11}$$

Here, A_t ist the linear system model that relates the previous state to the current one and B_t is the linear control model that describes the influence of the control vector on the predicted state. The covariance of the filter is also projected forward in time:

$$\overline{P_t} = A_t P_{t-1} {A_t}^T + Q_t \tag{2.12}$$

with Q_t being the process noise covariance of the system.
For the measurement update, the Kalman gain is determined:

$$K_t = \overline{P_t} {H_t}^T (H_t \overline{P_t} {H_t}^T + R_t)^{-1} \tag{2.13}$$

H_t is the linear measurement model, while R_t ist the measurement noise covariance.
The new state estimate is then

$$\hat{x}_t = \hat{x}_t + K_t(z_t - H_t \overline{\hat{x}_t}) \tag{2.14}$$

with covariance

$$P_t = (I - K_t H_t)\overline{P_t} \tag{2.15}$$

Algorithm 1 states an overview of the Kalman filter.
The unimodal Kalman filter as well as the derivatives mentioned below can be modified to support multiple hypotheses by means of incorporating not only one, but multiple weighted gaussian state estimates and covariances. This extension is known as the multi-hypothesis Kalman filter and widely used for tracking applications [5].

Algorithm 1 Kalman Filter$(\hat{x}_{t-1}, P_{t-1}, u_t, z_t)$

1: $\overline{\hat{x}_t} = A_t \hat{x}_{t-1} + B_t u_t$
2: $\overline{P_t} = A_t P_{t-1} {A_t}^T + Q_t$
3: $K_t = \overline{P_t} {H_t}^T (H_t \overline{P_t} {H_t}^T + R_t)^{-1}$
4: $\hat{x}_t = \hat{x}_t + K_t(z_t - H_t \overline{\hat{x}_t})$
5: $P_t = (I - K_t H_t)\overline{P_t}$
6: **return** $\hat{x}_t, P_t$

2.3.3 Extended Kalman Filter

The Extended Kalman Filter (EKF) is a modification to the linear Kalman filter that supports nonlinear systems. Instead of matrices and thus linear relationships for the process model and measurement model functions, the EKF uses

$$x_t = f_t(x_{t-1}, u_t) \tag{2.16}$$

as the (nonlinear) process model function and

$$z_t = h_t(x_t) \tag{2.17}$$

as the (nonlinear) measurement model function. The linear transformation of the gaussian estimates used in the Kalman filter generally is not possible for these nonlinear functions. For this reason, by Taylor expansion and dropping of higher order terms, linear approximations are used:

$$A_t = \frac{\partial f_t}{\partial x_{t-1}}(x_{t-1}, u_t) \tag{2.18}$$

is the Jacobian of the process model function with regards to x_{t-1} and

$$H_t = \frac{\partial h_t}{\partial x_t}(x_t) \tag{2.19}$$

is the Jacobian of the measurement model function with regards to x_t.
Using these approximations the prediction step in the EKF amounts to equations 2.16 and 2.12. The measurement update is similar to that of the linear Kalman filter, only the measurement state update changes to:

$$\hat{x}_t = \hat{x}_t + K_t(z_t - h(\overline{\hat{x}_t})) \tag{2.20}$$

Algorithm 2 states an overview of the extended Kalman filter.
Due to the linear approximation of nonlinear functions, the EKF is not an optimal estimator, unlike the linear Kalman filter. Linearization error can lead to divergence of the filter if it is too large, so care has to be taken to keep the state estimate reasonably close to the true state, as to keep linearization errors small.

Algorithm 2 Extended Kalman filter($\hat{x}_{t-1}, P_{t-1}, u_t, z_t$)

1: $\overline{\hat{x}_t} = f(\hat{x}_{t-1}, u_t)$
2: $\overline{P_t} = A_t P_{t-1} {A_t}^T + Q_t$
3: $K_t = \overline{P_t} {H_t}^T (H_t \overline{P_t} {H_t}^T + R_t)^{-1}$
4: $\hat{x}_t = \hat{x}_t + K_t(z_t - h(\overline{\hat{x}_t}))$
5: $P_t = (I - K_t H_t)\overline{P_t}$
6: **return** x_t, P_t

2.3.4 Unscented Kalman Filter

The Unscented Kalman Filter (UKF) is another variant of the Kalman filter for nonlinear systems. Instead of using Linearization via Taylor Expansion, the UKF samples the involved distributions around the mean using deterministic sample points commonly called sigma points. For highly nonlinear systems it gives better results than the EKF as it approximates nonlinearities up to second order unlike the EKF which uses a first order approximation [6].

2.3.5 Particle Filter

The Particle Filter is a nonparametric, discrete variant of the Bayes filter. Particle filters are a relatively recent approach to Bayesian State Estimation [7] compared to the Kalman filter, but have quickly gained importance for state estimation in robotics [8]. In contrast to Kalman filters, particle filters are not limited to parametric, gaussian distributions. The estimated density is represented by a discrete number N of particles, considered to be drawn from the density. This way, arbitrary shapes of distributions can be represented, only limited by the number of particles and the computational cost associated with computing the particle set.
Each particle ${x_t}^{[i]}$ can be thought of as a distinct hypothesis about the system state that gets moved by a model of the system dynamics over time (generally odometry in UGVs) and then gets

weighted depending on the degree to which it fits a measurement. Depending on the weight, the particle might be purged from the particle set if it has low weight, or it might get copied and generate offspring if it has high weight. Viewing the particle filter as a "survival of the fittest" approach is intuitive, even if simplistic compared to the probabilistic derivation.
Particle filters have anytime characteristics, meaning they do not have fixed computation time, but their accuracy increases with more computation time (and thus particles) allocated to them. Of course, depending on the application, too small particle sets will not give meaningful results.
Algorithm 3 shows a basic implementation of the particle filter. In line 3, the prediction step samples N particles from the proposal distribution determined by state x_{t-1} and the current control u_t. In line 4, the particle importance weights ${w_t}^{[m]}$ are determined by evaluating the observation model.
The resampling step starting in line 7 then draws particles proportional to their weight from $\overline{\chi_t}$ and adds them to χ_t. Particles with high weight are more likely to be copied into the new particle set than those with low weights, resulting in the particle set better resembling the target distribution.
After the resampling step, the new particle set χ_t representing the robot belief is returned.

Algorithm 3 Particle Filter (χ_{t-1}, u_t, z_t)

1: $\overline{\chi_t} = \chi_t = \emptyset$
2: **for** $m = 1$ to N **do**
3: sample $x_t^{[m]} \sim p(x_t|u_t, {x_{t-1}}^{[m]})$
4: ${w_t}^{[m]} = p(z_t|{x_t}^{[m]})$
5: $\overline{\chi_t} = \overline{\chi_t} + \langle {x_t}^{[m]}, {w_t}^{[m]} \rangle$
6: **end for**
7: **for** $m = 1$ to N **do**
8: draw i with probability $\propto {w_t}^{[i]}$
9: add ${x_t}^{[i]}$ to χ_t
10: **end for**
11: **return** χ_t

2.4 Occupancy Grid Mapping

Occupancy grid maps were introduced in [9] called evidence grids in that context. Occupancy grid mapping is a metric mapping technique. The environment is represented by a regular grid, with each grid cell containing a binary random variable indicating the probability of occupancy of that cell. The goal of occupancy grid mapping is estimating the posterior:

$$p(m|x_{1:t}, z_{1:t}) \tag{2.21}$$

Even for maps with few grid cells estimating this posterior is computationally intractable. For this reason, with the assumption of independence between grid cells we have:

$$p(m_i|x_{1:t}, z_{1:t}) \tag{2.22}$$

With conditional independence between grid cells, the problem is tractable. Occupancy grid are commonly modelled using the two main approaches described below.

2.4.1 Log Odds Model Grid Maps

The occupancy of the single grid cells can be estimated by means of a binary Bayes filter. It is convenient to use the log odds representation of occupancy probability, as numerical instabilities are averted and the update can be computed efficiently by a sum. The log odds representation of occupancy for grid cell m_i is:

$$l_t(m_i) = \log \frac{p(m_i|x_{1:t}, z_{1:t})}{1 - p(m_i|x_{1:t}, z_{1:t})} \tag{2.23}$$

The occupancy probability can be recovered from the log odds ratio:

$$p(m_i) = 1 - \frac{1}{1 + \exp(l(m_i))} \tag{2.24}$$

Considering the prior probability of occupancy to be 0.5, which is an assumption that is adopted throughout this work, the update of a grid cell amounts to:

$$\begin{aligned} l_t(m_i) &= \log \frac{p(m_i|x_t, z_t)}{1 - p(m_i|x_t, z_t)} + \log \frac{p(m_i|x_{1:t-1}, z_{1:t-1})}{1 - p(m_i|x_{1:t-1}, z_{1:t-1})} \\ &= \log \frac{p(m_i|x_t, z_t)}{1 - p(m_i|x_t, z_t)} + l_{t-1}(m_i) \end{aligned} \tag{2.25}$$

2.4.2 Reflectance Count Grid Maps

With the counting model, the grid cell occupancy probability $p(m_i)$ is determined by counting the instances c_{occ} of a ray ending in grid cell m_i as well as counting the instances c_{free} of a beam covering m_i without ending in it. The ratio

$$r = \frac{c_{occ}}{c_{occ} + c_{free}} \tag{2.26}$$

is then an estimate of the probability that a beam will be reflected when visiting the grid cell. This probability can be used instead of the log odds representation of occupancy for representing grid cells in an occupancy grid map.

3 Related Work

The SLAM problem has received considerable attention by the scientific community, mainly in the past two decades. A plethora of approaches is available, a short overview of which will be given here. The use of laser scanners and probabilistic methods for solving the SLAM problem is state of the art, which is adapted throughout this work.
The SLAM problem is difficult because in the general case where the robot does not have sensors that measure absolute location information (like GPS), so only data relative to the robot can be gathered. This invariably means that small errors due to sensor noise or bias will add up during their integration, leading to errors both in estimated path and map. It is therefore vital to use techniques that minimize these errors.
The majority of SLAM approaches build 2D maps of the environment with the estimated vehicle state being the 2D pose of the robot. True 3D SLAM approaches using monocular vision, 3D laser ranger finders, 3D time-of-flight cameras or stereo cameras are topics of current research and were no option for use in this work due to their experimental nature.

3.1 Taxonomy of SLAM approaches

Approaches to the SLAM problem can be categorized using a number of different criteria.

- Online and offline approaches: Online approaches process incoming information during the robot mission, which means an estimate of the current state is available at runtime. They solve the online SLAM problem, giving a state estimate for time t. In Offline approaches, the data is first gathered as a whole and then processed. They solve the full SLAM problem, estimating the whole trajectory of the vehicle as well as the map.
- Feature and grid based approaches: Feature based approaches extract features from sensor data (for example lines from a laser scan). Those are then used for building a feature based map. In contrast, grid Based approaches use sensor data directly, so arbitrary shaped environments can be mapped.
- Parametric and nonparametric representations: In parametric representations, probability distributions use a functional representation that, while often having computational advantages, generally cannot model arbitrary densities. This is in contrast to nonparametric representations which allow the representation of arbitrary densities, albeit often at higher computational cost.

3.2 Extended Kalman Filter SLAM

Extended Kalman Filter SLAM (EKF SLAM) uses the extended Kalman eilter for state estimation and thus is a parametric approach. Features are extracted from the sensor data and used in the filter. The estimated state consists of the robot pose as well as the location of the map features seen. Because features are part of the state estimate, the state covariance matrix grows

quadratically in size with the number of landmarks tracked, making the use of the standard EKF approach infeasible for environments with a large amount of landmarks. There exist methods to remedy this problem, like approaches using local submaps [10].
EKF SLAM is susceptible to the problem of wrong data assocation, where a detected feature is wrongly matched to an existing feature in the map. This can lead to divergence of the filter with an resulting inconsistent map and loss of localization.

3.3 Particle Filter SLAM

Plain Particle Filter approaches [11] estimate the pose of the robot via a particle filter, while simultaneously estimating the state of the environment in a single grid map.
FastSLAM [12] is a Rao-Blackwellized particle filter approach that is based on factoring the SLAM problem into a pose tracking problem based on a particle filter, while landmark locations are estimated by one EKF for each landmark. For this reason FastSLAM scales much better with the number of features than standard EKF SLAM does. The factorization used is exact and estimates the full posterior over robot and landmark locations. In contrast to EKF SLAM, FastSLAM can track different data associations in the different state estimates of the particles and thus is more robust against wrong data association than EKF based methods.
Grid based FastSLAM does not use features but multiple occupancy grid maps for the map state estimate, one for each particle. Using this approach, there is no need for feature extraction and sensor data can be used directly. This allows for mapping arbitrary environments that might otherwise exhibit poor results with feature based approaches. There is a large body of work available on Rao-Blackwellized particle filters that extends and improves the FastSLAM formulation, for example [13].

3.4 Graph-based SLAM

Introduced in a seminal paper by Lu and Milios [14], Graph-based approaches build a graphical representation of the robot path and map estimate, connecting subsequent poses and measurements. A global optimization of this collected data then gives a globally consistent estimate of the solution to the full SLAM problem.
GraphSLAM [15] uses the information representation for efficient inference, while Nüchter [16] extends the original approach by Lu and Milios to 6 DoF, performing 3D registration of laser scans taken at different locations.

3.5 SLAM in USAR Environments

Much research on SLAM concentrates on 2D mapping in controlled environments, like offices. In contrast to these environments, Urban Search and Rescue scenarios might be less structured and might feature rough and uneven terrain. For this reason, grid based methods are commonly used here.
Special consideration has to be devoted to modeling effects of uneven terrain on the robot sensors, for example changes in laser scanner attitude and thus scans lying outside of the intended scan plane. Robot odometry might also exhibit much larger and irregular errors due to wheel slip.

4 Implementation

In this chapter, the principal, platform agnostic approaches implemented for SLAM in USAR environments are described.

4.1 Design Considerations

The goal of this work is to enable autonomous operation for generic USAR robots. Other than the need for a suitable distance sensor, there are no assumptions about the type of robot. The approach therefore has to scale with the processing power available to the platform and enable autonomous operation, even if in some cases with lower fidelity.
USAR scenarios are usually of small scale, with heavily cluttered and uneven indoor environments. This is in contrast to the setting of many SLAM approaches who map larger, but flat environments like offices and research labs. Given a high enough accuracy and precision of the approach, the loop closing problem can be sidestepped as the environment does not exhibit large loops.
On the other hand, it is very important to have a good estimate of the vehicle and laser scanner attitude, as for example not to accidentally mistake laser scans of the ground for walls, which can lead to serious unrecoverable error if not detected.
Due to the mostly static nature of the environment in the USAR scenario, robustness against moving objects and people is not one of the primary requirements.

4.1.1 Choice of SLAM Approach

Examples of recently successful approaches include the use of a single map combined with a particle filter [17] and a scan matching approach with an underlying EKF for state and odometry estimation [18]. Both approaches were successfully used in RoboCup Rescue competitions featuring USAR scenarios.
As described in the remainder of this section, for maximum flexibility both a particle filter based approach as well as a scan matching approach are implemented, who can be combined. This way a flexible system is available that can be tailored to the environment it has to work in as well as the hardware it has to run on.

4.1.2 Software Design

RoboFrame [19] is a platform independent framework for teams of heterogeneous robots developed at TU Darmstadt and used successfully in applications like the humanoid robots of the Darmstadt Dribblers [20] team. The decision to use RoboFrame was driven by the availability of the following features:

- Platform independence, supports multiple Operating System via an abstraction layer.
- Support for running distributed on multiple, heterogeneous processors.

- Existing software base with world modeling [21] and behavior modeling [22] modules as well as a graphical user interface.

RoboFrame based software is regularly used on computers running Linux, Windows and Mac OS in 32 and 64 bit varieties. While being convenient as the development environment can be chosen freely to a degree, the fact that different platforms are actually used in practice means the code stays truly platform independent.
RoboGui is the framework providing a graphical user interface for RoboFrame applications. It provides logging and connectivity functionality and reduces the time needed to create new or extend existing dialogs for debugging and visualization of the robot state.
The existing world and behavior modeling software provides an implementation of team communication between agents, so realizing cooperative behavior between robots can be realized easily if required.
The SLAM implementation uses C++ templates to achieve modularity. This way, compile time instead of runtime inheritance is used to enable inlining of code inside of tight loops.
For matrix and vector operations the Eigen2 library [23] is utilized. It is an open source C++ template library for linear algebra. Eigen2 is a header-only library offering good performance while being lightweight and providing a convenient API.

4.2 Modeling Maps

Occupancy grid maps lend themselves naturally to mapping of unstructured environments. Due to the comparably small scale of the USAR scenario, a fixed size grid map is used for this work. The cell type is a template parameter of the *GridMapBase* class, therefore the actual approach used for updating and storing information in the grid cells can be changed at compile time. A class using the log odds represention of occupancy as well as an implementation of the counting model as described in Section 2.4 is implemented and available.
The basic grid map implementation is provided the *GridMapBase* class providing common functionality that also is useful for applications other than occupancy grid mapping. It is for example used as a data structure for the exploration planning module that was developed in parallel with this work. *GridMapBase* provides functions for coordinate conversions between map and world coordinates as well as accessing the grid cells.
The implementation of the grid can be selected to use either the Boost.MultiArray Library [24] or a plain two dimensional pointer array. Testing revealed the runtime of Boost.MultiArray to be up to three times slower even with preprocessor define `BOOST_DISABLE_ASSERTS` activated. For this reason it is not used per default.
OccGridMapBase extends the *GridMapBase* class and supplies functions specific to occupancy grid maps. This includes functions for changing single grid occupancy values according to the update scheme used and updating the map via laser scans.

4.2.1 Accessing Maps

The discrete and discontinuous nature of occupancy grid maps limits the precision that can be achieved and also does not allow the computation of interpolated values or derivatives in the plain implementation. The computation of approximate map occupancy values and derivatives

for non-integer map coordinates is important for some applications, as described in Section 4.5. For this reason, a interpolation scheme allowing sub-grid cell accuracy is implemented by employing bilinear filtering.

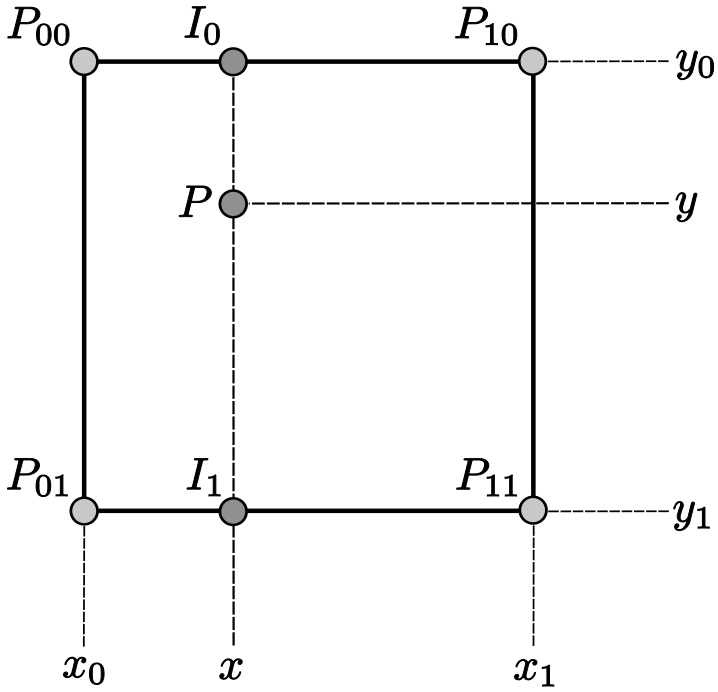


Figure 4.1: Bilinear filtering of the occupancy grid map. Point P_m is the point whose value shall be interpolated.

Given a continuous map coordinate P_m, occupancy value $M(P_m)$ as well as gradient $\bigtriangledown M(P_m)$ are approximated as follows: First the four closest integer coordinates $P_{00..11}$ are established as depicted in figure 4.1. For bilinear interpolation, linear interpolation is first performed on the x axis:

$$M(I_0) \approx \frac{x - x_0}{x_1 - x_0} M(P_{10}) + \frac{x_1 - x}{x_1 - x_0} M(P_{00}) \tag{4.1}$$

$$M(I_1) \approx \frac{x - x_0}{x_1 - x_0} M(P_{11}) + \frac{x_1 - x}{x_1 - x_0} M(P_{01}) \tag{4.2}$$

Linear interpolation about the y axis then yields:

$$M(P_m) \approx \frac{y - y_0}{y1 - y_0} M(I_1) + \frac{y_1 - y}{y_1 - y_0} M(I_0) \tag{4.3}$$

The derivative along the x direction can be approximated by:

$$\frac{\partial M}{\partial x}(P_m) \approx \frac{y - y_0}{y_1 - y_0}(M(P_{11}) - M(P_{01})) + \frac{y1 - y}{y_1 - y_0}(M(P_{10}) - M(P_{00})) \tag{4.4}$$

and the approximate derivative in y direction is

$$\frac{\partial M}{\partial y}(P_m) \approx \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{10})) + \frac{x_1 - x}{x_1 - x_0}(M(P_{01}) - M(P_{00})) \tag{4.5}$$

(a) (b) (c)

Figure 4.2: Occupancy grid map access: (a): Occupancy grid map (b) Spatial derivative $\frac{\partial M}{\partial x}$ of gaussian blurred map (c) Spatial derivative $\frac{\partial M}{\partial y}$ of gaussian blurred map.

It should be noted that the sample points/grid cells in the map are situated on a regular grid with distance 1 from each other, which simplifies the gradient equations above.
With the approximate derivatives in x and y direction available, we now have an estimate of the map gradient:

$$\bigtriangledown M(P_m) = \left(\frac{\partial M}{\partial x}(P_m), \frac{\partial M}{\partial y}(P_m) \right) \tag{4.6}$$

For some applications, obtaining map data that has been smoothed by gaussian convolution is needed. This is achieved by applying a gaussian convolution operation to grid points $P_{00..11}$ and using the result of the filter operation as $M(P_{00..11})$ in the linear filtering operations described above.
The results of the filter operation are cached, so the expensive convolution operation is only executed once per grid cell in case of multiple access. When the underlying map changes, the cached filter data is discarded.
The caching scheme is selectable between two options. The first variant caches the filter data in a copy of the map, looking up the map coordinates and returning the cached value if available. If the filtered value is not yet stored in the cache map, the convolution operation is executed, storing the result in the map and returning it.
The second option uses the unordered_map class from the Boost Unordered Library [25] for storing the filtered values. It is slower than the map based cache method, but far more memory efficient. The decision which caching scheme to use therefore depends on the memory and runtime efficiency needs of the specific platform.
Figure 4.2 shows an example for the application of the grid map access mechanisms described.

4.3 Modeling Laser Scanners

Most Laser scanners currently used for robotic mapping applications scan a planar portion of the environment. A laser range finder emits beams towards a rotating mirror, which then reflects them towards different parts of the environment depending on the mirror rotation angle. The

two major methods for estimating the distances are phase shift and time of flight ranging. For a short overview about these methods the reader might refer to [26].
Figure 4.3 shows a schematic of lasers canner operation and the necessary coordinate transformation. The term "laser scan" refers to the measurements taken during one complete sweep of the mirror and consists of hundreds of distance samples for modern scanners.

4.3.1 Coordinate Transformation

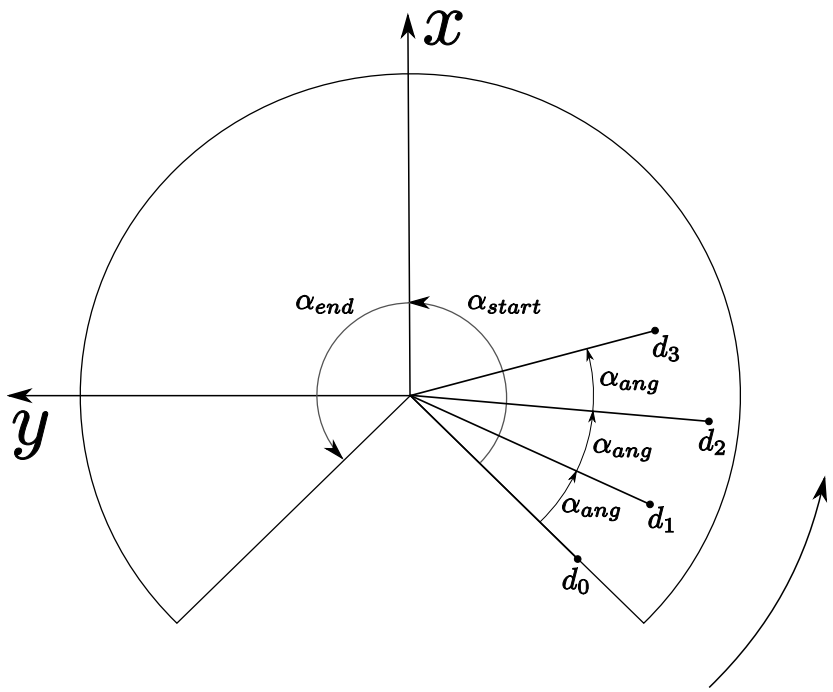


Figure 4.3: Example schematic of a laser scanner. The rotating mirror is situated in the middle. The scanner in this example starts the scan at α_{start} and rotates the mirror in counterclockwise direction. Distance samples d_i are taken with an angular resolution of α_{ang}. A whole scan consists of the distance samples taken from α_{start} to α_{end}. As in this example, most laser scanners have a 'blind' area of mirror rotation where no measurements are taken.

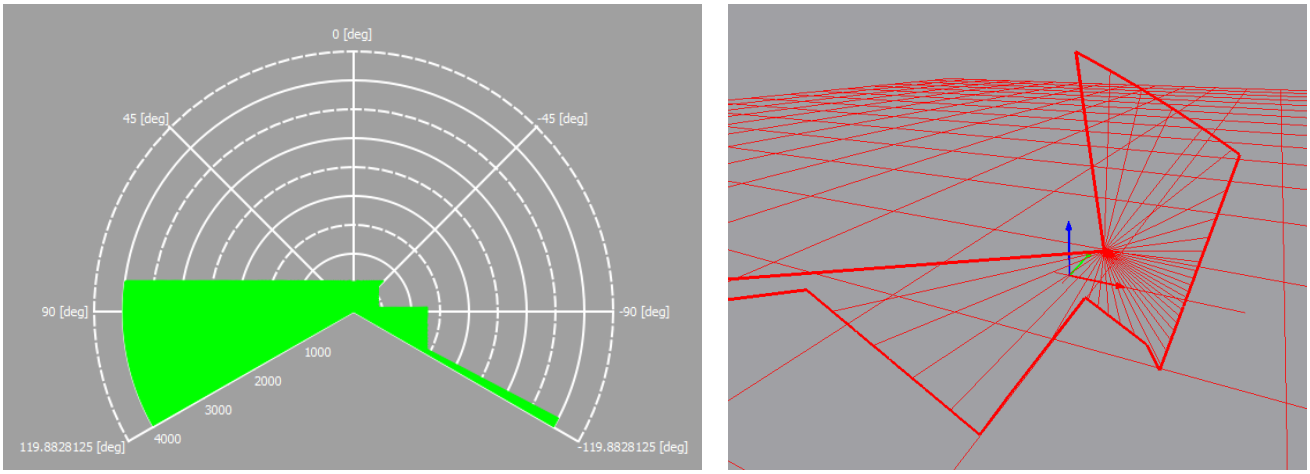
The raw data is typically given as an array of distance values. Together with information about the laser scanner angular resolution and scan start angle, these can be transformed into a representation of the scan in the laser scanner polar coordinate system, with the origin being in the axis and at the height of the rotating mirror. A transformation of the laser data into the robot coordinate system is necessary for use in the SLAM algorithm. The transformation from distance measurements d_i to cartesian laser scanner coordinates ${}^{s}s_i$ by using the known starting angle α_{start} and angular resolution α_{ang} is straightforward as follows:

$$ {}^{s}s_i = \begin{pmatrix} {x_s}^i \\ {y_s}^i \\ {z_s}^i \end{pmatrix} = \begin{pmatrix} cos(\alpha_{start} + i\alpha_{ang}) \\ sin(\alpha_{start} + i\alpha_{ang}) \\ 0 \end{pmatrix} d_i \tag{4.7} $$

The data now available in cartesian sensor coordinates now can be transformed into the robot coordinate system:

$$^{r}s_i = {}^{r}T_b \, {}^{b}T_s \, {}^{s}s_i \tag{4.8}$$

with ${}^{b}T_s$ being the transformation from the scanner to the body fixed coordinate system of the robot and ${}^{r}T_b$ being the attitude of the robot coordinate system in relation to the world coordinate system. Figure 4.4 shows an example for a scan shown in scan coordinates and the same scan transformed into robot coordinates.



(a) (b)

Figure 4.4: (a) Laser scan as visualized in sensor coordinates (b) The same scan transformed into robot coordinates. It can easily be seen that scanner is inclined downwards, scanning the ground in front of the robot. The robot coordinate system depicted by the three orthogonal arrows in RGB colors.

The transformed scan might then be used for 2D mapping, by using only the x and y coordinates of the transformed endpoints. Depending of the application and pose of the scanner, endpoints with too high or low z coordinates have to be filtered out, as to not use scans of the ground or locations outside of the map plane. The range of valid scan z-values depends on the environment and can be changed during runtime. For certain environments (e.g. office buildings) with known vertical walls, the limits might not be as restrictive as in environments where this cannot be guaranteed. Depending on the accuracy of the vehicle and laser scanner attitude estimation as well as the structure of the environment, laser scans with too close or too far distance values might also be filtered. For large measured distances, even small errors in attitude estimation can result in erroneous scans, due to accidentally scanning the ground instead of walls.

4.3.2 Forward Sensor Model

A tractable model is needed for estimating the laser scan likelihood, so the scan data can be used in a probabilistic filter. By assuming all laser beams in a scan to be independent, a range scan likelihood might be considered to be the product of the individual beam likelihoods:

$$p(z_t|x_t, m_{t-1}) = \prod_{i=1}^{n} p(z_t^i|x_t, m_{t-1}) \tag{4.9}$$

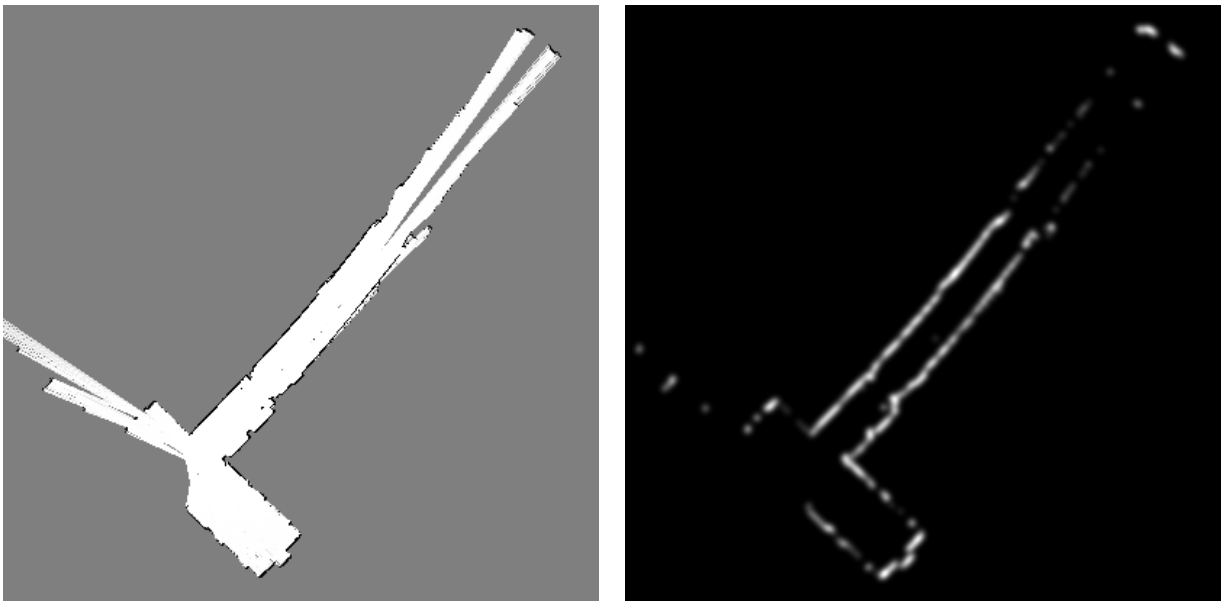
The independence assumption for the individual beams is a very strong one. In reality, neighboring scans are often correlated with each other due to measuring the same object. For this reason, taking the product of all beams can lead to extremely peaked likelihood functions which is undesirable. For this reason, often only a fraction of the available beams are used for determining the scan likelihood.

Beam based Model

A physically plausible method of computing the likelihood of a laser scan are beam based models. Just as on the real physical robot, a ray starting at the laser scanner position in world coordinates is cast into the map until it intersects a cell that is marked as occupied. The distance measured by the sensor z_i and the distance determined by the ray casting operation serve as input for a one dimensional parametric function that evaluates the likelihood for the given combination of inputs.
The ray casting approach has desirable properties: As the propagation of the real physical laser is emulated, the sensor cannot see through walls. Also, the parameters of the function determining the likelihood can be fine tuned by hand or optimized by learning from sensor data using the Expectation Maximization algorithm [27]. On the other hand, due to the complexity of the model, the computation time for evaluating the scan likelihood via the ray casting model is large compared to methods that use a simpler model which evaluates likelihoods based on scan end points. A further refinement of the ray casting model estimating correlation between individual beams in the context of localization can be found in [28].

Beam Endpoint Model

Methods using beam endpoints have gained in popularity in recent years and are also used in this work as the main sensor model. Here, only the beam endpoint in world coordinates is used for determining the likelihood for the respective beam. The ray casting operation can then be replaced by a simple lookup into the map. The endpoint based model is more crude than the ray casting model in that it lacks the plausible physical interpretation of the latter. Despite this fact, endpoint approaches are widely used as they use less computational power per evaluated likelihood, which enables the use of more particles in a particle filter approach. This in turn can improve overall performance, even if the sensor model itself is not immediately physically plausible.
Using the endpoint of a beam, different approaches can be used to arrive at a likelihood estimate. A simple ad-hoc approach used in practice [17] to estimate a weight for a combination of a scan



(a) (b)

Figure 4.5: (a) Occupancy grid map (b) Likelihood field generated from this map used for the scan endpoint based sensor model. Note the map likelihood field is smoothed with a gaussian as described in Section 4.2.1

and pose is to take all cells c_i of the map that correspond to endpoints of the scan and sum the probability of occupancy:

$$w = \sum_{i=1}^{n} p_{occ}(c_i) \tag{4.10}$$

In other endpoint based approaches [29], the closest obstacle from the given beam endpoint is determined and the distance between endpoint and obstacle serves as an estimate for the measurement likelihood for the given beam endpoint. The likelihood for the complete scan is determined by the product of the individual beam likelihoods as in equation 4.9.
Instead of determining the closest obstacle by searching, a convolution of the grid cell containing an endpoint with a gaussian that smoothes over the surrounding occupancy probabilites can be used to estimate likelihood for the scan endpoint. The farther away obstacles are, the smaller the result of the convolution operation. Figure 4.5 shows a likelihood field generated using this method.
This approach is implemented using the Grid Map access scheme described in chapter 4.2.1 to provide the gaussian smoothing.

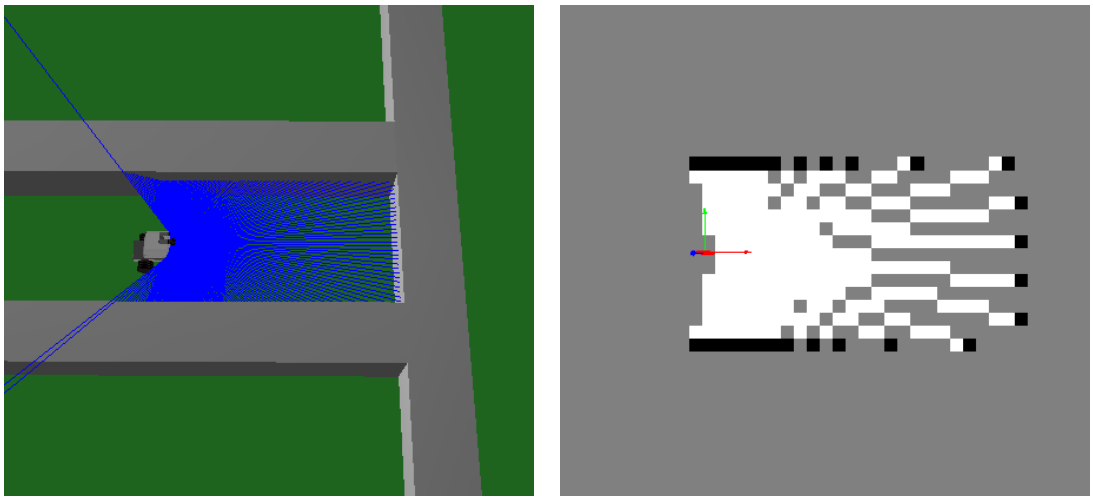
4.3.3 Inverse Sensor Model

For updating the map, the inverse sensor model has to be specified:

$$p(m_i|x_t, z_t) \tag{4.11}$$

Given the current measurement z_t and the robot state estimate x_t, the inverse measurement model returns the occupancy probability for all grid cells m_i of the map, which can then be used

to update the map estimate. In practice, only a small portion of all grid cells is covered by a laser scan, so a more efficient updating scheme than iterating over the complete map is needed.
As laser scans can conveniently be modeled to be consisting of individual beams, these can be integrated into the map update by using a slightly modified variant of the widely used Bresenham algorithm for rasterization [30].
Because modern laser scanners are very accurate and precise, the inverse sensor model can be implemented by updating all cells visited by the Bresenham rasterization for a given beam as free, with the exception being the cell containing the endpoint which is updated as occupied. The update itself may either use the log odds ratio of occupancy or the counting model scheme as described in chapter 2.4.
A common modification is to only allow one update or a limited number of updates per grid cell per invocation of the inverse sensor model. This averts updating cells close to the robot repeatedly as well as other rasterization artifacts.
Figure 4.6 shows a map update by the inverse sensor model.



(a) (b)

Figure 4.6: Updating the map with the inverse sensor model: (a) shows ground truth in the simulation environment. (b) shows the map as updated by the inverse sensor model. Only every 30th scan was used for the map update here, to make the Bresenham rasterization visible.

4.4 Particle Filter

A generic particle filter based on Algorithm 3 is implemented for this work. The filter is configurable via C++ templates indicating which state, motion update, measurement update and resampling approach to use. By offering modularity via templates, components can easily be exchanged.
For the purpose of this work, the robot state to be estimated is represented by the 2D pose of the robot, while the map estimate is provided by an occupancy grid map as described in chapter 4.2.

4.4.1 Motion Update

The motion update is used for the prediction step in the Particle Filter. It samples from the density

$$p(x_t|x_{t-1}, u_t) \tag{4.12}$$

and generates a new particle set that represents the predicted density. In practice, this is achieved by sampling a transformation from the probabilistic motion model for each particle and transforming each particle of set χ_{t-1} by one such sampled transformation. The data structure representing χ_{t-1} is overwritten in the process and afterwards holds the predictive particle set $\overline{\chi_t}$.
A flexible motion model taking estimated velocity v_t and angular rate ω_t of the robot as inputs is implemented in algorithm 4 taken from [1]. Depending on the robot platform, the vehicle might supply timestamped odometry data with estimated translation and rotation or it might only have control inputs available that are generally less accurate. The velocity based motion model can cope with both cases via different noise parameters $\alpha_{1..6}$.
While odometry data are in reality measurements, treating them as such would mean expanding the state vector to include them. For this reason, odometry, or more general any estimate about the motion of the vehicle often is treated as a control.
In lines 1 to 3 of algorithm 4, samples of gaussian noise with variance depending on the noise

Algorithm 4 SampleMotionModel(u_t, x_{t-1})

1: $\hat{v} = v + sampleGaussian(\alpha_1 v^2 + \alpha_2 \omega^2)$
2: $\hat{\omega} = \omega + sampleGaussian(\alpha_3 v^2 + \alpha_4 \omega^2)$
3: $\hat{\gamma} = sampleGaussian(\alpha_5 v^2 + \alpha_6 \omega^2)$
4: $x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\triangle t)$
5: $y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\triangle t)$
6: $\theta' = \theta + \hat{\omega}\triangle t + \hat{\gamma}\triangle t$
7: **return** $x_t = (x', y', \theta')^T$

parameters is added to the velocity and angular rate supplied by odometry. In lines 4 and 5, the new estimated position after travelling with velocity v and angular rate ω is determined. Line 6 updates the orientation. Note that an additional term depending on γ is added, as to not constrain the vehicle to perfect circular motion, which would be the case if γ was omitted.
The *sampleGaussian* method is implemented using the Boost Random Number library [31], providing convenient functions for sampling from a multitude of distributions.

4.4.2 Measurement Update

In the particle filter Measurement Update step, particles are weighted according to the likelihood as determined by the the Forward Sensor Model described in 4.3.2. For numerical stability, the log likelihood of the measurement model equation 4.9 is used:

$$\log p(z_t|x_t, m_{t-1}) = \sum_{i=1}^{N} \log p(z_t^i|x_t, m_{t-1}) \tag{4.13}$$

The importance weight for particle ${x_t}^{[i]}$ is then given by:

$$ {w_t}^{[i]} = \log p(z_t | {x_t}^{[i]}, m_{t-1}) \tag{4.14} $$

After the measurement update is complete, particle weights are normalized to sum to 1.

4.4.3 Resampling

For the Resampling step, the effective sample size [32] is first determined:

$$ N_{eff} = \frac{1}{\sum_{i=1}^{N} ({w_t}^{[i]})^2} \tag{4.15} $$

The effective sample size is a measure of similarity between the sample set and the target distribution. The higher the variance in importance weights, the lower N_{eff} becomes, meaning that many particles are located in an area of the target distribution having low probability density. Resampling only takes place if N_{eff} is below a threshold value, often chosen to be $N/2$.
Resampling is implemented as a low variance resampler. Here, instead of repeatedly drawing N random numbers and copying the particle with the corresponding weight into the new set, one random number r is generated in $(0; N^{-1})$ and used as the base for iterating over the particle set. Figure 4.7 shows a comparison of the naive and the low variance resampling approach.
Using selective Resampling via thresholding N_{eff} as well as using low variance resampling helps to keep particle set diversity and to reduce the risk of sample impoverishment, where a particle set fails at representing the target distribution because particles do not sample it sufficiently.

Algorithm 5 SelectiveResampler(χ_t, W_t)

$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^{[}i])^2}$
if $N_{eff} < N_{resampleThreshold}$ **then**
  $\overline{\chi_t} = \emptyset$
  $r = rand(0; N^{-1})$
  $c = {w_t}^{[1]}$
  $i = 1$
  for $n = 1$ to N **do**
    $U = r + (n-1)N^{-1}$
    while $U > c$ **do**
      $i = i + 1$
      $c = c + {w_t}^{[i]}$
    end while
    add ${x_t}^{[i]}$ to $\overline{\chi_t}$
  end for
  return $\overline{\chi_t}$
else
  return χ_t
end if

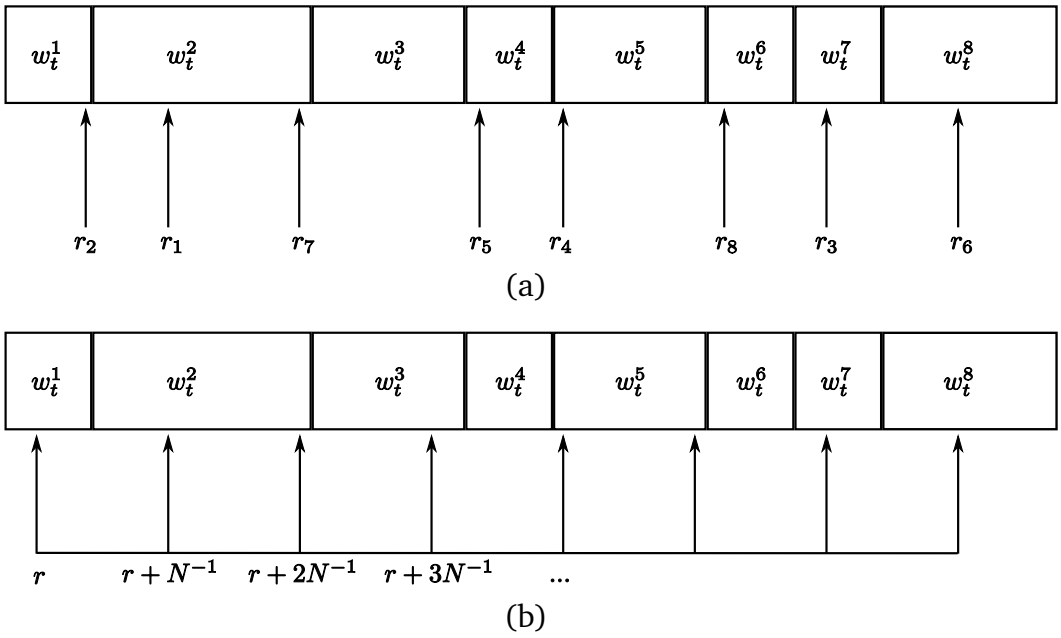


Figure 4.7: Comparison of resampling methods: (a) shows the naive approach where N random numbers r_i are generated to select particles. (b) shows the Low Variance resampling approach that selects particles in a more systematic way.

4.4.4 Density Estimation

For many applications, a continuous and compact belief representation is needed, which is not provided by the particle set as it uses a discrete approximation to the continuous belief. For the purpose of this work, a gaussian approximation is used as the particle set generally stays approximately unimodal for the SLAM application and a gaussian representation can easily be used for visualization or updating parametric gaussian filters as described in Section 5.5.

4.5 Scan Matching

Scan matching is the process of aligning laser scans with each other or with an existing map. Modern laser scanners have very low measurement noise, in the order of few millimeters standard deviation. A method for registering scans might yield very accurate results for this reason. For many robotic systems the accuracy and precision of the laser scanner is much higher than that of odometry data.

The Iterative Closest Point (ICP) [33] method for scan matching originated as a general approach for registering two 3D point clouds. Here, Point correspondences between the two point sets are established with an exact or approximate nearest neighbor criterion. A transformation between both point sets is estimated that minimizes the distance between the point correspondences in a least squares sense. This transformation is applied to one of the sets and the next iteration of the correspondence search is started. This process is iterated till convergence. The main drawback of the ICP method is the expensive search for point correspondences, which has to be done in every iteration.

Polar Scan Matching (PSM) [34] avoids the correspondence search by taking advantage of the

natural polar coordinate system of the laser scans to estimate a match between them. The scans have to be preprocessed here to be used in the polar scanmatcher.
In [35] an approach that matches a scan with an occupancy grid is presented. This occupancy grid might be built from one or more preceding scans. Matching scans against an existing map has the following desirable properties:

- No need for establishing correspondences between scans.
- No need for preprocessing scans.
- Possibility to match one scan against a history of preceding scans.
- Possibility to match a scan with an preexisting occupancy grid map.

Because of these properties, a scan matching approach for registering scans with an occupancy grid is implemented in this work. The laser scan is represented by the scan endpoints in robot coordinates. To align the scan with the occupancy grid, a configuration that maximizes the likelihood of the scan with regards to the map has to be found. The registration approach described here is similar to the stereo vision disparity estimation method presented in [36]. We seek to minimize

$$\underset{p}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(S_i(p))\right]^2 \tag{4.16}$$

that is, we want to find the transformation p that gives the best alignment of the laser scan with the map. Here, $S_i(p)$ are the world coordinates of scan endpoint s_i. They are a function of a rigid transformation with parameters $p = (t_x, t_y, \theta)$, the pose of the robot in world coordinates:

$$S_i(p) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & cos(\theta) & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_i x \\ s_i y \\ 1 \end{pmatrix} \tag{4.17}$$

The function $M(S_i(p))$ returns the map value at the coordinates given by $S_i(p)$. We have some starting estimate of p, and want to estimate Δp which is closer to the minimum:

$$\sum_{i=1}^{n} \left[1 - M(S_i(p + \Delta p))\right]^2 \tag{4.18}$$

By first order Taylor expansion of $M(S_i(p + \Delta p))$ we get:

$$\sum_{i=1}^{n} \left[1 - M(S_i(p)) - \bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p} \Delta p))\right]^2 \tag{4.19}$$

This equation is minimized by setting the partial derivative with respect to Δp to zero:

$$2 \sum_{i=1}^{n} \left[\bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p}\right]^T \left[1 - M(S_i(p)) - \bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p} \Delta p))\right] = 0 \tag{4.20}$$

solving for Δp gives the Gauss-Newton equation for the minimization problem:

$$\Delta p = H^{-1} \sum_{i=1}^{n} \left[\bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p} \right]^T \left[1 - M(S_i(p)) \right] \tag{4.21}$$

with

$$H = \left[\bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p} \right]^T \left[\bigtriangledown M(S_i(p)) \frac{\partial S_i(p)}{\partial p} \right] \tag{4.22}$$

An approximation for the map gradient $\bigtriangledown M(S_i(p))$ is provided in chapter 4.2.1. With equation 4.17 we get

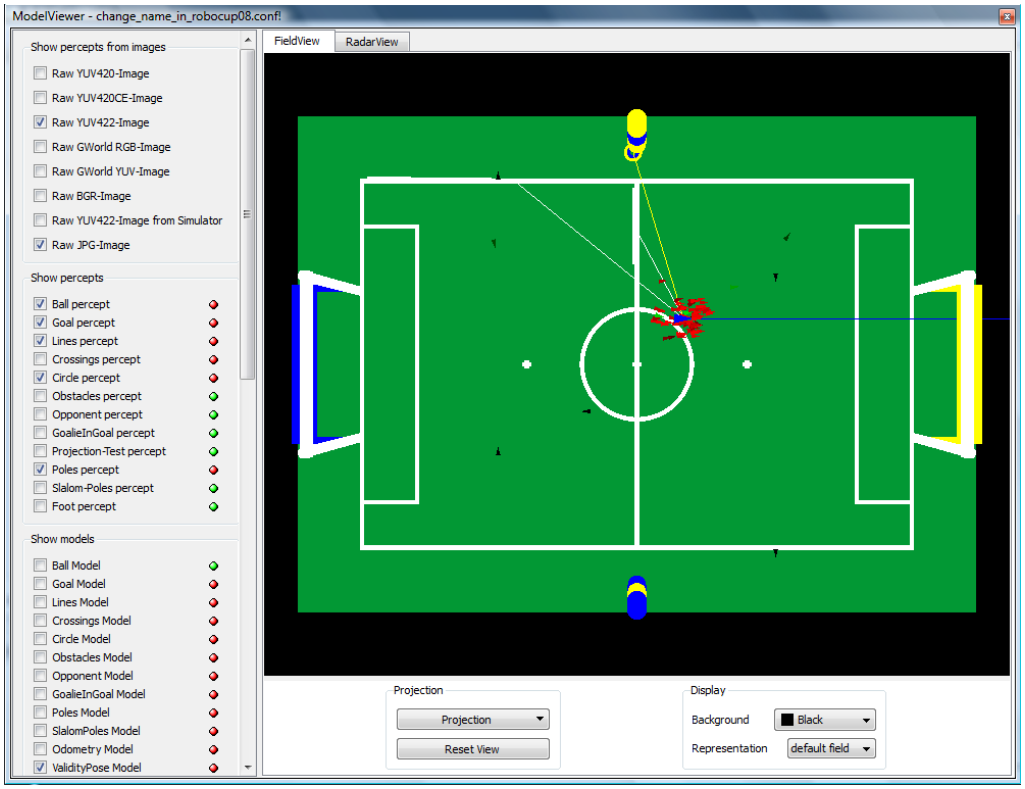
$$\frac{\partial S_i(p)}{\partial p} = \begin{pmatrix} 1 & 0 & -\sin(\theta) s_i x - \cos(\theta) s_i y \\ 0 & 1 & \cos(\theta) s_i x - \sin(\theta) s_i y \end{pmatrix} \tag{4.23}$$

Using $\bigtriangledown M(S_i(p))$ and $\frac{\partial S_i(p)}{\partial p}$, the Gauss-Newton equation 4.21 can now be evaluated, yielding a step Δp towards the minimum. With updating p by Δp this process might be repeated till a convergence criterion is met.
It is important to note that the algorithm works on non-smooth linear approximations of the map gradient $\bigtriangledown M(S_i(p))$ meaning local quadratic convergence towards a minimum cannot be guaranteed. Nevertheless, the algorithm works with sufficient accuracy in practice.

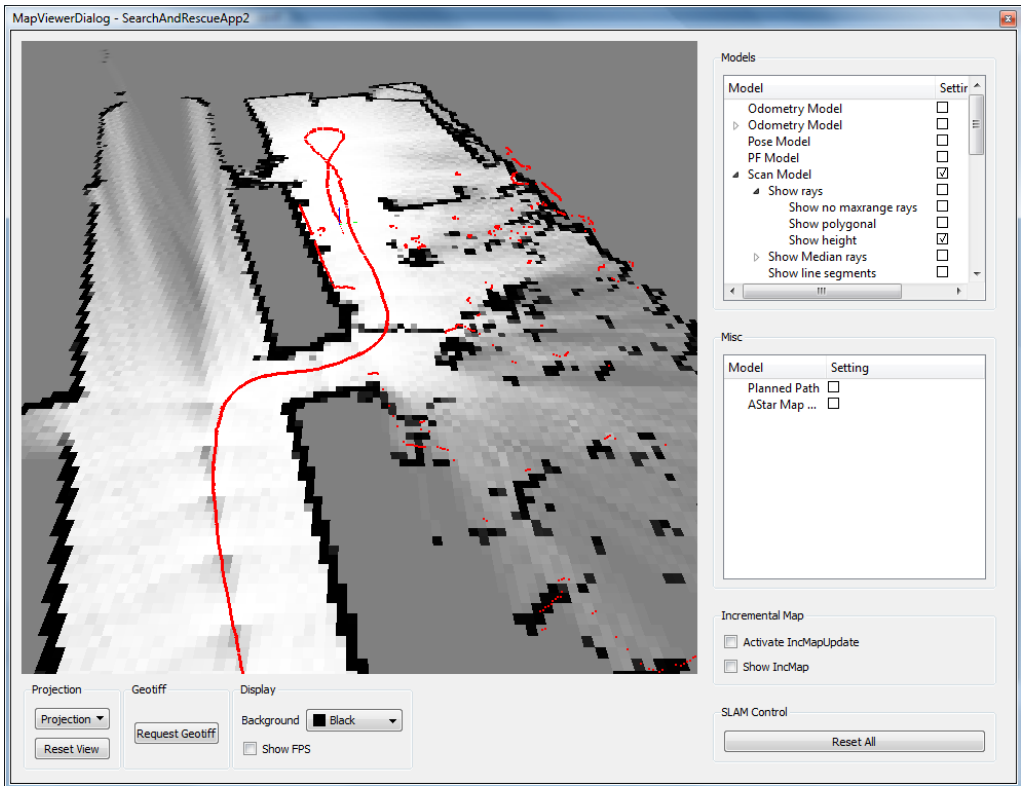
4.6 Graphical User Interface

The existing *ModelViewer* dialog for visualization of world model and sensor data originally developed for use with Humanoid Soccer Robots proved too limited for use with the SLAM application. Due to the need for estimation of the 6DoF vehicle pose, a capability of full 3D visualization as well as capability to visualize large amounts of data like point clouds and occupancy grids is needed. The *Modelviewer* dialog, while providing a 3D view, does not satisfy these requirements. For these reasons a new visualization system is implemented. The *DrawingInterface* class provides an interface for drawing operations that might be implemented for different rendering platforms. An implementation for OpenGL together with Qt is provided as the *GLDrawingInterface* class and used in the *GLVisualizationWidget* class. This is a generic widget that can be used as a GUI element in the RoboGui, like any other Qt widget. This way, a convenient and generic 3D visualization element is available also for use in other dialogs of the GUI system. Fast and intuitive navigation in the 3D view is possible by clicking and holding the mouse for rotating the view as well as grabbing the ground plane via the middle mouse button for translation.
The GLEW [37] library is used to make OpenGL extensions easily available. The inclusion was driven by GL extensions that enable more efficient updating of textures that are used for drawing the map.
The *GLVisualizationWidget* is also used in the main visualization for the SLAM system, the *MapViewer*. Here, the visualization is accompanied by a tree view of available worldmodel data, allowing selection of the information to be visualized.
Using the *DrawProperty* mechanism, a tree structured set of visualization options can be specified directly in source code for each class that is part of world modeling, with the drawing options then conveniently available and selectable in the *MapViewer* without further implementation

overhead.
Realtime viewing of the mapping process is realized by optionally sending a *IncMapUpdate* message to the GUI when requested. The incremental map update contains the scan and the pose used for updating the map on the robot. The map update is then carried out on a local map in the *MapViewer* in the exact same way like on the robot. This way, a convenient realtime view of the robot progress through the environment is possible while causing minimum network traffic.
The *GLVisualizationWidget* class is also used in the *AttitudeViewer* dialog providing three 3D views for easy debugging of attitude information as well as the *GridMapTestDlg* which was used for debugging and visualization purposes during development of the SLAM system.
The parameters of the SLAM system can be changed via the *SlamParametersDialog*, using the *VariantSet* class provided by RoboFrame. Here, parameters are read from a configuration file at startup and can be changed via the GUI at runtime.
Figure 4.8 shows a comparison of the pre-existing *ModelViewer* and the newly developed *MapViewer* dialogs.



(a)



(b)

Figure 4.8: Graphical user interface: (a) ModelViewer Dialog not used for SLAM visualization as it proved too limited. (b) MapViewer dialog developed for SLAM and generic 3D visualization. Visualization of trajectories and map textures is supported. Also note the tree structure of visualization options on the right.

5 Experimental Platform

(a) (b)

Figure 5.1: Vehicle: (a): Unmodified Kyosho TwinForce R/C car (Image from [38]) (b): The experimental platform created by modification of a TwinForce R/C car.

The vehicle used for testing and evaluation of this work is used by the Darmstadt Rescue Robot Team [39] for participation in the RoboCup Rescue League. In the RoboCup Rescue scenario, UGVs have to search a simulated USAR scenario for victims, which are represented by baby dolls and heat blankets. To fulfill the autonomous mission task, the vehicle has not only be able to solve the online SLAM problem, but also has to detect victims and make useful decisions towards this goal autonomously. For this reason, the vehicle is also equipped with a daylight and thermal camera.
The UGV system is split into two major parts, one being the chassis with the built in PC104 board and the other being the detachable Vision Box that supplies the sensors and computation capabilites needed for tasks like the RoboCup Rescue mission. While the vehicle is mostly operated using both components, this is not a requirement. The UGV may also operate without the Vision Box, albeit with less external sensors and computation capability.

5.1 Chassis

The vehicle is based on a heavily modified Kysosho TwinForce R/C car chassis. A unmodified Kyosho TwinForce R/C car is depicted in figure 5.1(a), while figure 5.1(b) shows the modified vehicle. Among the major hardware changes to the original vehicle are:

- All wheel steering for better maneuverability.
- Removal of one of the original two motors and exchange of the gear system, to allow for slower movement.
- Addition of a brake system to permit better control when the vehicle is located on slopes.

	Hokuyo URG-04LX	Hokuyo UTM-30LX
protocol	SCIP V2.0	SCIP V2.0
weight[g]	160	370
range[mm]	4000	30000
max scan rate[Hz]	10	40
angular resolution[°]	0.36	0.25
distance measurements per scan	682	921

Table 5.1: Types of Laserscanners used

- Removal of bodywork and addition of custom casings for onboard systems.

5.2 Internal Sensors

The vehicle carries a suite of internal sensors, consisting of:

- An Analog Devices ADIS16350 Inertial Measurement Unit.
- A magnetic compass.
- 4 Wheel Encoders, one in each of the four wheels.
- Joint Angle Sensors in the servos used for pointing the sensors.

5.3 External Sensors

Among the external sensors are a UEye Daylight Camera and a Thermal Vision thermal Camera. Both are mounted on a stabilized pan/tilt servo mount and used for victim detection or other vision tasks.
Three ultrasonic sensors are mounted at the rear of the vehicle. They are currently not used in the SLAM system as their data is too noisy and unreliable to augment the laser scanner data in a meaningful way. They are currently only used by the autonomous behavior in cases when a map of the area behind the vehicle is not available.
Two laser scanners as described in Table 5.1 are mounted at the front of the vehicle.
A Hokuyo URG-04LX is mounted on a tiltable servo and mainly intended to be used for mapping the ground in front of the vehicle. If the vehicle is used without the modular Vision Box, the SLAM system can also run on the PC104 board and use the URG-04LX scanner.
A Hokuyo UTM-30LX laser scanner is mounted on a roll/tilt unit on the Vision Box and is the preferred sensor for SLAM, as it has long range and a high scan rate of 40Hz. The stabilization in roll and tilt allows to keep the sensor attitude close to the intended scan plane for SLAM.

5.4 Data Processing and Infrastructure

Figure 5.2 shows an overview about the UGV component systems and their hardware connections. On the lowest level, an interface board reads out most of the internal sensors. Low level control

and high update rate state estimation of the vehicle are provided by a PC-104 board with a 500MHz Geode LX CPU. This board is running the realtime Linux Kernel Xenomai [40] and estimates the vehicle navigation solution (NavSolution) using an Inertial Navigation System (INS) with an extended Kalman filter with an update rate of 50Hz. The NavSolution state includes the vehicle position and orientation as well as linear and angular velocities. The timestamped NavSolution data is made available at the same 50Hz rate.
A picoITX board with a Intel Core 2 Duo 2.4 GHz CPU and a nVidia GeForce 9500 GT graphics card is used for high level behavior control, vision and SLAM processing. The graphics card so far is used exclusively for vision processing using CUDA [41].

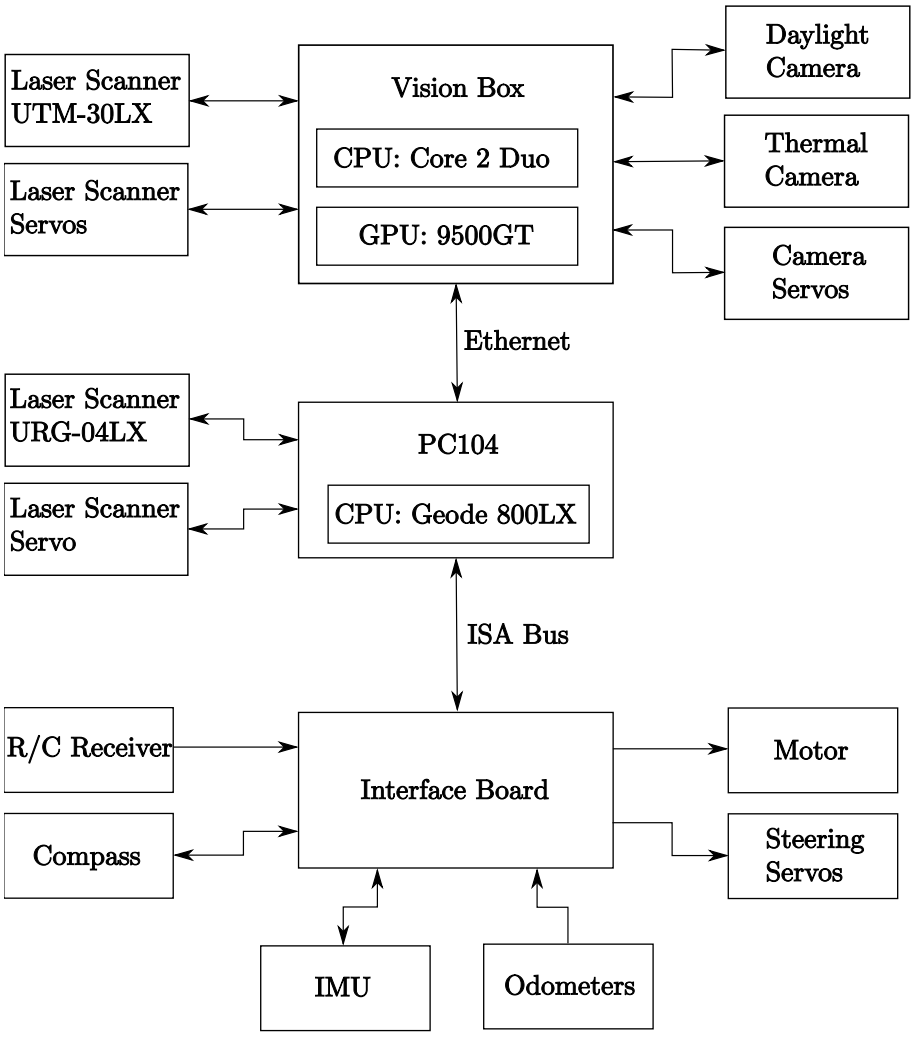


Figure 5.2: UGV component systems and their connections. Arrows depict the (possible) direction of data flow.

5.5 SLAM on the UGV

The generic SLAM system described in chapter 4 has to get adapted for use on the UGV platform. Figure 5.3 shows an overview of the interaction between the SLAM system and the INS.
The EKF based INS provides the principal state estimate of the system with the NavSolution

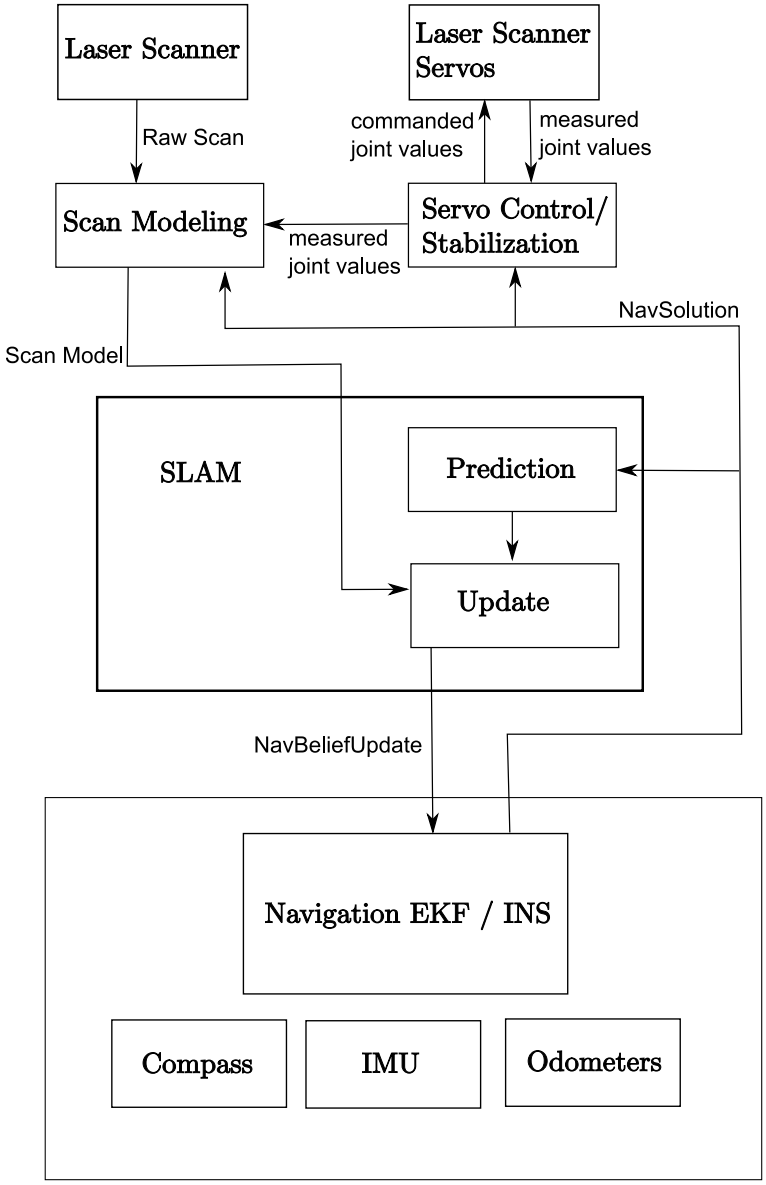


Figure 5.3: Integration of SLAM system and INS on the UGV

and integrates data from all available sensors. The estimated velocity and angular rate of the NavSolution are used by the motion model for the prediction step of the SLAM system.
The Hokuyo UTM-30LX laser scanner stabilized in roll and pitch is used as the sensor for the SLAM observation update. The stabilization is based on the NavSolution data and stabilizes the scanner at a control rate of 10Hz. The servo joint values are also read at 10Hz and used in ScanModeling, where the translation, roll and pitch of the servo as well as the roll and pitch of the UGV provided by the NavSolution are used to transform the scan into robot coordinates.
The SLAM system then updates the robot state estimate, consisting of the map and the pose estimate. The SLAM pose estimate is fed back into the INS as a pose belief update (NavBeliefUpdate) containing the estimated pose as well as the covariance of the 2D pose estimate. Using this feedback mechanism, uncertainty of the INS solution in translation and orientation can be reduced and bias estimation of inertial sensors can be updated frequently and precisely. This in turn leads to an improvement of the NavSolution estimated by the INS.

By using this approach, a state estimate of the vehicle is available with an update rate of 50Hz, while the SLAM system can run asynchronous in parallel and send NavBeliefUpdates when they are available.
When using only the PC104 board for SLAM without the Vision Box, the mechanism described transparently works the same, albeit with lower CPU power available and using the lower range and scan rate URG-04LX scanner.

5.6 Odometry Estimation

As the INS described above might not always be available, an odometry estimation module is implemented that might model odometry independently given the wheel encoder data as input. As described in Section 5.2, the vehicle is equipped with a wheel encoder in each of the four wheels. For modeling motion in the 2D plane, velocity and angular rate of the vehicle have to be estimated. Estimating the two unknowns from four equations amounts to an overdetermined, least squares estimation problem. Being a sequential estimation problem, an extended Kalman filter lends itself nicely for this task. Even if noise is not normally distributed, the EKF works well in practice and can be easily tuned. Figure 5.4 shows a schematic overview of the relevant vehicle geometry. Here, without loss of generality, the robot coordinate system is positioned at the rear axle to simplify notation.
The state to be estimated is composed of velocity and angular rate of the vehicle:

$$x_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix} \tag{5.1}$$

The wheel encoders are read with a sampling rate of $f_w = 50Hz$ and return the number of encoder spokes that passed by the optical sensor since the last measurement. Together with the wheel radius r_w and the known sample interval $t_s = 1/f_w$, the distance travelled by the wheel can be calculated:

$$d_w = 2\pi r_w t_s \tag{5.2}$$

The measurement vector consists of the four measured wheel distances:

$$z = \begin{pmatrix} d_{fl} \\ d_{fr} \\ d_{rl} \\ d_{rr} \end{pmatrix} \tag{5.3}$$

Due to the mechanical properties of the platform, control information u_t is not sufficiently accurate to aid the filter prediction step with any meaningful information. Taking into account the comparably slow dynamics of the vehicle and in the absence of other information, it can be assumed to keep velocity and angular rate constant. The process model equation is therefore very simple, with no change in the estimated state taking place:

$$\hat{x_t} = \hat{x}_{t-1} \tag{5.4}$$

Because of the simple process model, the covariance prediction step simplifies to:

$$\hat{P}_t = \hat{P}_{t-1} + Q_t \tag{5.5}$$

For the measurement update, the function $h(\hat{x}_t)$ has to be defined, which returns the expected measurements for the given state estimate $\hat{x}_t$. From the geometrical model follows:

$$h(\hat{x}_t) = \begin{pmatrix} d_{fl} \\ d_{fr} \\ d_{rl} \\ d_{rr} \end{pmatrix} = \begin{pmatrix} \sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)^2 + l^2}\hat{\omega}_t \\ \sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)^2 + l^2}\hat{\omega}_t \\ \hat{v}_t + \frac{1}{2}b\hat{\omega}_t \\ \hat{v}_t - \frac{1}{2}b\hat{\omega}_t \end{pmatrix} \tag{5.6}$$

The Jacobian of $h(\hat{x}_t)$ is then:

$$H_t = \frac{\partial h_t}{\partial \hat{x}_t}(\hat{x}_t) \begin{pmatrix} \frac{t_s(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)}{\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)^2 + l^2}} & t_s\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)^2 + l^2} - \frac{t_s v(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)}{\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} + \frac{1}{2}b)^2 + l^2}} \\ \frac{t_s(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)}{\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)^2 + l^2}} & t_s\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)^2 + l^2} - \frac{t_s v(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)}{\sqrt{(\frac{\hat{v}_t}{\hat{\omega}_t} - \frac{1}{2}b)^2 + l^2}} \\ t_s & \frac{1}{2}bt_s \\ t_s & -\frac{1}{2}bt_s \end{pmatrix} \tag{5.7}$$

With the process model and measurement model function specified, the EKF as described in Algorithm 2 can be used to estimate odometry. By modifying the process noise matrix Q_t as well as the measurement noise matrix R_t the filter behavior can be adjusted to account for uncertainties arising during operation.
A simple example of such an adjustment, if one of the wheels is slipping and spinning much faster than others, the corresponding entry in R_t might be set to a high value temporarily to minimize the contribution of this wrong measurement.
Care has to be taken to handle the case where $\omega \approx 0$ as H_t has an infinite discontinuity for $\omega = 0$ due to d_{ICC} being infinitely far away and switching sign at sign switch of ω.

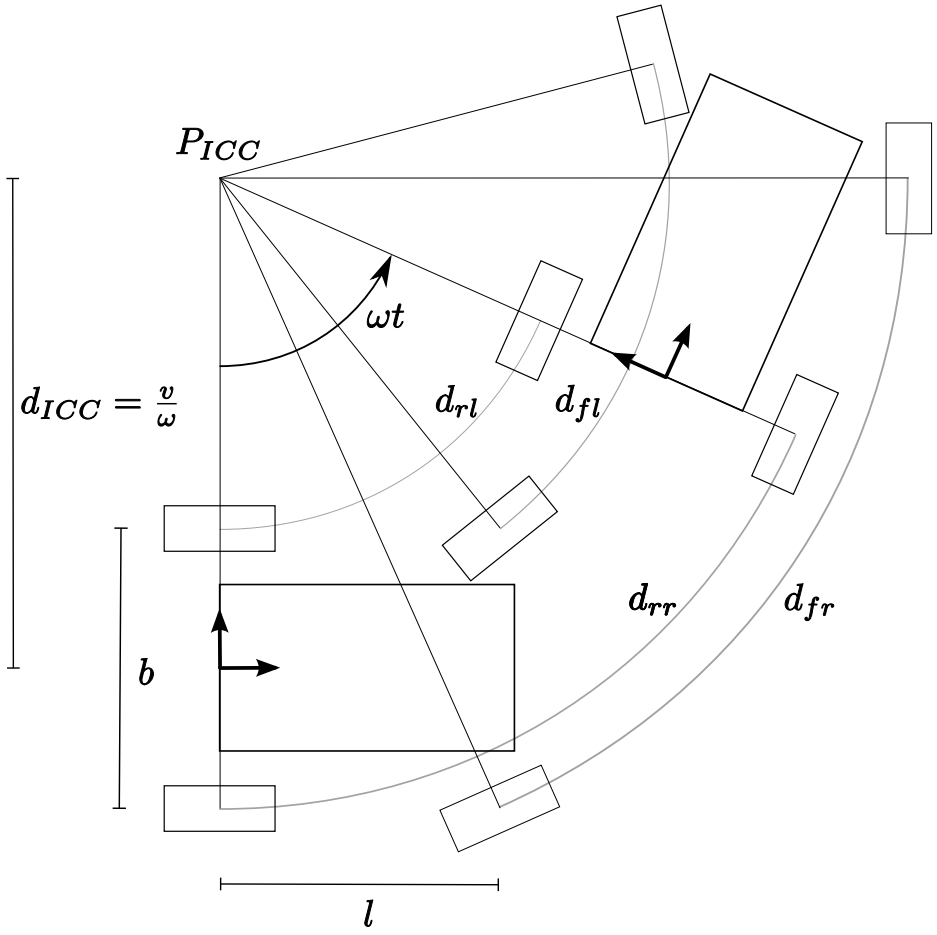


Figure 5.4: Geometry for odometry estimation from 4 wheel encoders

6 Results

As described in chapter 4, two approaches to the USAR SLAM problem are implemented. The particle filter as well as the scan matcher can be used alone or in combination.
Generally, the particle filter can be expected to be more robust as it samples a larger area of the state space compared to the scan matcher. The scan matcher uses the Gauss-Newton approach based on gradient information and thus is prone to converge towards a local minimum if the start estimate is too far from the global minimum. On the other hand, because of using gradient information, the scan matcher can find the maximum likelihood pose with less computational expense than a particle filter, which might need hundreds or thousands of particles and thus evaluations of the observation model to get a good approximation of the maximum likelihood robot pose.
For this reason, both approaches might be combined. A particle filter with a low number of particles can be used to get a rough estimate of the posterior density. The mean of the particle density then might be used as a starting estimate for the scan matcher which is already close to the maximum likelihood pose and leads to quick converges towards the correct pose.

6.1 Experiments

During development, a simulation based on the multi-robot-simulation framework MuroSimF [42] was used for experiments, validation and debugging of the SLAM system. The simulator proved to be an important asset, as experiments and test scenarios could be conducted easily and repeatedly.
For verification in real world scenarios, test runs were done in the Robert Piloty building of the Department of Computer Science at TU Darmstadt. Relevant data for SLAM was recorded in logfiles, yielding data sets that can be used to test different parameter settings and approaches in a repeatable and comparable environment.

6.1.1 Simulator

The UGV simulation provided by MuroSimF incorporates simulated laser scanner, odometry and INS data as well as ground truth. A 3D view of the simulated setting is available. The simulated UGV can transparently be controlled from the RoboGui like the real vehicle.
Using the simulator, arbitrary test cases, environments and robot behavior can be tested without the need to use the real UGV, which takes up time and resources. Figure 6.1 shows the use of the MuroSimF based simulator to test scan matching behavior in long, featureless corridors.

6.1.2 Validation Using the Real Robot

Experiments on the lowest floor of the Robert Piloty building of the department of computer science at TU Darmstadt were performed to test the system and validate the principal scan

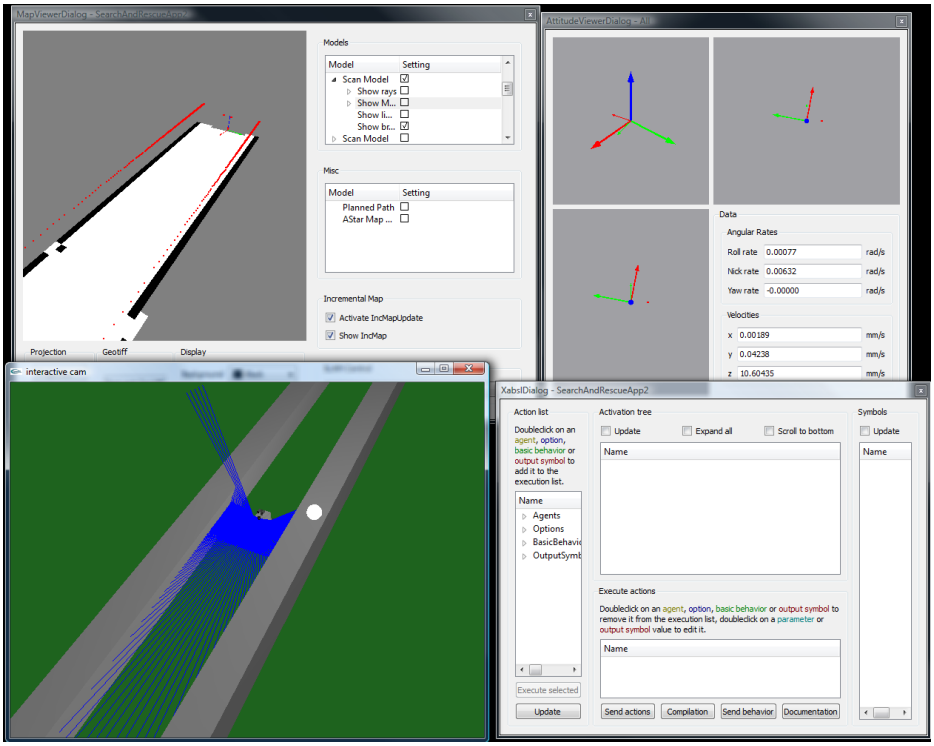
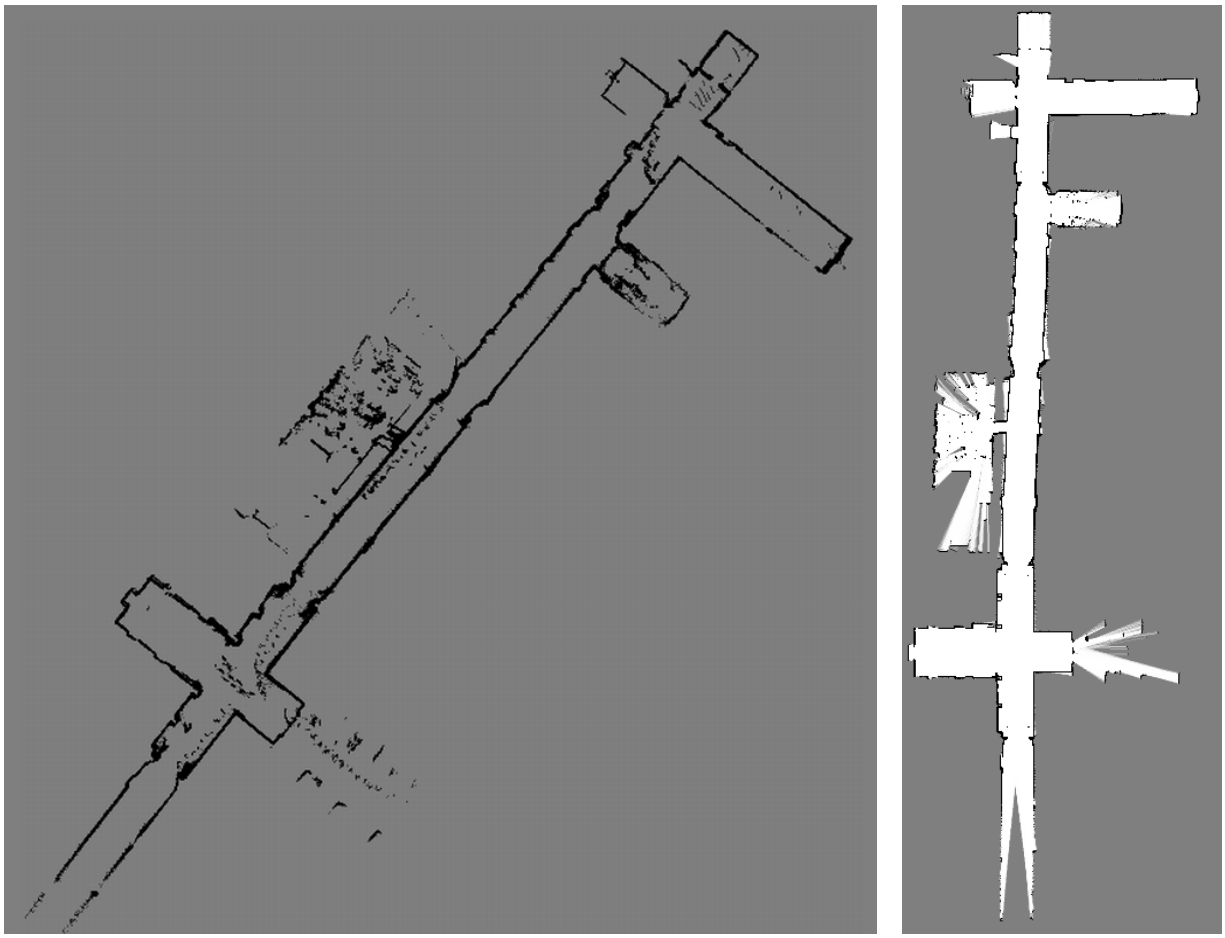


Figure 6.1: Usage of MuroSimF for simulation. The lower left window is the visualization of the simulator world state while the top left window shows the internal robot state. The scenario depicted was used for testing ScanMatching behavior in featureless corridors.

matching and particle filter approach. The environment is more forgiving than the RoboCup Rescue scenario as there is no uneven ground. On the other hand, a larger area than the typical RoboCup Rescue USAR scenario was mapped with occassional people walking around, albeit not in large groups. First results can be seen in figure 6.2(a). Here, occupancy grid mapping using only scan endpoints was used, with people walking around clearly visible in the map as spurious measurements. Even though it contains people, the map is consistent and the approach showed itself to be robust against occasional unmodeled moving obstacles.
Figure 6.2(b) shows a map taken with the full inverse sensor model, marking free space in the map. Due to a bug in the inverse sensor model, some occupied grid cells got overwritten as free during a map update, which explains the lower overall quality of the inverse sensor model map.
A data set incorporating the lower floor of the Piloty building was recorded and used to compare different parameters for the SLAM system in terms of runtime and map quality. The vehicle was remotely controlled for this purpose, using the maximum speed of 1 m/s for the straight parts of the trajectory. Figure 6.3 shows a visualization of the robot trajectory.
Logged data includes the NavSolution, the raw scan data of the UTM-30LX laser scanner, as well as the joint values of the roll/tilt unit of the laser scanner. The logfile has 49004 entries, covers 466 seconds and has a size of 70 megabytes. Most of the entries are NavSolution data, which as described in Section 5.4 gets estimated at a rate of 50Hz. The laser scanner and SLAM module both run at a rate of 20Hz.
As described in [43] and [44], benchmarking the quality and accuracy of a SLAM approach can be a nontrivial task. For the Piloty data set, knowledge about the rectangular and well defined shape enables a basic benchmarking of different maps, even though strict metric measuring is



(a) (b)

Figure 6.2: (a) Map created using scan endpoints in the map only. The regular patterns on some parts of the floor are measurements of people walking by. (b) Map created using inverse sensor model. Refer to Section 6.1 for detailed comments

not applied.
After preliminary testing, four major classes of approaches to the SLAM problem were considered and investigated:

- PF/20: Particle Filter with 20 particles.
- PF/2000: Particle Filter with 2000 particles.
- PF/20+SM: Particle with 20 particles with subsequent ScanMatching.
- SM: ScanMatching only. No odometry information is used here.

For the PF/20+SM approach, first the particle filter is run, with the mean particle pose then being used as the starting estimate for the scan matcher.
For all approaches, an version that updates the map after every SLAM update step (e.g. every processed laser scan gets incorporated into the map) was compared to an version that updates the map only after the robot moved at least $0.7m$ from the last map update location. Earlier experiments suggested that too frequent map updates tend to lower map quality by integrating the noise inherent in the robot state estimation into the map, effectively blurring it.

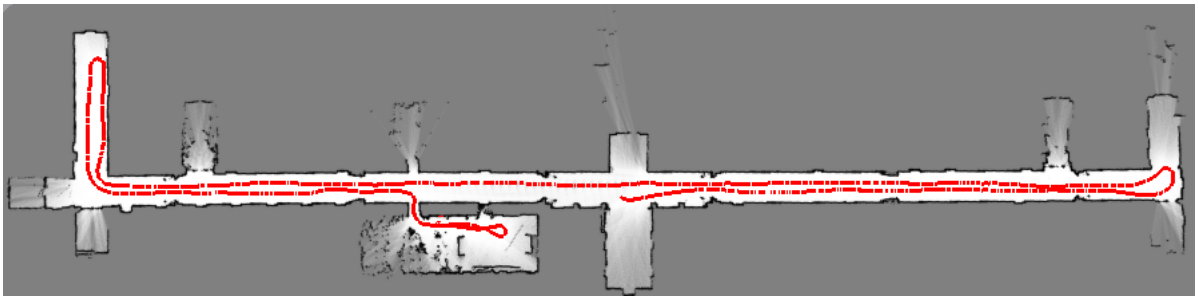


Figure 6.3: The trajectory of the robot for the Piloty data set. The UGV started at the staircase in the middle and moved to the right. It then moved to the opposite side of the building and and finished the run in the robotics lab.

Map Quality

Figure 6.6 shows maps learned by using the particle filter. Using every laser scan for the map update, the map becomes inconsistent at the turning point to the upper right, using 20 as well as 2000 particles.
When updating the map only when the robot moved $0.7m$ from the last map update position, the map stays consistent, even if straight walls of the building are bent in the learned maps. As can be expected, the particle filter using more particles performs better.
Figure 6.7 shows maps learned by scan matching only as well as using the combination of particle filter and scan matching. Walls are aligned better using the scan matcher. The scan matcher using the frequent map update fails due to the map getting ambiguous in the long corridor so a displacement of the estimated robot position along the corridor axis causes the middle staircase to appear twice.
It should be noted that the plain scan matcher without aid by the particle filter also built a consistent map in the case of the $0.7m$ map update version. This means that it is able to track the pose of the vehicle without using any odometry data, suggesting that in an environment with enough structure, an approach using only scan matching can successfully be applied.
The combined approach using the particle filter to generate the starting estimate for the scan matcher is the most successful one. Using this approach, the maps using both update rules are consistent and of good quality. This approach therefore is chosen as the default setting for the SLAM system. Figure 6.5 shows two examples of locations in the Piloty building and their map respresentations.

Runtime Efficiency

For the tests using the Piloty data set, runtime of all approaches was measured. The maps were learned on a platform using a Core 2 Duo P7350 2.0 GHz CPU, running Windows Vista 32 bit. Table 6.1 shows the results. As can be expected, the particle filter with 2000 particles takes the longest time, with a maximum time per iteration of 37ms and an average of 26.1ms. The particle filter approach using 20 particles naturally takes much shorter time.
One iteration of the scan matching approach takes 11 to 16 ms maximum, with the average being 3 to 4 times lower. This large disparity between maximum and average runtime suggests

	PF/20	PF/2000	PF/20+SM	SM
maximum runtime[ms]	2	37	16	11
average runtime[ms]	0.68	26.1	4.1	3.8

Table 6.1: Performance measurements performed on the Piloty data set. The platform used for these measurements is a Intel Core 2 Duo P7350 2GHz running Windows Vista 32 bit. The time is measured from the beginning to the end of a state estimation update.

that the scan matcher has widely varying convergence times.
On the real robot, running Linux on a Core 2 Duo 2.4GHz platform, maximum time for PF/20+SM is 5ms, with the average time per iteration being 1.36ms.
The SLAM system presented in this work can thus easily run in realtime on current computing hardware, with modern laser scanners having a 40-50Hz scan rate it takes on average less than 10% CPU load on a single core. Runtime can be further reduced by using a lower number of beams of the scan for state estimation. Investigation into the cases where the scan matcher takes many iterations might also yield performance improvements.
The system was also tested on the Geode 800LX CPU in the chassis of the vehicle and the URG-04LX laser scanner. Using the most successful PF/20+SM approach, the maximum time measured per iteration was 81ms, with the average being 33ms. The system built locally consistent maps using this approach, but the low range of the URG-04LX limits the quality that can be achieved. Figure 6.4 shows a result. For smaller scale scenarios like those of the RoboCup Rescue competition, better results can be expected.

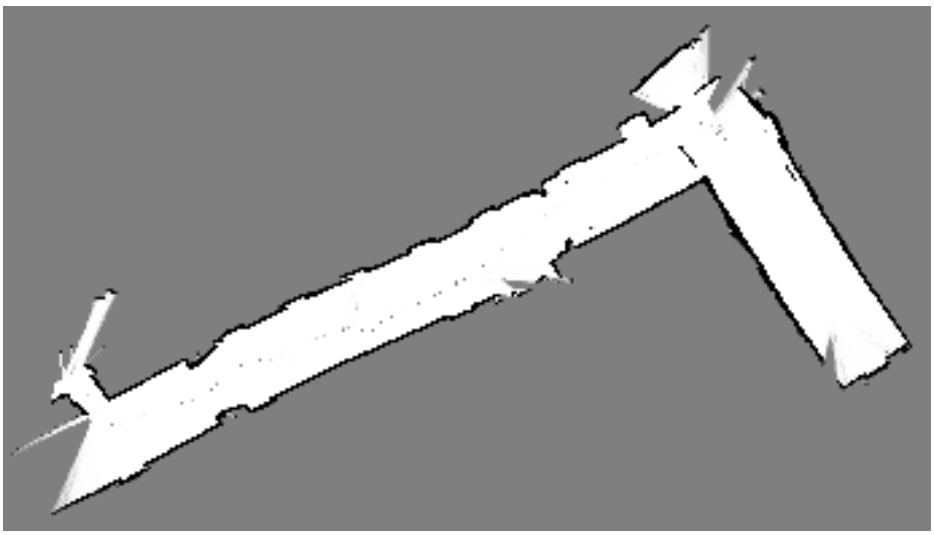
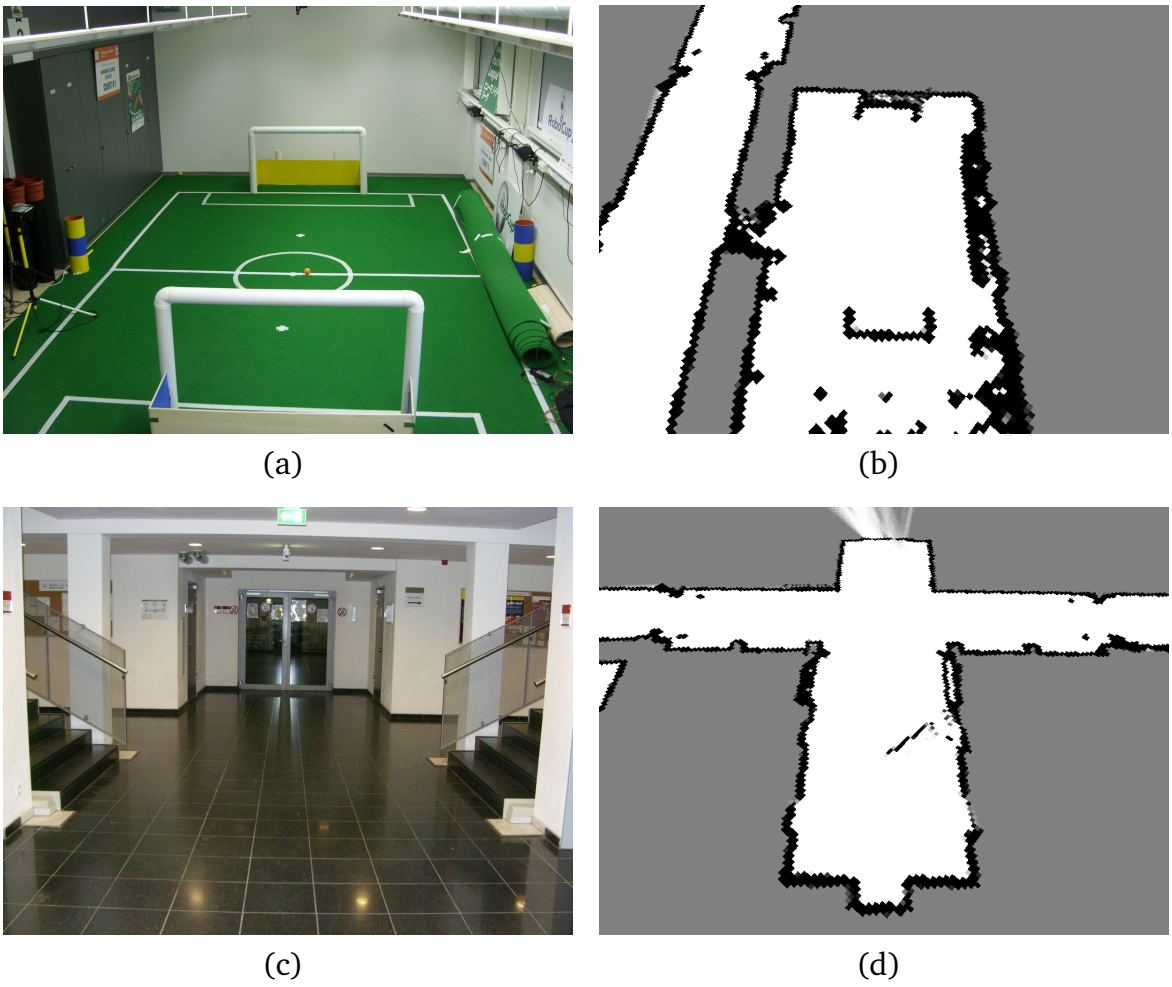


Figure 6.4: Local Map learned by using SLAM on a low power Geode 800LX CPU with a URG-04LX scanner.

6.2 RoboCup Rescue

The UGV as described in chapter 5 was used for the RoboCup Rescue competition at RoboCup 2009 in Graz. The robot managed to map more than half of the rescue arena during the mapping



(a) (b)

(c) (d)

Figure 6.5: Examples of the environment in the Piloty building and the corresponding representation as an occupancy grid map: (a) and (b): The RoboCup Humanoid KidSize playing field in the robotics lab. Note the goals and rolled carpet on the right. (c) and (d): Area around the staircase in the middle of the lower floor. A spurious measurement of the ground is visible in the map on the right.

run, despite large parts of the software still being in the development stage and the generally challenging design of the UGV for use in this environment (high center of mass, soft suspension and thus moving sensors as well as less precise odometry than many other purpose built UGVs). Figure 6.2 shows the map learned during the mapping run. This map was learned using an early version of the scanmatching approach described in chapter 4.5. The robot used a stop and go motion pattern, alternating moving a distance of 50 cm and then stopping for 5 seconds, to give the scanmatcher a stable platform and prevent laser scans transformed with a very noise vehicle attitude estimate to be used for matching.

The early scanmatching approach mistook a local minimum for the correct alignment shortly after the map state depicted in figure 6.8(a). At the lower right corner of figure 6.8(b), the vehicle got stuck but reported a forward velocity in odometry, which lead to the map becoming inconsistent.

The mapping run was recorded in a logfile, so improvements to the SLAM system can be verified and tested with real data. Figure 6.2 shows two maps with different resolutions learned of the

Rescue Arena by using the logfile of the mapping run and the final version of the SLAM system. The maps have no obvious inconsistencies like the one shown in 6.8(b).

6.3 Sick Robot Day

The Darmstadt Rescue Robot team successfully participated in the Sick Robot Day 2009 competition using the UGV described in chapter 5, with some modifications detailed below. The task in this competition is to have the autonomous UGV visit 10 landmarks marked by numbers in either ascending or descending order, while the UGV of a competing team has to do the same starting counting in the opposite direction. UGVs may not touch each other or obstacles. The environment is an enclosed arena with 60cm high walls with the landmarks situated near the walls. The numbers on the landmarks are not known in advance an have to be autonomously identified by the vehicle.
Due to many obstacles being too low for the UTM-30LX scanner to measure, the lower scanner was removed and the UTM-30LX was rotated by a roll angle of 180°, as to scan from a lower position and detect low obstacles. The thermal camera was removed, as it is not needed for this scenario.
For the competition, a landmark model taking the vision system observations as input and estimating the landmark positions as a mixture of gaussians was implemented. As the distance measurement from the monocular camera has low accuracy, ray casting into the map was employed to find the nearest obstacle in the direction of close vision observations.
Using the described approach, our UGV managed to score by reaching one of the number signs, securing the third place in the competition for the Darmstadt Rescue team.

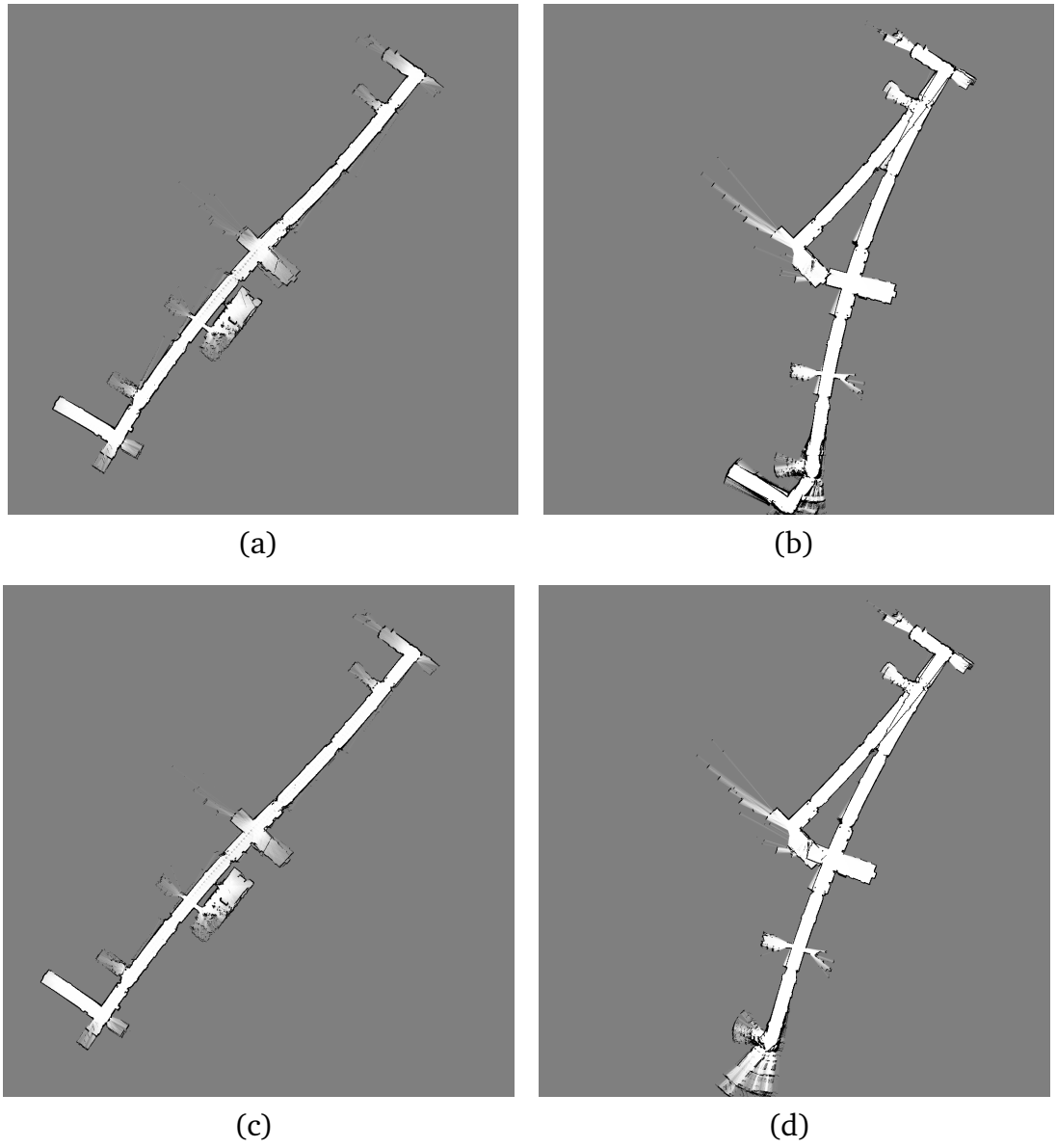


Figure 6.6: Piloty data set processed with the particle filter: (a): Using 20 particles and map update after moving at least 0.7m. (b): Using 20 particles and map update after every iteration.
(c): Using 2000 particles and map update after moving at least 0.7m. (d): Using 2000 particles and map update after every iteration.

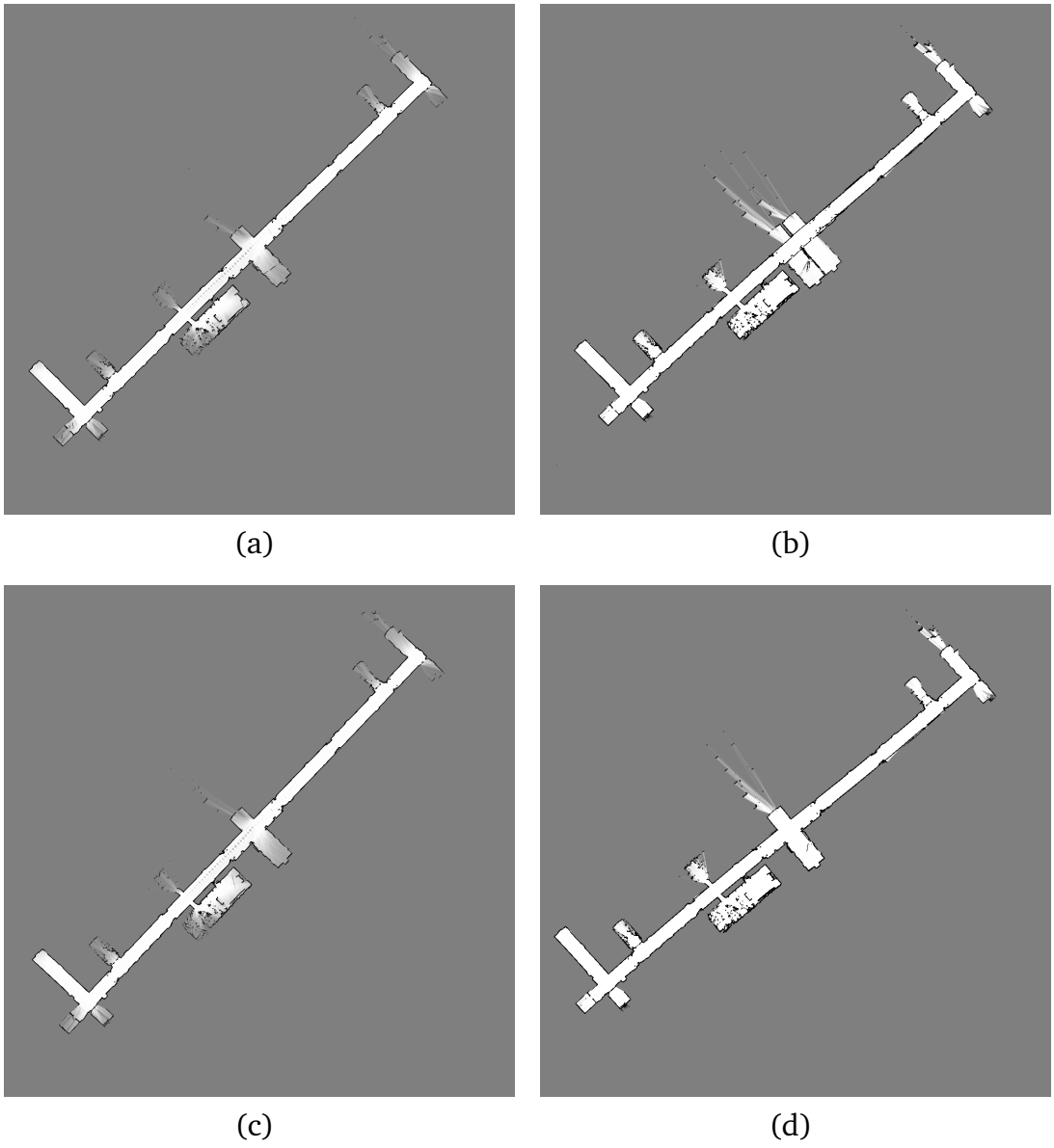
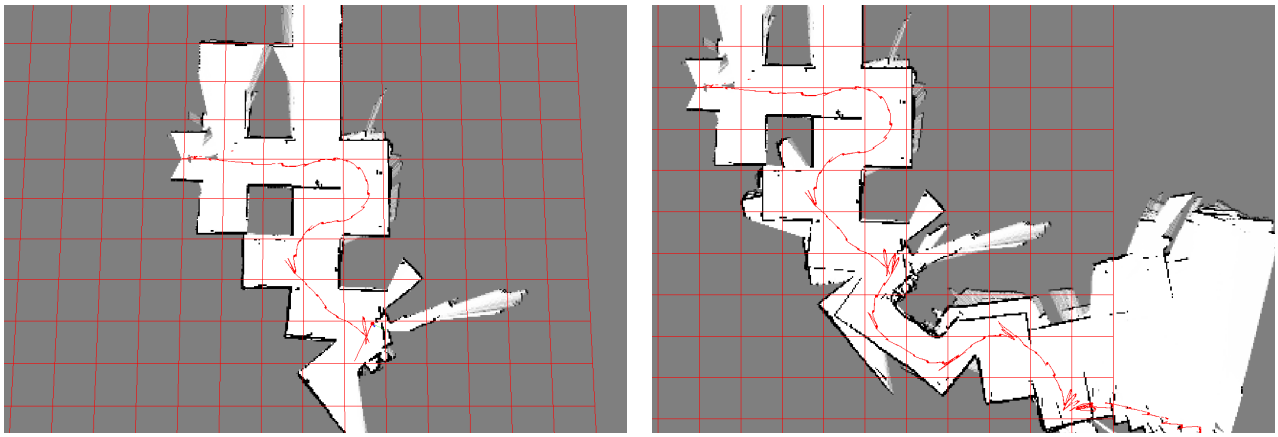
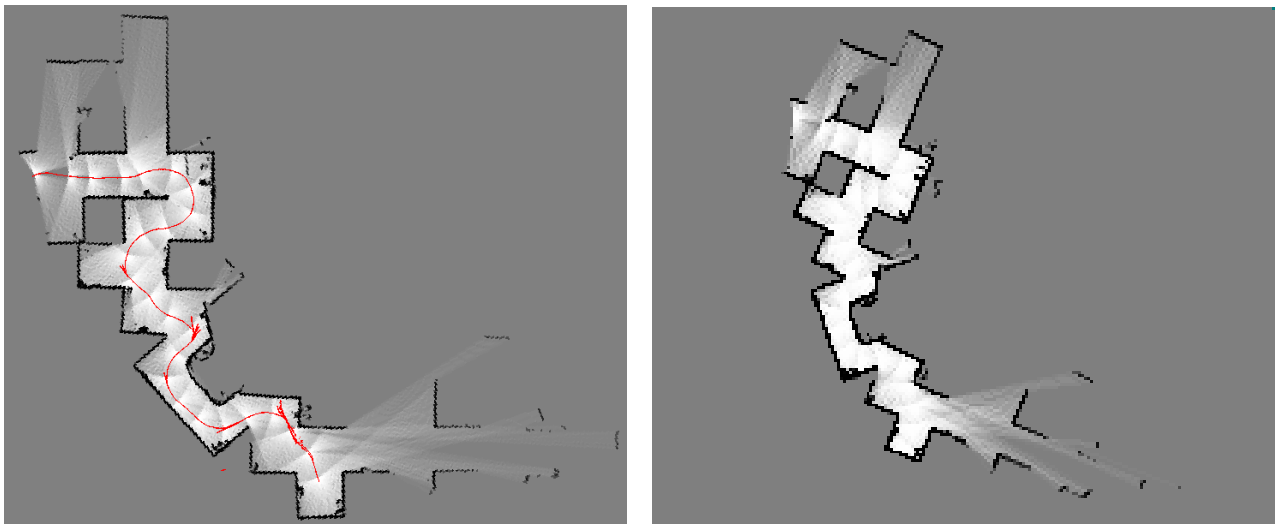


Figure 6.7: Piloty data set processed with scan matching and a combination of scan matching and particle filter: (a): Using the scan matcher ,map update after moving at least 0.7m. (b): Using the scan matcher, map update after every iteration.
(c): Using the particle filter (20 particles) and scan matching, update after moving at least 0.7m. (d): Using the particle filter(20 particles) and scan matching, map update after every iteration.



(a) (b)

Figure 6.8: Results achieved at the RoboCup Rescue autonomy finals at RoboCup2009: (a) Intermediate result (b) Finished map with inconsistent parts after scanmatching failure and the UGV getting stuck



(a) (b)

Figure 6.9: Results with the current SLAM system using logfiles of the RoboCup2009 autonomy challenge : (a) Result with trajectory visualized, grid cell size 50mm. The map was rotated for better visualization. (b) Result using grid cell size 100mm.

7 Conclusions and Outlook

In this work, a platform independent, scalable SLAM system for USAR environments was developed. The requirements as described in chapter 1.3 were fulfilled and validated experimentally. The SLAM system is able to create consistent maps of USAR scenarios, as shown in chapter 6.2. Scalability was proven by running the system on a low power CPU. While degraded in performance, it still gives meaningful results.
Platform Independence is provided by the choice of RoboFrame and was repeatedly demonstrated during development, running on different Windows and Linux platforms.
The system was integrated into an existing behavior and world modeling framework and successfully used in robotics competitions, as described in chapters 6.2 and 6.3.

7.1 Possible Improvements

The loop closing problem was side-stepped as loops in the USAR scenario are generally small enough as to not require explicit loop closing. To make the system more flexible, loop closing and further robustness by using a Rao-Blackwellized particle filter might be implemented by using existing components with comparably low implementation expense.
Another possibility for achieving loop closing capability and higher quality maps is the implementation of a batch optimization algorithm that gets executed asynchronous with the SLAM system described in this work, to provide better map estimates at lower update rate.
For mapping in environments with rough ground conditions, like those encountered in the RoboCup Rescue Arena, having a good estimate of the laser scan attitude is vital. On the current UGV, the transformation has to be estimated from the estimated vehicle attitude and the roll and tilt servo joint values. As the servo joint values are sampled rather slowly at 10Hz, there is potential for improvement in this regard.
A basic 2.5D mapping technique might be implemented by using the second laser scanner tilted downwards. As the localization problem gets solved by the SLAM system with sufficient precision, a 2.5D map can be built using mapping with known poses.

7.2 Integration of the System on a Quadrotor UAV

As mentioned in chapter 1.3, a quadrotor UAV is among the autonomous systems used at TU Darmstadt. Due to the instable dynamics of UAVs of this kind, very accurate attitude estimates are already available as they are needed for control of the vehicle.
As shown in chapter 6.1.2, the scan matcher also works in absence of odometry data, given an environment with enough structure. With the attitude estimation provided by the UAV INS and a Hokuyo UTM-30LX scanner fixed to the body of the UAV, scan attitude estimation will potentially be better than on the UGV described in this work. Using the onboard Intel Atom Z530 CPU, scan matching then can be performed at the 40Hz update rate of the laser scanner.

Integration of the system on the UAV therefore has the potential to enable the UAV to do SLAM in indoor scenarios similar to what was described in the results of this work using the UGV.

List of Figures

List of Algorithms

Glossary

CUDA Compute Unified Device Architecture, 30

DoF Degrees of Freedom, 12

EKF Extended Kalman filter, 7

GPS Global Positioning System, 11
GUI Graphical User Interface, 26

IMU Inertial Measurement Unit, 30
INS Inertial Navigation System, 30

SLAM Simultaneous Localization and Mapping, 1

UAV Unmanned Aerial Vehicle, 1
UGV Unmanned Ground Vehicle, 1
UKF Unscented Kalman filter, 8
USAR Urban Search and Rescue, 1

Bibliography

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, September 2005.

[2] Leonard A. McGee and Stanley F. Schmidt. Discovery of the kalman filter as a practical tool for aerospace and industry. Technical report, National Aeronautics and Space Administration, 1985. NASA-TM-86847.

[3] Larry Matthies, Takeo Kanade, and Richard Szeliski. Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3(3):209–238, 1989.

[4] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, 2004.

[5] Patric Jensfelt and Steen Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. In *IEEE Transactions on Robotics and Automation*, pages 13–22, 1999.

[6] E. Wan and R. van der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control (AS-SPCC)*, Lake Louise, Alberta, Canada, October 2000. IEEE.

[7] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113, 1993.

[8] S. Thrun. Particle filters in robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[9] Hans Moravec and A. E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 116 – 121, March 1985.

[10] John J. Leonard, Hans Jacob S. Feder, Hans Jacob, and S. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Ninth International Symposium on Robotics Research*, pages 169–176. Springer-Verlag, 2000.

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping, 2000.

[12] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[13] C. Stachniss. *Exploration and Mapping with Mobile Robots*. PhD thesis, University of Freiburg, Department of Computer Science, April 2006.

[14] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[15] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25(5/6):403–430, 2005.

[16] Andreas Nüchter. *3D Robotic Mapping. The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom.*, volume 52 of *Springer Tracts in Advanced Robotics*. 2009.

[17] Johannes Pellenz. Robocup 2008 - robocuprescue: Team resko@unikoblenz (germany). Technical report, Universität Koblenz-Landau, 2008.

[18] A. Kleiner. *Mapping and Exploration for Search and Rescue with Humans and Mobile Robots*. Ph.D., University of Freiburg, 2008.

[19] S. Petters, D. Thomas, and O. von Stryk. Roboframe - a modular software framework for lightweight autonomous robots. In *Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA, USA, Oct. 29 2007.

[20] M. Friedmann, K. Petersen, S. Petters, K. Radkhah, Dorian Scholz, D. Thomas, and O. von Stryk. Darmstadt dribblers: Team description for humanoid kidsize league of robocup 2009. Technical report, Technische Universität Darmstadt, 2009.

[21] Marcus Schobbe und Patrick Stamm. Modulare weltmodellierung und kommunikation in heterogenen robotersystemen. Master's thesis, Technische Universität Darmstadt, Computational Engineering, 2007.

[22] M. Risler and O. von Stryk. Formal behavior specification of multi-robot systems using hierarchical state machines in xabsl. In *AAMAS08-Workshop on Formal Models and Methods for Multi-Robot Systems*, Estoril, Portugal, May 12-16 2008.

[23] Benoît Jacob et al. Gaël Guennebaud. Eigen c++ template library for linear algebra. `http://eigen.tuxfamily.org`.

[24] Ronald Garcia. Boost multidimensional array library. `http://www.boost.org/libs/multi_array`, February 2006.

[25] Jeremy B. Maitin-Shepard and Daniel James. Boost unordered library. `http://www.boost.org/libs/unordered`, August 2009.

[26] Siddharth Jain. A survey of laser range finding, 2003.

[27] Tinne De Laet, Joris De Schutter, and Herman Bruyninckx. Rigorously bayesian range finder sensor model for dynamic environments. In *2008 IEEE International Conference on Robotics and Automation, ICRA 2008, May 19-23, 2008, Pasadena, California, USA*, pages 2994–3001, 2008.

[28] Patrick Pfaff, Christian Plagemann, and Wolfram Burgard. Improved likelihood models for probabilistic localization based on range scans. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA, 2007.

[29] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[30] Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[31] Jens Maurer. Boost random number library. `http://www.boost.org/libs/random`, December 2006.

[32] Arnaud Doucet, Nando Defreitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice (Statistics for Engineering and Information Science)*. Springer, 1 edition, June 2001.

[33] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[34] Albert Diosi and Lindsay Kleeman. Fast laser scan matching using polar coordinates. *Int. J. Rob. Res.*, 26(10):1125–1153, 2007.

[35] Dirk Haehnel. *Mapping with Mobile Robots*. PhD thesis, University of Freiburg, 2004.

[36] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[37] Marcelo E. Magallon Milan Ikits and Lev Povalahev. Glew opengl extension wrangler library. `http://glew.sourceforge.net`.

[38] Kyosho Corporation. Kyosho twinforce website. `http://www.kyosho.com/`, October 2009.

[39] Micha Andriluka, Martin Friedmann, Stefan Kohlbrecher, Johannes Meyer, Karen Petersen, Christian Reinl, Peter Schauß, Paul Schnitzspan, Armin Strobel, Dirk Thomas, and Oskar von Stryk. Robocuprescue 2009 - robot league team: Darmstadt rescue robot team (germany). Technical report, Technische Universität Darmstadt, 2009.

[40] Philippe Gerum. Xenomai: Real-time framework for linux. `http://www.xenomai.org`.

[41] Christian Wojek, Gyuri Dorkó, André Schulz, and Bernt Schiele. Sliding-windows for rapid object class localization: A parallel technique. In *Proceedings of the 30th DAGM symposium on Pattern Recognition*, pages 71–81, Berlin, Heidelberg, 2008. Springer-Verlag.

[42] M. Friedmann, K. Petersen, and O. von Stryk. Simulation of multi-robot teams with flexible level of detail. In S. Carpin et al., editor, *Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*, number 5325 in Lecture Notes in Artificial Intelligence, pages 29–40, Venice, Italy, November 2008. Springer.

[43] Johannes Pellenz. Mapping and map scoring at the robocuprescue competition. In *Quantitative Performance Evaluation of Navigation Solutions for Mobile Robots (RSS 08, Workshop CD)*, 2008.

[44] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Journal of Autonomous Robots*, 2009.