To appear in: AI-based Mobile Robots: Case studies of successful robot systems
MIT Press, D. Kortenkamp, R.P. Bonasso, and R.R. Murphy(eds), invited paper

Map Learning and High-Speed Navigation in RHINO

Sebastian Thrun[†] **Arno Bücken** **Wolfram Burgard** **Dieter Fox**
Thorsten Fröhlinghaus **Daniel Hennig** **Thomas Hofmann** **Michael Krell** **Timo Schmidt**

Institut für Informatik III
Universität Bonn
D–53117 Bonn, Germany

[†]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This chapter surveys basic methods for learning maps and high speed autonomous navigation for indoor mobile robots. The methods have been developed in our lab over the past few years, and most of them have been tested thoroughly in various indoor environments. The chapter is targeted towards researchers and engineers who attempt to build reliable mobile robot navigation software.

1 Introduction

Building autonomous mobile robots has been a primary goal of robotics and artificial intelligence. This chapter surveys some of the best methods for indoor mobile robot navigation that have been developed in our lab over the past few years. The central objective of our research is to construct reliable mobile robots, with a special emphasis on autonomy, learning, and human interaction. In the last few years, we have built a mobile robot system, RHINO, which is capable of exploring and navigating in unknown indoor office environments with a speed of approximately 90 cm/sec. While doing so, it can learn metric and topological maps and, based on these, perform all kinds of missions. In addition, RHINO is capable of locating and retrieving objects, delivering them to specific locations or dumping them into trash-bins without human intervention, and giving tours to visitors.

The purpose of this article is to present the key ideas and algorithms underlying our research in a coherent and accessible form, in order to share our experiences and, if possible, provide some guidance for building autonomous mobile robots. The following list summarizes the primary software design principles underlying our approach.

- **Distributed and decentralized processing.** Control is distributed and decentralized. Several on-board and off-board machines are dedicated to several sub-problems of modeling and control. All communication between modules is asynchronous. There is no central clock, and no central process controls all other processes.
- **Any-time algorithms.** Any-time algorithms are able to make decisions regardless of the time spent for computation [12]. Whenever possible, any-time algorithms are employed to ensure that the robot operates in real-time.
- **Hybrid architecture.** Fast, reactive mechanisms are integrated with computationally intense, deliberative modules.
- **Models.** Models, such as the two-dimensional maps described below, are used at all levels of the architecture. Whenever possible, models are learned from data.

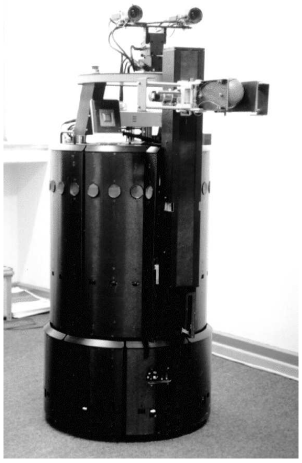

Figure 1: The robot RHINO.

- **Learning.** Machine learning algorithms are employed to increase the flexibility and the robustness of the system. Thus far, learning has proven most useful close to the sensory side of the system, where algorithms such as artificial neural networks interpret the robot's sensors.
- **Modularity.** The software is modular. A plug-and-play architecture allows us to quickly reconfigure the system, depending on the particular configuration and application.
- **Sensor fusion.** Different sensors have different perceptual characteristics. To maximize the robustness of the approach, most of the techniques described here rely on more than just a single type of sensor.

These principles are important, since robots and their environments are complex physical systems. Robot environments are dynamic and inherently unpredictable, and robot hardware often fails or malfunctions, as do our computers and our communication networks. To control a robot reliably, the software must react adequately and timely to sudden changes, failures, delays, or other unforeseen events—which requires special care when designing robot control software.

This chapter focuses exclusively on robot navigation. In particular, it describes our current best approaches for mapping indoor environments, and for high-speed exploration and navigation in dynamic environments. It is organized as follows.

1. **Mapping.** Section 2 describes our method for constructing two-dimensional occupancy grids using sonar sensors and the cameras. Artificial neural networks are used to interpret sonar measurements. On top of the grid-based maps, more compact topological maps are constructed that facilitate fast planning.
2. **Localization.** Localization is the problem of aligning the robot's coordinate system with the global world coordinates. Section 3 describes algorithms for self-localization and position tracking.
3. **Navigation.** Approaches to global path planning, exploration and reactive collision avoidance are described in Section 4.

Most of the software has been developed using a B21 mobile robot manufactured by Real World Interface Inc. The robot, shown in Figure 1, is a synchro-drive robot equipped with a stereo camera system and 24 ultrasonic transducers (in short: sonars). Its maximum velocity is 90 cm/sec. Our robot is equipped with two on-board 486 personal computers that are connected via a radio Ethernet link to several SUN Sparc stations. Notice that currently some of the processing is done off-board. For robots equipped with Pentium computers, the entire software is run locally on the robot (except for the stereo vision system). A second radio link communicates video images to a Datacube, a special-purpose machine for processing images in real-time. The robot is also equipped with active infra-red proximity sensors and tactile sensors, which are currently being incorporated into our map building and collision avoidance routines.

2 Mapping

Mapping refers to the process of constructing a model of the environment based on sensor measurements. This section describes an approach that integrates *grid-based* and *topological* representations. Grid-based approaches, such as those proposed by Moravec/Elfes [31] and Borenstein/Koren [4] and many others, represent environments by evenly-spaced grids. Each grid cell contains a value which indicates the presence or absence of an obstacle in the corresponding region of the environment. Topological approaches, such as those described in [15, 25, 26, 28, 35, 47], represent robot environments by graphs. Nodes in such graphs correspond to distinct situations, places, or landmarks (such as doorways). They are connected by arcs if there exists a direct path between them.

As argued in more detail elsewhere [45], both grid-based and topological representations exhibit orthogonal strengths and weaknesses: Grid-based maps are considerably easier to learn, partially because they facilitate accurate localization, partially because they are easy to maintain. Topological maps, on the other hand, are more compact and thus facilitate fast planning.

2.1 Grid-Based Maps

The metric maps considered here are two-dimensional, discrete occupancy grids, as originally proposed in [14, 31] and since implemented successfully in various systems. Each grid-cell $\langle x, y \rangle$ in the map has an *occupancy value* attached (denoted by $Prob(occ_{x,y})$), which measures the robot's subjective belief whether or not its center can be moved to the center of that cell (*i.e.*, the occupancy map models the *configuration space* of the robot, see *e.g.*, [27]). This section describes the two major steps in building grid-based maps (see also [43]): *sensor interpretation*, and *integration*. Examples of metric maps are shown in various places in this chapter.

2.1.1 Sonar Sensor Interpretation

Sonar sensors measure approximate echo distances to nearby obstacles, along with noise. To build metric maps, sensor reading must be "translated" into occupancy values $Prob(occ_{x,y})$ for each grid cell $\langle x, y \rangle$. The idea here is to train an artificial neural network using Back-Propagation [38] to map sonar measurements to occupancy values [43, 48]. The input to the network consists of the four sensor readings closest to $\langle x, y \rangle$, along with two values that encode $\langle x, y \rangle$ in polar coordinates relative to the robot (angle to the first of the four sensors, and distance). The output target for the network is 1, if $\langle x, y \rangle$ is occupied, and 0 otherwise. Training examples can be obtained by operating a robot in a known environment and recording and labeling its sensor readings; notice that each sonar scan can be used to construct many training examples for different x-y coordinates. In our implementation, training examples are generated with a mobile robot simulator.

Once trained, the network generates values in $[0, 1]$ that can be interpreted as probability for occupancy. Figure 2 shows three examples of sonar scans (top row, bird's eye view) along with their neural network interpretation (bottom row). The darker a value in the circular region around the robot, the larger the occupancy value computed by the network. Figures 2a&b show situations in a corridor. Here the network predicts the walls correctly. Notice the interpretation of the erroneous long reading in the left side of Figure 2a, and the erroneous short reading in 2b. For the area covered by those readings, the network outputs roughly 0.5, which indicates its uncertainty. Figure 2c shows a different situation in which the interpretation of the sensor values is less straightforward. This example illustrates that the network interprets sensors in the context of neighboring sensors. Long readings are only interpreted as free-space, if the neighboring sensors agree. Otherwise, the network returns values close to 0.5, which again indicates uncertainty. Situations such as the one shown in Figure 2c—that defy simple interpretation—are typical for cluttered indoor environments.

Training a neural network to interpret sonar sensors has two key advantages over hand-crafted approaches to sensor interpretation. First, since neural networks are trained based on examples, they can easily be adapted to new circumstances. For example, the walls in the competition ring of the 1994 AAAI robot competition [42] were much smoother than the walls in the building in which the software was originally developed. Even though time was short, the neural network could quickly be retrained to accommodate this new situation. Secondly, multiple sensor readings are interpreted simultaneously. Most existing approaches interpret each sensor reading individually, one-by-one, which can be problematic in practice. For example, the reflection properties of most surfaces depend strongly on the angle of the surface to the sonar beam, which can only be detected by interpreting multiple sonar sensors simultaneously.

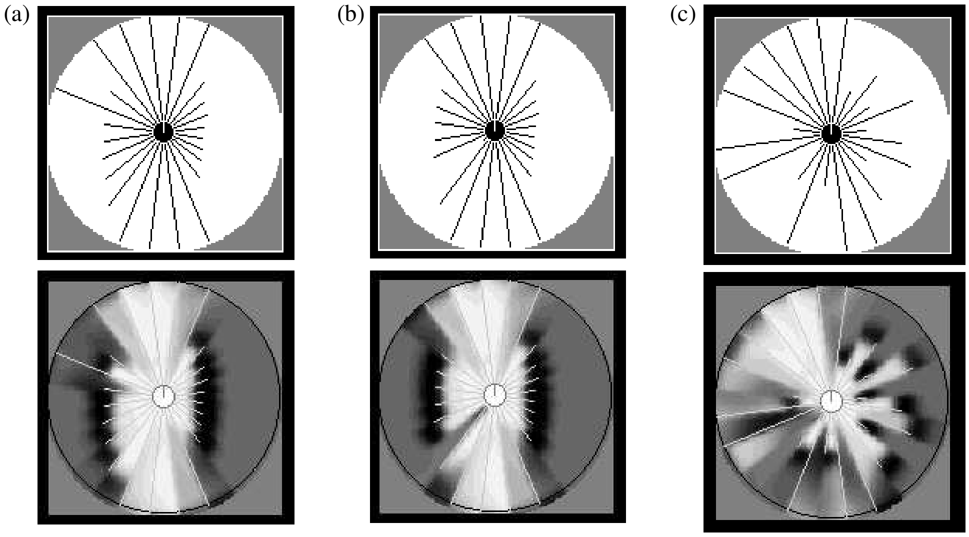

Figure 2: Sonar sensor interpretation: Three example sonar scans (top row) and local occupancy maps (bottom row), generated by the neural network. Bright regions indicate free-space, and dark regions indicate walls and obstacles (enlarged by a robot radius).

2.1.2 Stereo Camera Interpretation

A second source of occupancy information is the stereo camera system, which provides pairs of images recorded simultaneously from different spatial viewpoints. Stereo images are transmitted via a radio link to a *Datacube*, a special-purpose computer for image processing. Like the human visual apparatus, stereo images can be used to compute depth information (*i.e.*, proximity). Put shortly, our approach [20, 21] analyzes stereo images for co-occurrences of vertical edges. By analyzing the disparity of vertical edges found in both images the proximity of obstacles is estimated, and projected onto a two-dimensional occupancy grid.

Figure 3 illustrates the stereo image analysis. Images are first pre-processed (separately for both channels) using a set of three horizontal *Gabor filters* [39]. Gabor filters, which are similar to local Fourier transforms, basically band-filter images to obtain two local coefficients: The *amplitude* of a certain, filter-specific frequency, and its *phase*. Vertical edges characterized by a sharp horizontal brightness difference yield large amplitudes; their precise location in the image is determined with sub-pixel accuracy based on the phase. Figure 3a depicts one of the images, and Figure 3b shows the vertical edges extracted from this image using Gabor filters. To establish correspondence between the left and the right image, both amplitude and phase information is used to identify the projection of the same real-world edge in both images. Once such a vertical edge has been identified in both images, its disparity (spatial shift between both images) is used to estimate its proximity to the cameras (depth). A simple geometric projection yields the x-y-location of the edge relative to the robot. Figure 3c shows the projected location (bird's eye view) of the edges found in Figure 3b; notice that the information concerning the height of an edge is lost when projecting the edge information onto the two-dimensional plane. To construct the final occupancy grid (like the one shown in Figure 3d), edges are enlarged by a robot radius, and the area between the edge projections and the robot is marked as free-space. The last step relies on the assumption that vertical edges correspond to corners of obstacles; since occupancy maps represent configuration space, these edges are increased by a robot radius. To compensate some of the uncertainty in depth estimation, corners of obstacles are smoothed with a Gaussian, and the free-space between obstacles and the robot is smoothed with a logistic function ($f(x) = (1 + e^{-x})^{-1}$). Notice in our implementation, the Datacube pre-processes images at about 4 Hertz. The entire generation of occupancy maps, most of which is done on a SUN Sparc station, requires currently less than a second.

When comparing the original image (Figure 3a) with the resulting map (Figure 3b), the door posts and two of the posters can clearly be identified in the final map (large dark areas in the map). The region close to the white poster does not contain clear vertical edges; thus the robot fails to identify the white wall. This example illustrates that

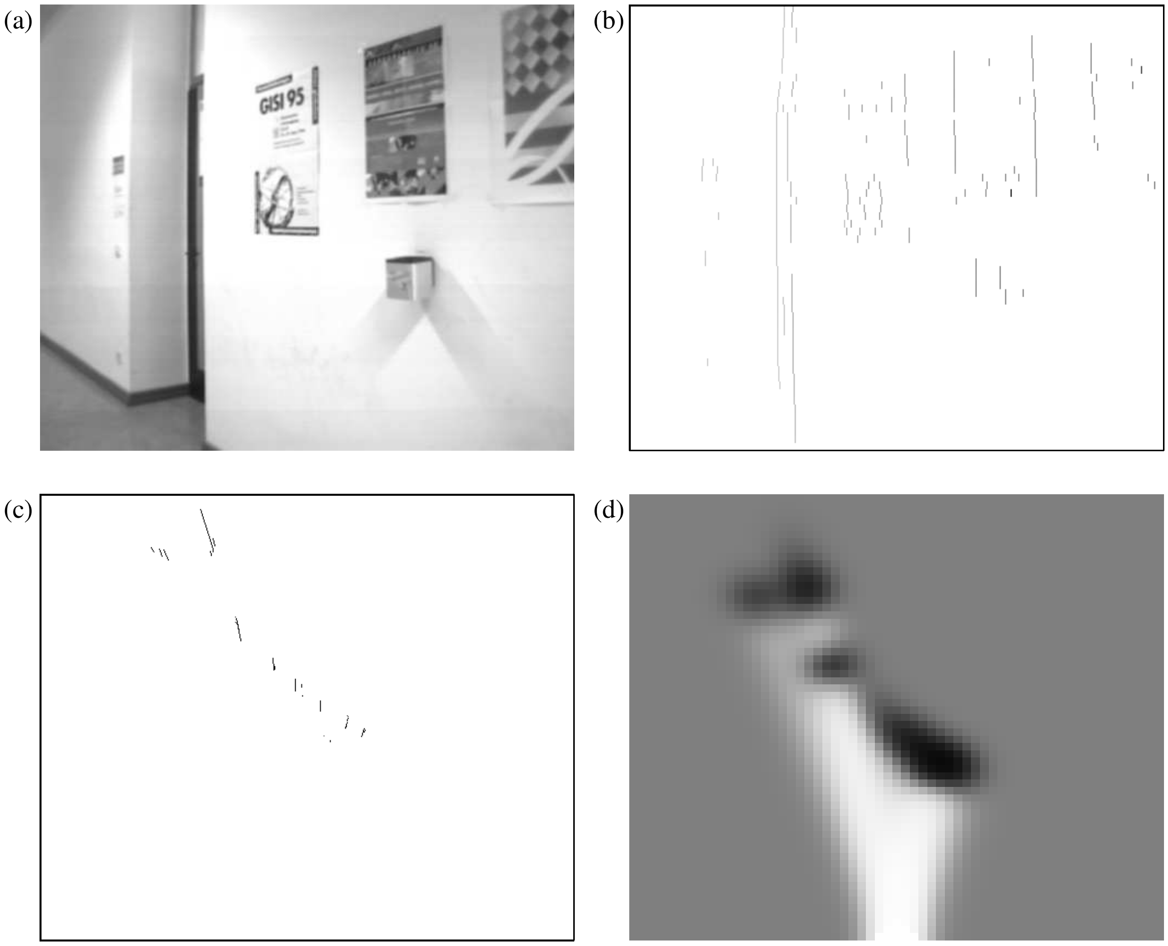

Figure 3: Estimation of occupancy maps using stereo vision: (a) Left camera image, (b) sparse disparity map, (c) two-dimensional edge projections, and (d) local occupancy map.

strictly speaking, occupancy maps derived from stereo vision contain only edges of obstacles—large unstructured obstacles such as walls are "invisible" and hence will not be mapped. Consequently, stereo vision alone would not be sufficient for building accurate maps. On the other hand, stereo vision gives more accurate obstacle information than sonar sensors do, due to the higher resolution of cameras. It also frequently detects obstacles that are "invisible" to sonar sensors, such as objects that absorb sound. As demonstrated below, stereo vision is well-suited to augment sonar information.

2.1.3 Integration Over Time

Sensor interpretations must be integrated over time, to yield a single, consistent map. To do so, it is convenient to interpret the interpretation of the t-th sensor reading (denoted by $s^{(t)}$) as the *probability* that a grid cell $\langle x, y \rangle$ is occupied, conditioned on the sensor reading $s^{(t)}$:

$$Prob(occ_{x,y} | s^{(t)})$$

A map is obtained by integrating these probabilities for all available sensor readings, denoted by $s^{(1)}, s^{(2)}, \ldots, s^{(T)}$. In other words, the desired occupancy value for each grid cell $\langle x, y \rangle$ can be written as the probability

$$Prob(occ_{x,y} | s^{(1)}, s^{(2)}, \ldots, s^{(T)}),$$

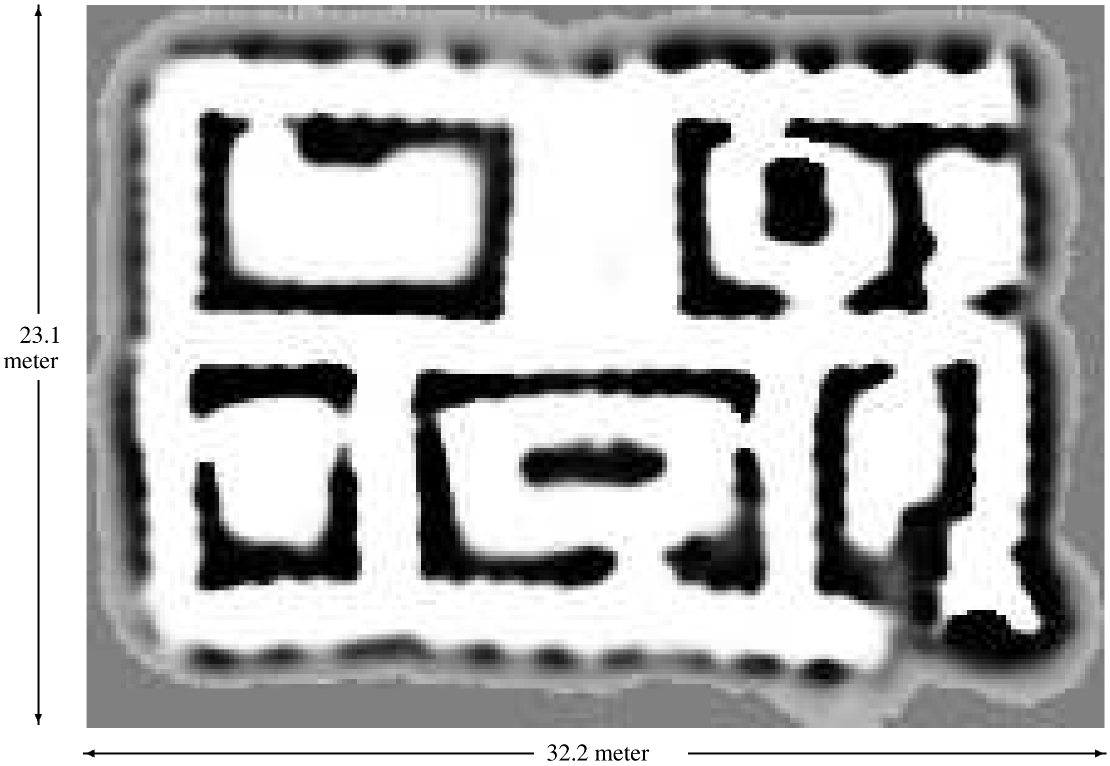

Figure 4: Grid-based map, constructed at the 1994 AAAI autonomous mobile robot competition with the techniques described here.

which is conditioned on *all* sensor readings. A straightforward approach to estimating this quantity is to apply Bayes' rule. To do so, one has to assume independence of the noise in different readings. More specifically, given the true occupancy of a grid cell $\langle x, y\rangle$, the conditional probability $Prob(s^{(t)}|occ_{x,y})$ must be assumed to be independent of $Prob(s^{(t')}|occ_{x,y})$ if $t \neq t'$. This assumption (a Markov assumption [10]) is commonly made in approaches to building occupancy grids. The desired probability can now be computed as follows:

$$\begin{aligned} &Prob(occ_{x,y}|s^{(1)}, s^{(2)}, \ldots, s^{(T)}) \\ &\quad = 1 - \left(1 + \frac{Prob(occ_{x,y})}{1-Prob(occ_{x,y})} \prod_{\tau=1}^{T} \frac{Prob(occ_{x,y}|s^{(\tau)})}{1-Prob(occ_{x,y}|s^{(\tau)})} \frac{1-Prob(occ_{x,y})}{Prob(occ_{x,y})}\right)^{-1} \end{aligned} \tag{1}$$

Here $Prob(occ_{x,y})$ denotes the prior probability for occupancy (which, if set to 0.5, can be omitted in this equation). The derivation of this formula is straightforward and can be found in [31, 34]. Notice that this formula can be used to update occupancy values incrementally.

An example map of a competition ring constructed at the 1994 AAAI autonomous mobile robot competition is shown in Figure 4. This map has been constructed exclusively from sonar information. Figure 5 illustrates the advantage of integrating sonar and stereo vision. All three maps shown there show the same experiment. The map shown in Figure 5a has been constructed purely based on sonar information. While sonar models walls well, it misses the two sound-absorbing chairs in the middle of the hallway. Stereo vision (Figure 5b) detects these chairs, but fails to detect a glass door at the top of the diagram. The map in Figure 5c, which is superior to both single-sensor maps since is contains the chairs and models the walls consistently, is obtained by integrating both maps. Notice that both maps were integrated by taking the *maximum* occupancy value, at each grid cell. Taking the maximum is the most conservative way to integrate maps, since all obstacles are preserved. In principle, the maps could also have been integrated by (1); however, then the result is influenced by the relative frequency of

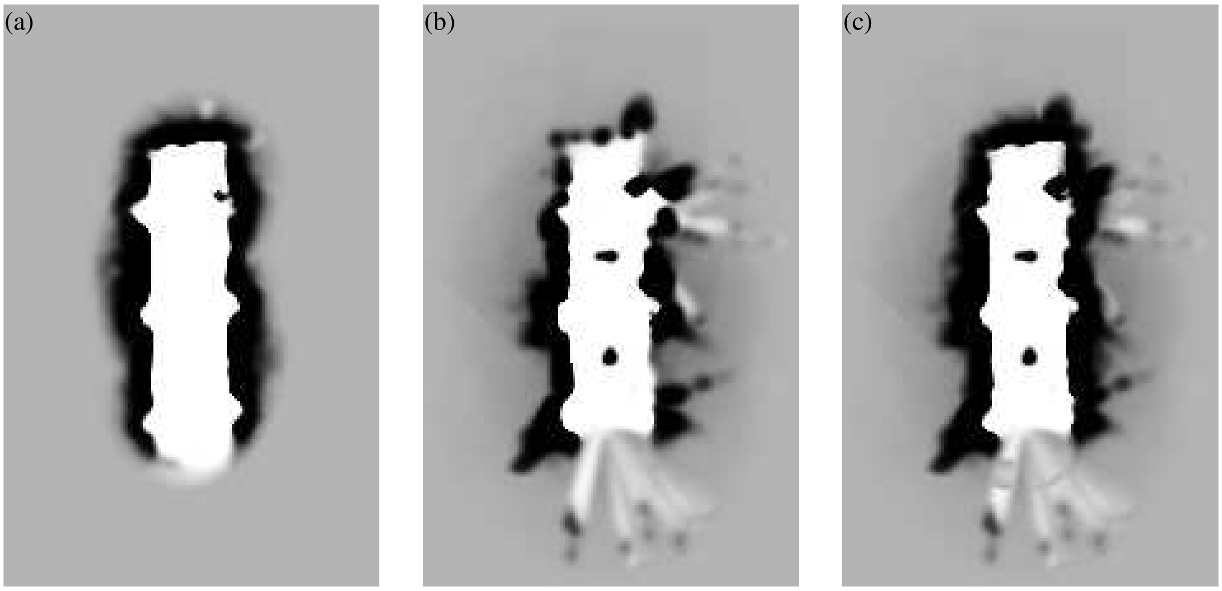

Figure 5: Maps built in a single run (a) using only sonar sensors, (b) using only stereo information, and (c) integrating both. Notice that sonar models the world more consistently, but misses the two sonar-absorbing chairs which are found using stereo vision.

---

sonar and camera measurements which in this example makes the obstacles disappear.

2.2 Topological Maps

Topological maps represent robot environments as graphs, where nodes correspond to distinct places, and arcs represent adjacency. A key advantage of topological representations is their compactness. In our approach, topological maps are built on top of the grid-based maps. The basic idea is simple but very effective: The free-space of a grid-based map is partitioned into a small number of regions, separated by *critical lines*. Critical lines correspond to narrow passages such as doorways. The partitioned map is then mapped into an isomorphic graph. The precise algorithm works as follows:

1. **Thresholding.** Initially, each occupancy value in the occupancy grid is thresholded. Cells whose occupancy value is below the threshold are considered free-space (denoted by C). All other points are considered occupied (denoted by $\bar{C}$).
2. **Voronoi diagram.** Consider an arbitrary point $\langle x, y\rangle \in C$ in free-space. The *basis points* of $\langle x, y\rangle$ are the *closest point(s)* $\langle x', y'\rangle$ in the occupied space $\bar{C}$, *i.e.*, all points $\langle x', y'\rangle \in \bar{C}$ that minimize the Euclidean distance to x, y. We will call these points $\langle x', y'\rangle$ the *basis points of* $\langle x, y\rangle$, and the distance between $\langle x, y\rangle$ and its basis points the *clearance of* $\langle x, y\rangle$. The Voronoi diagram, which is a form of skeletonization [27], is the set of points in free-space that have at least two different (equidistant) basis-points. Figure 6a sketches the Voronoi diagram for the map shown in Figure 4.
3. **Critical points.** The key idea for partitioning the free-space is to find *"critical points."* Critical points $\langle x, y\rangle$ are points on the Voronoi diagram that minimize clearance locally. In other words, each critical point $\langle x, y\rangle$ has the following two properties: (a) it is part of the Voronoi diagram, and (b) there exists an $\varepsilon > 0$ for which the clearance of all points in an ε-neighborhood of $\langle x, y\rangle$ is *not* smaller.
4. **Critical lines.** Critical lines are obtained by connecting each critical point with its basis points (*cf.* Figure 6b). Critical points have exactly two basis points (otherwise they would not be local minima of the clearance function). Critical lines partition the free-space into disjoint regions (see also Figure 6c).
5. **Topological graph**. The partitioning is mapped into an isomorphic graph. Each region corresponds to a node in the topological graph, and each critical line to an arc.

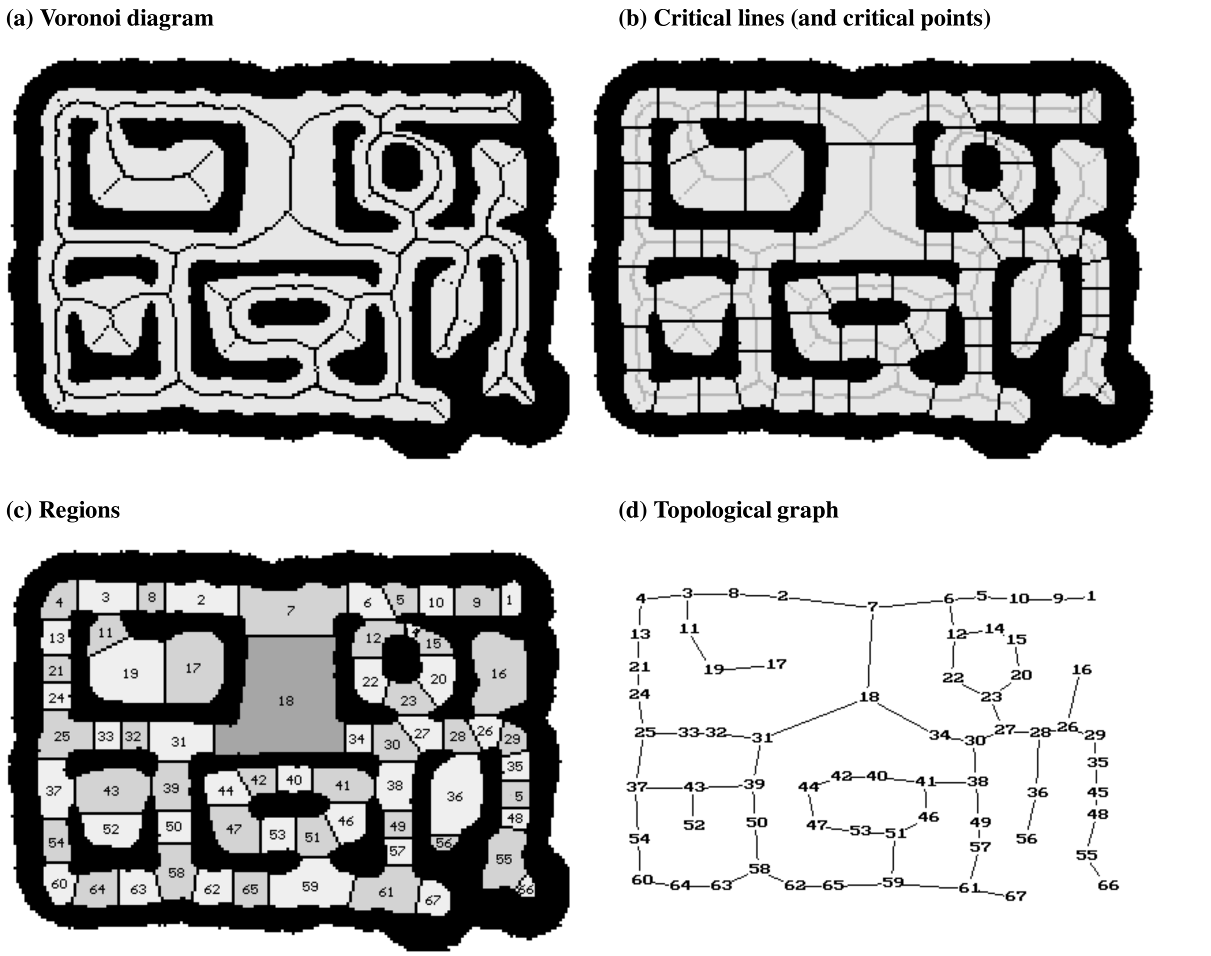

Figure 6: Extracting the topological graph from the map depicted in Figure 4: (a) Voronoi diagram, (b) Critical points and lines, (c) regions, and (d) the final graph. (e) and (f) show a pruned version (see text).

Figure 6d shows an example of a topological graph. The compression is enormous: The topological graph has 67 nodes, whereas the original map contains 27,280 occupied cells. Notice that critical lines are useful for decomposing metric maps primarily for two reasons. Firstly, when passing through a critical line, the robot is forced to move in a considerably small region. Hence, the loss in performance inferred by planning using the topological map (as opposed to the grid-based map) is considerably small. Secondly, narrow regions are more likely blocked by obstacles (such as doors, which can be open or closed).

3 Localization

Localization is the process of aligning the robot's local coordinate system with the global coordinate system of the map. Localization is particularly important (and particularly difficult) for map-based approaches that learn their maps, since the accuracy of a metric map depends crucially on the alignment of the robot with its map [17, 37]. Figure 7 gives an example that illustrates the importance of localization in robot mapping. In Figure 7a, the position is determined solely based on dead-reckoning. After approximately 15 minutes of robot operation, the position error is approximately 11.5 meter. Obviously, the resulting map is too erroneous to be of practical use. Figure 7b is the result of applying the position tracking method described below. In fact, none of the maps presented in the previous section would have been possible without our methods for localizing the robot based on sensor input.

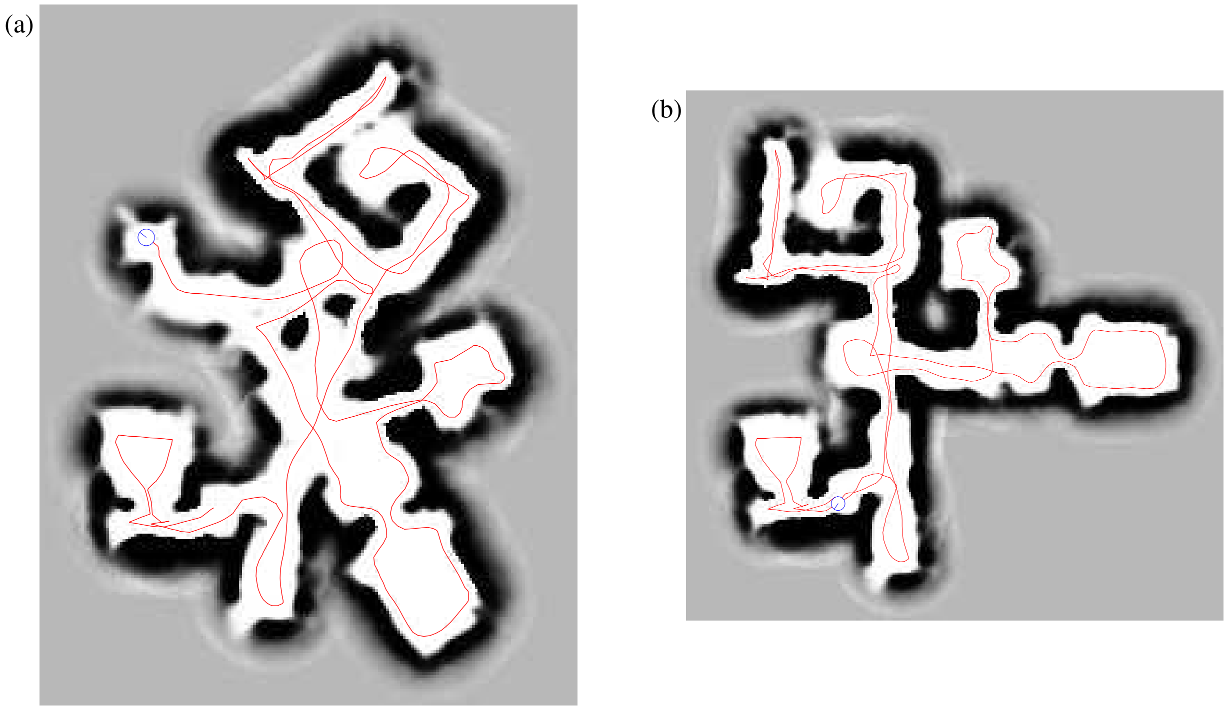

Figure 7: Map constructed without (a) and with (b) the position estimation mechanism described in this article. In (a), only the wheel encoders are used to determine the robot's position. The positional error accumulates to more than 11 meter, and the resulting map is clearly unusable. This illustrates the importance of sensor-based position estimation for map building.

Identifying and correcting for slippage and drift is thus a most important issue in map building.

An excellent overview over different approaches to localization can be found in [17]. Traditionally, localization addresses two sub-problems which are often attacked separately:

1. **Position Tracking.** Position tracking refers to the problem of estimation the location of the robot while it is moving. Drift and slippage impose limits on the ability to estimate the location of the robot within its global map. As Figure 7 demonstrates, even the smallest errors in the robot's odometry can have devastating effects. The problem of position tracking is particularly difficult to solve if mapping is interleaved with localization.
2. **Global localization.** Global localization is the problem of determining the position of the robot under global uncertainty. This problem arises, for example, when a robot uses a map that has been generated in a previous run, and when it is not informed about its initial location within the map.

Global localization and position tracking are two sides of the same coin: localization under uncertainty. In our current implementation, different computational mechanisms are used for localization within a previously learned map (global localization and position tracking), and position tracking when interleaved with exploration and mapping. Each of these approaches exploits the specific properties of the two problems.

3.1 Probabilistic Model

The problem of localization is most generally described in probabilistic terms. Let $l^{(t)}$ denote the location of the robot at time t. For mobile robots, l is usually three-dimensional (x-y location and heading direction). l is not directly accessible, *i.e.*, the robot does not know where it is. Instead, it maintains an internal belief as to where it might be, and uses its sensors periodically to update this belief. It is convenient to denote the belief as a probability density $Prob^{(t)}(l)$ over locations l. The problem of localization is then to estimate $Prob^{(t)}(l)$ so that it matches as closely as possible the true location, $l^{(t)}$. Initially, at time t=0, $Prob^{(0)}(l)$ may be distributed uniformly, assuming that the robot does not know its initial location, or, alternatively, $Prob^{(0)}(l)$ may contain a single peak at $l^{(0)}$ if the robot happens to know its location. $Prob(l)$ is updated whenever the robot senses, using mostly ad hoc approaches to determine the conditional probability $Prob(s^{(t)}|l)$:

$$Prob^{(t)}(l) \quad \longleftarrow \quad \alpha \int Prob^{(t-1)}(l) \cdot Prob(s^{(t)}|l) \; dl \tag{2}$$

Here α is a normalizer, $s^{(t)}$ is the sensor input at time t, and $Prob(s^{(t)}|l)$ is the probability of observing $s^{(t)}$ when at l. Strictly speaking, the update formula (2) is only valid under a conditional independence (Markov) assumption: Given the true location of the robot and the true model of the environment, subsequent sensor readings must be conditionally independent. In practice, we have found this approach to be fairly robust to various kinds of violations of this assumption (dependencies are due to non-stationary in the environment, model errors, crude representations of the space of locations, or unmodeled robot dynamics). However, the key thing to note here is that the problem of localization requires (a) sensors (to obtain $s^{(t)}$) and (b) knowledge about $Prob(s^{(t)}|l)$, *i.e.*, the probability of observing $s^{(t)}$ at l. We currently employ and integrate a variety of different sensor modalities:

- **Wheel encoders.** Wheel encoders measure the revolution of the robot's wheels. Based on their measurements, odometry yields an estimate of the robot's location $l^{(t)}$ which, when expressed relative to the robot's previous location, $l^{(t-1)}$, is impressively accurate. To model errors in odometry, we assume that the position at time t is distributed normally around the very location measured by odometry. The probability $Prob(s^{(t)}|l)$, thus, is normally distributed.
- **Map matching.** As described above, every sensor reading is converted into a "local" map (such as the ones shown in Figures 2 and 3). The robot can localize itself by comparing the global with the local map. More specifically, the *pixel-wise correlation* of the local and the global map—which is a function of the robot's location—is a measure of their correspondence [40]. The more correlated the maps are, the more likely is the corresponding location of the robot. Thus, the probability $Prob(s^{(t)}|l)$ is assumed to be proportional to the correlation of both maps if the robot were at l. See [46] for more details.
- **Sonar modeling.** Another source of information for localization, which we have begun to explore more recently [7], is obtained using a simplistic model of sonar sensors. In essence, it is assumed that each grid cell in the global map possesses a certain probability of being detected by a sonar sensor. More specifically, our model assumes that with probability $Prob(detect_{i,x,y})$ the i-th sonar sensor detects an obstacle at $\langle x, y\rangle$ (where $Prob(detect_{i,x,y})$ is a monotonic function of $Prob(occ_{x,y})$, which is 0 if $\langle x, y\rangle$ does not lie in the perceptual field of the i-th sensor, see [7]). For simplicity, we also assume that the detection probability is independent for different values of i, x, and y.

  Sonar sensors return the distance to the *nearest* obstacle. Thus, the probability that an obstacle is detected at distance d is given by the probability that one or more cells at distance d respond, times the probability that *no* obstacle at distance smaller than d is detected. Put mathematically, let $d(i, \langle x, y\rangle)$ denote the distance of $\langle x, y\rangle$ to the i-th sonar sensor. Then the probability of measuring a specific distance, say s_i, is given by

  $$Prob(s_i) \quad = \quad \beta \left(1 - \prod_{d(i,\langle x,y\rangle)=s_i} (1 - Prob(detect_{i,x,y})) \right) \prod_{d(i,\langle x,y\rangle)<s_i} (1 - Prob(detect_{i,x,y}))$$

  where β is a normalization factor that ensures that the probabilities for different measurements s_i sum up to 1. Here only cells that are in the primary perceptive field of the i-th sonar sensor are considered (a 15 degree cone). Notice that the distribution of $Prob(s_i)$ bears close resemblance to a geometric distribution.

  The probability of observing the entire sonar scan $s^{(t)}$ at l, $Prob(s^{(t)}|l)$, is the product of the individual sensor probabilities:

  $$Prob(s^{(t)}|l) \quad = \quad \prod_{i=1}^{24} Prob(s_i^{(t)}) \tag{3}$$

  Notice that the model of sonar sensors is extremely simplistic, partially because it considers only the main cone of sonar sensors (ignoring the side cones), and partially because it assumes independence between different grid cells, ignoring cumulative effects in the reflection of sound. However, it can be computed very efficiently, which is important if the location of the robot shall be estimated in real-time.

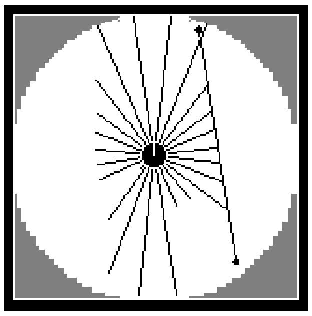

Figure 8: Wall, detected by considering five adjacent sonar measurements. Wall orientations are used to correct for dead-reckoning errors in the robot orientation.

- **Maneuverability.** Assuming the map is correct, the mere fact that a robot moves to a location $\langle x, y \rangle$ makes it unlikely that this location is occupied. Thus, the global map can be used to derive further probabilistic constraints on the robot location. More specifically, our approach assumes that the probability of being at $\langle x, y \rangle$ is proportional to the probability of this grid cell being unoccupied:

  $$Prob(\langle x, y \rangle) \quad = \quad \gamma(1 - Prob(occ_{x,y}))$$

  where γ is an appropriate normalization factor.
- **Wall orientation.** A final source information, which can be used to correct rotational errors, is the *global wall orientation* [11, 22]. This approach rests on the restrictive assumption that walls are either parallel or orthogonal to each other, or differ by more than 15 degrees from these canonical wall directions. In the beginning of robot operation, the global orientation of walls is estimated by searching straight line segments in consecutive sonar measurements (*cf.* Figure 8). Once the global wall orientation has been estimated, it is used to readjust the robot's orientation based on future sonar measurements. See [46] for more details.
- **Landmarks.** Landmarks are used in various approaches to mobile robot localization (see *e.g.*, [3, 25, 32] and references in [44]). We recently have begun to explore mechanisms that enable a robot to select its own landmarks, based on sonar and camera input. The key idea underlying this approach is to train artificial neural networks to recognize landmarks by minimizing the *average localization error* (assuming that update rule (2) is applied in localization). As a result, our robot successfully "discovered" a variety of useful visual landmarks, such as doors, wall color, ceiling lights and so on. Details of the algorithm and performance results are surveyed in [44].

This list of sources for estimating l has been developed over the last few years. Some of these methods make strong assumptions on the correctness of the global map (*e.g.*, the maneuverability method), hence cannot be interleaved with map learning. The reader should also notice that the computational complexity of these approaches differ significantly. For example, the map matching approach, as it is currently implemented, requires extensive comparisons of maps, whereas wall orientations can be determined very efficiently.

3.2 Global Localization

Global localization addresses the problem of mobile robot localization under global uncertainty. To localize the robot globally, the entire density $Prob(l)$ for arbitrary locations l is computed. In our current implementation, $Prob(l)$ is approximated using a grid representation, just like the occupancy grids described in Section 2. The orientation of the robot is represented with $1°$ resolution. Currently, only the wheel encoders, sonar modeling and maneuverability are used to localize the robot. To update $Prob(l)$ in real-time while the robot is in motion, only a subset of sonar readings is currently considered in global positioning, since the process of updating $Prob(l)$ is computationally expensive. From the previous list of sensor modalities, the global localization approach utilizes wheel encoders and sonar sensors, using the sonar modeling and maneuverability approach.

Figure 9a shows a path taken by the robot in the arena depicted in Figure 4 (same arena, different run). As shown in that figure, twelve sonar sweeps were used for the global position estimation, each of which consisted of eight sensor readings. Figure 9b shows the logarithm of the density $Prob(l)$ (maximized over all orientations) after

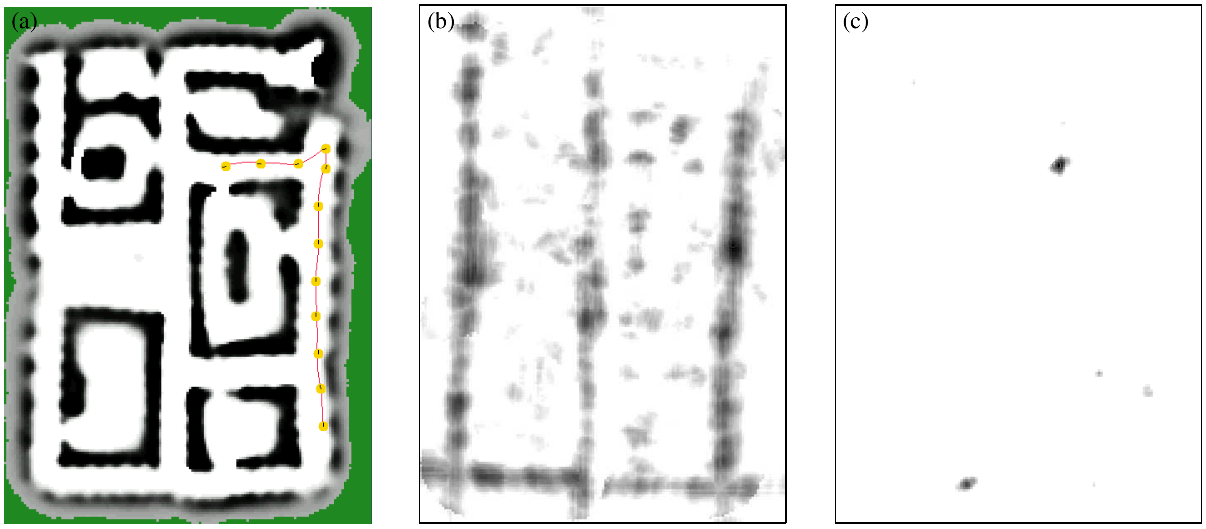

Figure 9: **Initial self-localization:** (a) Path of the robot, (b) $Prob(l)$ after evaluating six sonar readings, and (c) $Prob(l)$ after evaluating twelve sonar readings, both plotted logarithmically.

---

evaluating six of these twelve sensor scans. Although those six readings appear to be insufficient for uniquely localizing the robot, the density indicates that the robot is most likely in a corridor. After evaluating all twelve sensor readings (Figure 9c), the position of the robot is uniquely determined. Notice that the probability of the "correct" grid cell in Figure 9c is approximately 0.96, while the value of the second largest peak is less than $8 \cdot 10^{-6}$. Notice the approach deals adequately with uncertainty and ambiguities, as demonstrated by the empirical examples. The global localization approach has also given very reliable results for real-time position tracking [8]. However, since this approach estimates the robot's location in a previously learned map, it is not applicable during exploration and map learning.

3.3 Position Tracking When Learning Maps

When learning maps, the initial location is known by definition (*e.g.*, is defined to be origin of the global coordinate system). Thus, during exploration, position control seeks to compensate for short-term localization errors such as slippage and drift. The key assumption here is that the position of the robot is known except for some small error, so that instead of estimating an entire probability distribution, it suffices to keep track of the *most likely* location of the robot. Our current best approach for position tracking differs from the above approach to localization in two aspects (*cf.* [43, 46]):

1. The approach estimates only the point l that maximizes $Prob(l)$, instead of the entire density. The advantage of tracking only one value is two-fold: the space of localization does not have to be represented discretely (or by parametric densities), and the approach is computationally much more efficient. It comes at the obvious disadvantage that complex distributions, such as multi-modal distributions, cannot be represented. Thus, once the position is lost, this approach is unable to recover. However, if the initial location is known, we almost never observed that the location was estimated inaccurately.
2. Mainly for historical reasons, our approach to position tracking relies only on wheel encoders, map matching and the wall orientation to estimate location. Position control based on wheel encoders and map matching alone works well if the robot travels through mapped terrain, but ceases to function if the robot explores and maps unknown terrain. The third mechanism, which arguably relies on a restrictive assumption concerning the nature of indoor environments, has proven extremely valuable when autonomously exploring and mapping large-scale indoor environments.

Position tracking is implemented in an any-time fashion, using gradient descent to estimate the location that maximizes $Prob(l)$. When a new sonar reading arrives, the previous gradient search is terminated and its result is incorporated into the current position estimation. In practice, we have found this approach to be fast enough to accurately track the robot position even if the robot is mapping unknown terrain with maximum velocity. Notice that all maps shown in this chapter (with the exception of the map shown in Figure 7a) have been generated using this position tracking approach.

4 Navigation

This section is concerned with robot motion. RHINO's navigation system consists of two modules: A global planner [43], and a reactive collision avoidance module [19, 18]. Control is generated hierarchically: The global path planner generates minimum-cost paths to the goal(s) using the map. As a result, it communicates intermediate sub-goals to the collision avoidance routine, which controls the velocity and the exact motion direction of the robot reactively, based on the most recent sensor measurements only. Both modules adjust their plans/controls continuously in response to the current situation.

Notice that both approaches—the global path planner and the reactive collision avoidance approach—are characterized by orthogonal strengths and weaknesses: The collision avoidance approach is easily trapped in local minima, such as u-shaped obstacle configurations [27]. However, it reacts in real-time to unforeseen obstacles such as humans, and is capable of changing the motion direction while the robot is moving. The global planner, in contrast, does not suffer from the local minimum problem, since it plans globally. It alone, however, is not sufficient to control the robot, since it does not take robot dynamics into account, and since learned maps are incapable of capturing moving obstacles. Thus, global planning alone would simply not avoid collisions with humans and other rapidly moving obstacles.

4.1 Path Planning with Grid-Based Maps

The idea for path planning is to let the robot always move on a minimum-cost path to the goal (or the nearest goal, if multiple goals exist); The cost for traversing a grid cell is determined by its occupancy value. The minimum-cost path is computed using a modified version of *value iteration*, a popular dynamic programming algorithm [2, 24]:

1. **Initialization.** The grid cell that contains the target location is initialized with 0, all others with ∞:
$$V_{x,y} \quad \longleftarrow \quad \begin{cases} 0, & \text{if } \langle x,y \rangle \text{ target cell} \\ \infty, & \text{otherwise} \end{cases}$$
2. **Update loop.** For all non-target grid cells $\langle x,y \rangle$ do:
$$V_{x,y} \longleftarrow \min_{\substack{\xi=-1,0,1 \\ \zeta=-1,0,1}} \{V_{x+\xi,y+\zeta} + Prob(occ_{x+\xi,y+\zeta})\}$$
Value iteration updates the value of all grid cells by the value of their best neighbors, plus the costs of moving to this neighbor (just like A* [33]). Cost is here equivalent to the probability $Prob(occ_{x,y})$ that a grid cell $\langle x,y \rangle$ is occupied. The update rule is iterated. When the update converges, each value $V_{x,y}$ measures the *cumulative cost* for moving to the nearest goal. However, control can be generated at any time, long before value iteration converges.
3. **Determine motion direction.** To determine where to move, the robot generates a minimum-cost path to the goal. This is done by steepest descent in V, starting at the actual robot position. The steepest descent path is then post-processed to maintain a minimum clearance to the walls and, if possible, to move parallel to walls, using the global wall orientation described in the previous section. Determining the motion direction is done in regular time intervals and is fully interleaved with updating V.

Figure 10 shows V after convergence with one and two goals, respectively, using the map shown in Figure 4. The grey-level indicates the cumulative costs V for moving towards the nearest goal point. Notice that every local minimum in the value function corresponds to a goal. Thus, for every point $\langle x,y \rangle$, steepest descent in V leads to the nearest goal point.

Unfortunately, plain value iteration is too inefficient to allow the robot to navigate and learn maps in real-time. Strictly speaking, the basic value iteration algorithm can only be applied if the cost function does not increase (which frequently happens when the map is modified). This is because when the cost function increases, previously adjusted values V might become too small. While value iteration quickly decreases values that are too large,

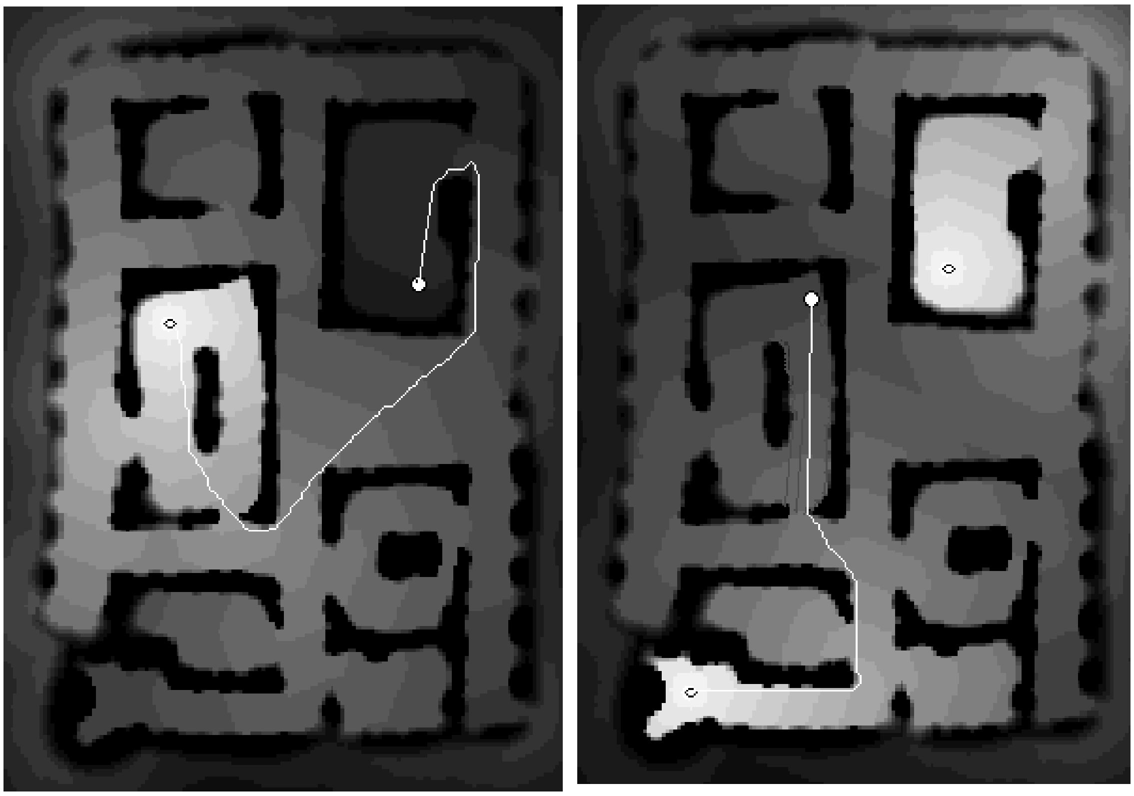

Figure 10: **Path planning with dynamic programming.** Value functions V, computed by value iteration for (a) one goal and (b) two goals (goals are marked by "0"). By following the grey-scale gradient, the robot moves to the next unexplored area on a minimum-cost path.

---

increasing too small a value can be arbitrarily slow [43]. Consequently, the basic value iteration algorithm requires that the value function be initialized completely (Step 1) whenever the map—and thus the cost function—is updated. This is very inefficient, since the map is updated almost constantly. To avoid complete re-initializations, and to further increase the efficiency of the approach, the basic paradigm was extended in the following way:

4. **Selective reset phase.** Every time the map is updated, values $V_{x,y}$ that are too small are identified and reset. This is achieved by the following loop, which is iterated:

   For all non-goal $\langle x, y \rangle$ do:

   $$V_{x,y} \longleftarrow \infty \qquad \text{if} \qquad V_{x,y} < \min_{\substack{\xi=-1,0,1 \\ \zeta=-1,0,1}} \{V_{x+\xi,y+\zeta} + Prob(occ_{x+\xi,y+\zeta})\}$$

   Notice that the remaining $V_{x,y}$-values are not affected. Resetting the value table is a version of value iteration.

5. **Bounding box.** To focus value iteration, a rectangular bounding box $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is maintained that contains all grid cells in which $V_{x,y}$ may change. This box is easily maintained in the value iteration update. As a result, value iteration focuses on a small fraction of the grid only, hence converges much faster. Notice that the bounding box bears similarity to prioritized sweeping [30].

Value iteration is a very general procedure, which has several properties that make it attractive for real-time mobile robot navigation:

- **Any-time algorithm.** As mentioned above, value iteration can be understood as an any-time planner [12]. Consequently, value iteration allows the robot to move in real-time, even though some of its motion commands might be sub-optimal.

- **Full exception handling.** Value iteration pre-plans for arbitrary robot locations. This is because V is computed for every location in the map, not just the current location of the robot. Consequently, the robot can quickly react if it finds itself in an unexpected location, and generate appropriate motion directions without any additional computational effort. This is particularly important in our approach, since the robot uses a fast routine for avoiding collisions (described below) which adjusts the motion direction commanded by the planner based on sensor readings.
- **Exploration.** To autonomously acquire a map, the robot has to explore. For exploration, the same value iteration algorithm is employed, with the only exception that goals correspond to unexplored grid cells. Figure 11a shows an autonomous exploration run. At the current point, the robot has already explored the major hallways and is about to continue to explore an interior room. Circular motion, such as found in the bottom of this plot, occur when two unexplored regions are about equally far away (=same costs). Notice that the complete exploration run shown here took less than 15 minutes. The robot moved constantly and frequently reached a velocity of 80 to 90 cm/sec (see also [6, 19]).

  Figure 11b shows the exploration value function. All white regions are unexplored, and the grey-level indicates the cumulative costs V for moving towards the nearest unexplored point. The value function indicates the robot would continue exploration by moving straight ahead.
- **Multi-agent exploration.** Since value iteration generates values for all grid-cells, it can easily be used for collaborative multi-agent exploration.

In grid maps of size 30 by 30 meter, optimized value iteration, done from scratch, requires approximately 2 to 10 seconds on a SUN Sparc station. In cases where the selective reset step does not reset large fractions of the map (which is the common situation), value iteration converges in less than a second. For example, the planning time in the map shown in Figure 4 lies generally under 2 seconds, and most of the time under a tenth of a second. In the light of these results, one might be inclined to think that grid-based maps are sufficient for autonomous robot navigation. However, value iteration (and similar planning approaches) requires time quadratic in the number of grid cells, imposing intrinsic scaling limitations that prohibit efficient planning in large-scale domains. Due to their compactness, topological maps scale much better to large environments. In what follows we will describe our approach to path planning with topological maps.

4.2 Path Planning with Topological Maps

The enormous compactness of topological maps—when compared to the underlying grid-based map—increases the efficiency of planning. To replace the grid-based planner by a topological planner, the planning problem is split into three sub-problems, all of which can be tackled separately and very efficiently.

1. **Topological planning.** First, paths are planned using the abstract, topological map. Shortest paths in the topological maps can easily be found using one of the standard graph search algorithms, such as Dijkstra's or Floyd/Warshal's shortest path algorithm, A*, or dynamic programming. In our implementation, we used the value iteration approach described in Section 4.1.
2. **Triplet planning.** To translate topological plans into motion commands, a so-called "triplet planner" generates (metric) paths for each set of three adjacent topological regions in the topological plan. More specifically, let $T_1, T_2, \ldots, T_n$ denote the plan generated by the topological planner, where each T_i corresponds to a region in the map. Then, for each triplet $\langle T_i, T_{i+1}, T_{i+2}\rangle$ ($i = 1, \ldots, n-1$ and $T_{n+1} := T_n$), and each grid cell in T_i, the triplet planner generates shortest paths to the cost-nearest point in T_{i+2} in the grid-based map, under the constraint that the robot exclusively moves through T_i and T_{i+1}. For each triplet, all shortest paths can be generated in a single value iteration run: Each point in T_{i+2} is marked as a (potential) goal point, and value iteration is used to propagate costs through T_{i+1} to T_i just as described in Section 4.1. Triplet plans are used to "translate" the topological plan into concrete motion commands: When the robot is in T_i, it moves according to the triplet plan obtained for $\langle T_i, T_{i+1}, T_{i+2}\rangle$. When the robot crosses the boundary of two topological regions, the next triplet plan $\langle T_{i+1}, T_{i+2}, T_{i+3}\rangle$ is activated. Notice that the triplet planner can be used to move the robot to the region that contains the goal location.
3. **Final goal planning.** The final step involves moving to the actual goal location, which again is done with value iteration. Notice that the computational cost for this final planning step does not depend on the size of the map. Instead, it depends on the size and the shape of the final topological region T_n, and the location of the goal.

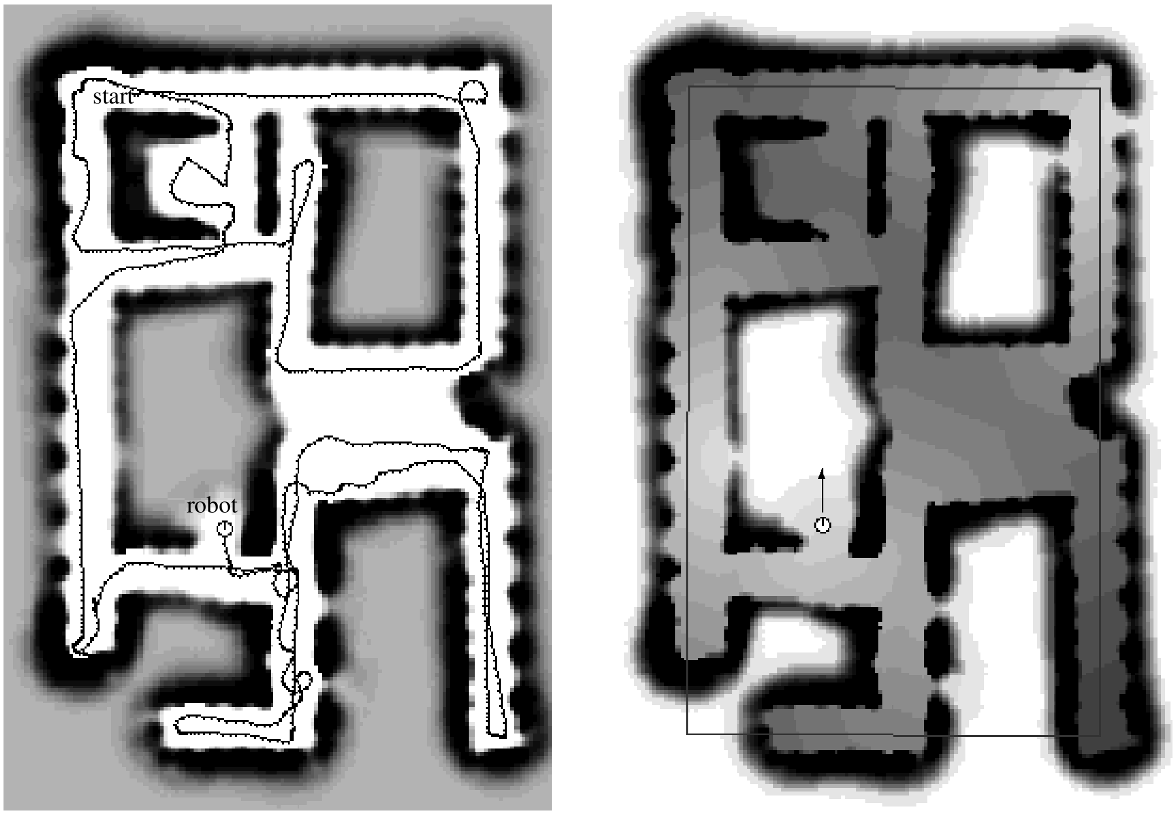

Figure 11: **Autonomous exploration.** (a) Exploration path and (b) value function during exploration. notice that the large black rectangle in (b) indicates the global wall orientation.

The key advantage of this decomposition is that all the expensive computation required for path planning can be done off-line, for all path planning problems. As documented in [45, 46], planning using the topological map is between three and four orders of magnitude more efficient than planning with the grid-based map, for maps similar to those shown in this chapter. On the other hand, plans generated with the topological map are typically between 1% and 4% longer than plans generated using the grid-based map, numbers that are considerably small given the huge computational savings.

4.3 Collision Avoidance

The task of the collision avoidance routine is to navigate the robot to sub-goals generated by the planner, while avoiding collisions with obstacles. It adjusts the actual velocity of the robot and chooses the concrete motion direction. For obvious reasons, the collision avoidance module must operate in real-time. When the robot moves as fast as 90 cm/sec, it is imperative that the robot dynamics (inertia, torque limits) are taken into account, particularly because the path planner considers only robot kinematics. The remainder of this section describes the *"dynamic window approach"* to collision avoidance [19, 18], our currently best collision avoidance routine.

The key idea of the dynamic window approach is to choose control in the *velocity space* of the robot. Figure 12 shows an example of the robot traveling down a hallway with a certain velocity, and Figure 13 shows the corresponding velocity space. The velocity space is a projection of the configuration space (with a fixed kinematic configuration). The horizontal axis in Figure 13 measures the rotational velocity, denoted by ω, and the vertical axis depicts the translational velocity, denoted by v. The actual velocity of the robot is a single point in this diagram (in the center of the white region). The robot sets its velocities in regular time intervals (in our current implementation every 0.25 seconds). To ensure that the robot travels safely and makes progress towards the goal, it has to obey a variety

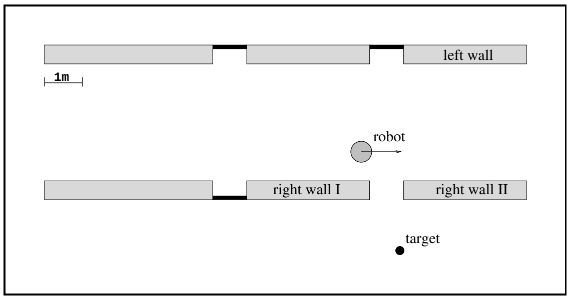

Figure 12. Example situation

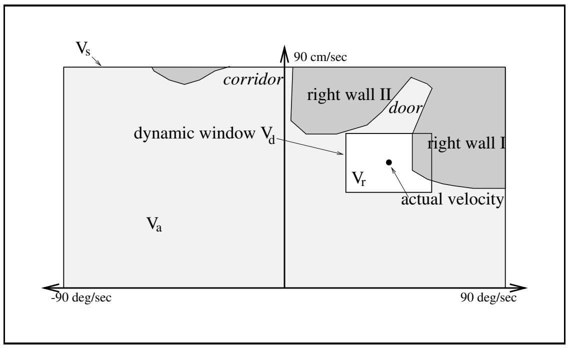

Figure 13. Velocity space

of constraints, which are described in turn.

4.3.1 Hard Constraints

Hard constraints are imposed by the requirement to not to collide with obstacles, and by the dynamics of the robot.

1. **Torque limits.** Torque limits impose bounds as to how the robot might change its velocity in the immediate next decision interval. In the example shown in Figure 13, these bounds are visualized by the rectangle V_d.
2. **Safety.** Obstacles impose additional constraints on the velocity. Velocities with which the robot would inevitably collide, even if decelerated maximally after the decision interval, are not admissible. The dark regions in Figure 13 illustrate such regions in the velocity space. Notice that obstacles such as walls are directly mapped into the velocity space.

These constraints are hard constraints, *i.e.*, for obvious reasons the robot must obey them. In Figure 13, the space of admissible velocities under these constraints is depicted in white color. Notice that these constraints do not specify preferences, neither do they contain goal-related information.

4.3.2 Soft Constraints

Soft constraints impose preferences on the motion direction and velocity of the robot. Currently, three soft constraints are used:

1. **Target heading.** The target heading, denoted *heading*(v, ω), is defined as the absolute angle of the target (sub-goal) relative to the robot's heading direction after 0.25 seconds of robot motion. The target point is usually set by the path planner. If *heading*$(v, \omega) = 0$, the target would be right in front of the robot at the beginning of the next time interval. To make progress towards its target, it is desired that the robot move towards it, *i.e.*, the target heading be as close as possible to zero.
2. **Clearance.** The clearance, denoted by *dist*(v, ω), is defined as the free distance in front of the robot, assuming that the robots sets its velocity once and does not change it thereafter. By maximizing clearance, the robot avoids being close to obstacles.
3. **Translational velocity.** The translational velocity v is also maximized, which causes the robot to always move as fast as permitted by the other constraints.

Notice that all three soft constraints, *heading*(v, ω), *dist*(v, ω), and v, are functions of v and ω. The actual velocity, denoted by $\langle v^*, \omega^* \rangle$, is obtained by maximizing a linear combination of these constraints:

$$\langle v^*, \omega^* \rangle \;=\; \operatorname*{argmax}_{v,\omega} \left(-\alpha_1 \cdot |\text{heading}(v, \omega)| \;+\; \alpha_2 \cdot \text{dist}(v, \omega) \;+\; \alpha_3 \cdot v \right) \tag{4}$$

Here α_1, α_2, and α_3 are constants which trade off the three different soft constraints, thus determine the overall-behavior of the robot. In our approach, (4) is optimized by discrete grid-search.

Figure 14 depicts the *value* (argument of the "argmax" in (4)) corresponding to the situation depicted in Figure 12, as a function of the velocities v and ω. Values that would violate the safety constraint are set to zero. The

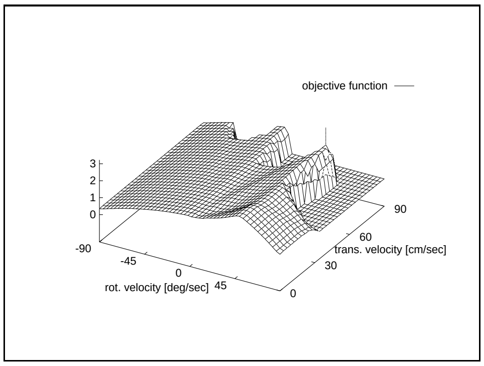

Figure 14. Example of an objective function. The vertical axis depicts value

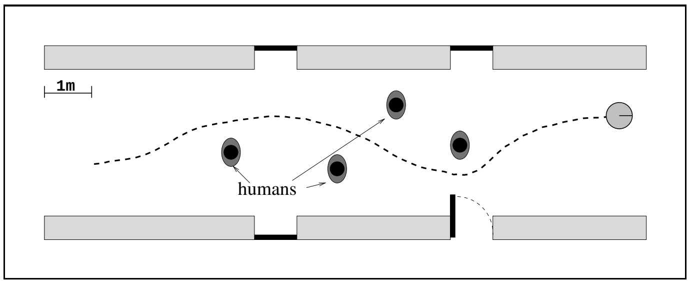

Figure 15. Trajectory through a cluttered corridor.

shape of the remaining function indicates that higher translational velocities are generally preferable, and that the rotational velocity possesses an "optimal" value, which is dictated by the target heading (ridge towards the right in Figure 14). The global maximum, marked by the vertical line in Figure 14, corresponds to a sharp right turn for moving directly through the door (see Figure 12). Notice that the dynamic window (torque limits) is not shown in Figure 14. If the dynamics of the robot do not permit such a turn, the robot would instead move straight, decelerate and eventually backup. Notice that taking the robot dynamics into account is important if the robot travels at more than 30 cm/sec—if the dynamics were ignored, an attempt to turn at high speed easily results in a collision.

4.3.3 Using Sensors

While the constraints are sufficient to generate collision-free robot control if the model of the environment is correct, sensors are erroneous and models err. Thus, our approach employs a variety of additional strategies to recover from errors in perception.

1. **Sensor data.** In reality, exact locations of obstacles are unknown. Maps react very slowly to changes in the environment, they are incapable of modeling fast-moving obstacles, and they typically lag behind due to the computational costs involved in sensor interpretation and localization. Thus, the dynamic window approach works with raw proximity data, as obtained by sonar sensors or computer vision (see below). Every sensor reading can potentially constrain the motion of the robot; however, sensor readings are only memorized for approximately three seconds. Such an approach is maximally conservative in the sense that no sensor reading is ignored. Nevertheless, the relatively short temporal window make it adapt quickly to moving obstacles.
2. **Smoothing.** The objective function is smoothed, to increase the side-clearance of the robot and to decrease the sensitivity to noise in the sensor readings.
3. **Safety margin.** To travel safely at high speed, the robot is enlarged by a safety margin. This safety margin increases with the forward velocity. As a result, the robot keeps safe distances to obstacles when traveling at high speed, while still being able to move through narrow doors with low speed.
4. **Rotate away mode.** Due to sensor noise and changes in the environment, it can happen that every non-zero velocity violates a hard constraint. In such cases, which are rare in practice, the robot turns completely away from the nearest obstacle, from which point on it resumes normal operation.

Figure 15 shows an example of the robot traveling through a cluttered environment, purely based on sonar information. All obstacles in the corridor are smoothly circumvented with a maximal speed of 90 cm/sec. Although in this experiment the robot decelerated to approximately 20 cm/sec when passing through the narrowest passage, it still maintained an average speed of 65 cm/sec. In extensive experiments we found that the dynamic window approach controls the robot very reliably even in populated environments with obstacles that are hard to detect for sonar sensors.

4.4 Real–Time Vision

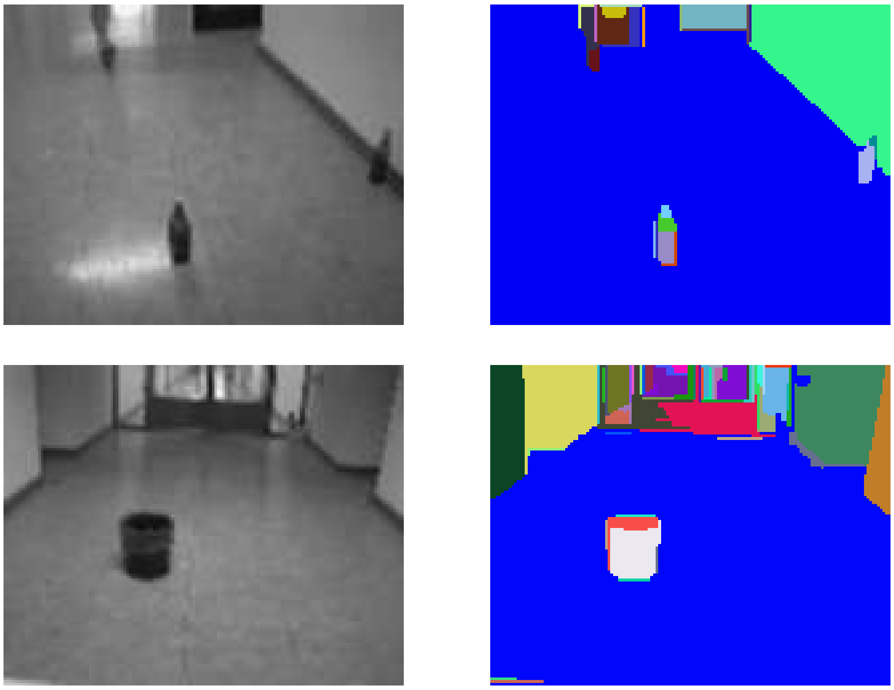

Figure 16: Two indoor scenes with obstacles and the corresponding image segmentation.

Sonars are convenient sensors in that they directly generate proximity information. However, they fail to detect obstacles outside their perceptual range, *e.g.*, small objects on the floor, and they also frequently fail to detect obstacles with sound-absorbing surfaces. To supplement the sonar information, a real–time vision module for obstacle detection is integrated into collision avoidance. Our monocular vision module is able to robustly detect and locate obstacles on the floor, based on image segmentation and discriminant analysis. The main processing stages are as follows.

1. **Pre-processing.** The color image is low–pass filtered and subsampled, typically to 150×120 pixels.
2. **Edge Detection.** The subsampled image is convolved with a gradient operator to detect vertically and horizontally oriented edges.
3. **Pre-segmentation.** A fast, pixel–based image segmentation evaluates the local contrast between pixels and links them together if the contrast is below a certain threshold θ_0 [1]. This pre-segmentation usually results in an over–segmented image with many small region patches.
4. **Segmentation.** Neighboring regions are merged [49], if the mean contrast along their common border is below a threshold θ_t, and if their average color differs by less than a second threshold, σ_t. Both thresholds are iteratively increased $(\theta_{t+1}, \sigma_{t+1}) = (\theta_t, \sigma_t) + (\triangle\theta, \triangle\sigma)$ to a final threshold tuple (θ_T, σ_T).
5. **Interpretation.** The segmented image is interpreted to identify the major floor region and to detect regions that possibly correspond to obstacles. Based on knowledge about the height and the tilting angle of the camera, the robot first locates the horizon within its camera image. It then identifies floor patches using size and location relative to the robot and the horizon as the major criteria. The remaining non-floor regions are obstacle candidates, given that they touch the floor, and given that they fit certain size constraints.
6. **Feature extraction.** For each region of interest, the following features are extracted: (1) the height, (2) the width and (3) the total area, (4/5) two color components (the "U" and "V" channel of the NTSC signal), (6) average brightness and (7) brightness variance; the latter feature provides texture information. Height, width and area are expressed in world-coordinates, computed under the assumption that obstacles extend all the way to the floor.

7. **Recognition.** Finally, regions are classified by a Bayesian classifier. In the "training phase," the feature mean and covariance of typical obstacles that may block the robot's path are estimated from 50 to 100 examples. During recognition, a region is considered an obstacle—and passed to the collision avoidance module—if its Mahalanobis distance to one of the pre-trained obstacles models exceeds a certain threshold. The Mahalanobis distance corresponds to the probability of observing a feature vector assuming Normal distribution [13].

Once a region has been classified as obstacle, its world coordinates are passed to the collision avoidance, to supplement the sonar information. The vision module evaluates images with a frequency of more than 1 Hz on a Pentium computer. Obstacles of the size of a bottle are usually detected at a maximal range of 5 meter, which is sufficient for real-time collision avoidance even if our robot is operated at its maximum speed. Fig. 16 shows two typical images together with random color representations of the segmentation result. In various experiments, we found this algorithm to reliably detect small, can-size obstacles in our university building.

The reader may notice that this algorithm has successfully been used for detecting and grasping free-standing objects on the floor [6], as demonstrated at the 1994 AAAI mobile robot competition [42].

5 Example Application

The RHINO-software described in this paper has served as a low-level platform for various indoor mobile robot applications. A complete coverage of our current applications is beyond the scope of this chapter. One of the most recent and most interesting applications, however, is that of a robotic "tour guide" (similar to the one proposed in [23]). The tour guide offers tours to visitors, explains rooms, locations and their relation to each other. For this purpose, RHINO is equipped with a CD ROM for storing and replaying music and text.

Figure 17 depicts one of the maps used when giving tours through our university building in Bonn. Maps are recorded by tele-operating (joy-sticking) the robot through the building, using some of the techniques described in this chapter. Each location or object of interest (*e.g.*, an office) is taught by moving towards it, and pressing a button when the robot is facing the object/location at the distance of one meter. The grayly shaded numbers in Figure 17 depict 11 target locations, and the corresponding numbers with white background show the positions at which the robot was taught. The entire teach-in requires less then 10 minutes, not counting the time required for recording the verbal explanations of the different tour items. When the robot gives a tour, its localization routines quickly align its position with its previously constructed map. It then sequentially navigates to the target locations, and replays previously recoded text for each of them (with user-controlled levels of granularity). The duration of the complete tour depends on the amount of information the user wants to obtain, and is typically in the order of 5 minutes. If the visitor looses interest, the tour can be terminated at any time. In approximately 30 testing runs in different buildings, we never observed a failure of the navigation routines, even in populated hallways. We found that the integration of speech, sound, interaction and fast motion contributes significantly to the interestingness of the guide.

6 Conclusion

This chapter presents our currently best approaches to autonomous mobile robot navigation. Our current approaches are map-based. In particular, integrated metric-topological maps are learned autonomously using sonar and camera information. Bayesian analysis permits the robot to track its position accurately during navigation and mapping, and to localize itself in cases where it is globally ignorant about its position. Path planning is performed by a fast dynamic programming routine, and collisions are avoided by a module that is capable of reacting to unforeseen obstacles by adjusting the motion direction and the velocity of the robot.

The software has been tested thoroughly using various mobile robots at different sites, and is now distributed and regularly exhibited by a major mobile robot manufacturer (Real World Interface) along with their robots. The software provides the "low level" control that allows several AI researchers inside and outside our University to perform high-level AI experiments, without having to pay much attention to the low-level navigation.

Certainly, there are a variety of limitations and desiderata that warrant future research. The following list addresses some of the most significant and challenging ones:

- **Dynamic environments.** Maps, as presented in this chapter, are generally incapable of modeling moving obstacles. In a recent thesis [41], Schneider extended our approach to model semi-dynamic obstacles such

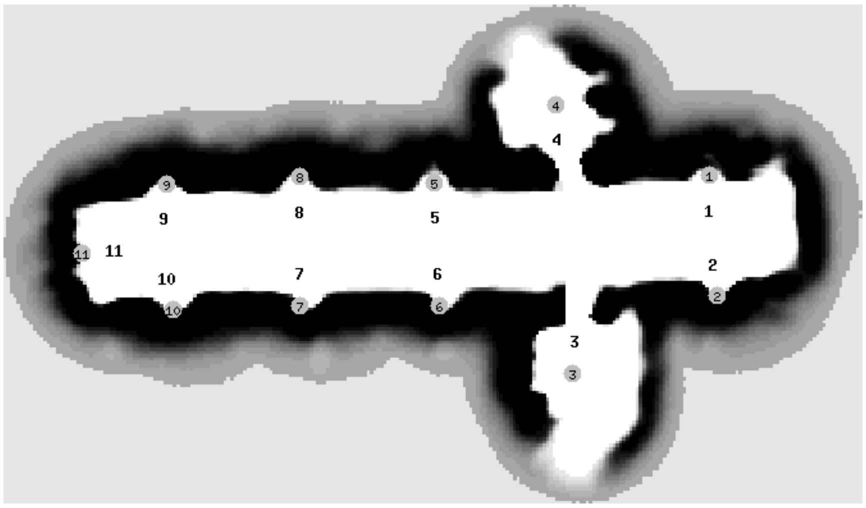

Figure 17: Map used for tours through our building. The numbers in grey circles indicate the different target objects, and the numbers with white background mark the positions at which these targets were taught.

as door. Such obstacles are dynamic but only appear at fixed locations, thus can be detected by analyzing long-term dependencies in the occupancy grid. Modeling moving objects such as humans is a significant open problem in map-based navigation.

- **Three-dimensional maps.** Our current maps are two-dimensional, and our planning algorithms are tailored towards navigation of a circular robot. This chapter does not address manipulation. To facilitate manipulation, three-dimensional representations are clearly advantageous. Extending our current approach to 3D representations is an open problem that clearly warrants further research. Planning using such representations is necessarily more complex, and more sophisticated approaches are probably required if the robot is to plan and act in real-time.
- **Self-tuning sensor interpretation.** Currently, our sensor models are adjusted once and frozen thereafter. It is generally desirable to adjust sensor models while the robot is operating, to compensate for sensor defects and drift.
- **A unified approach to localization.** Our present approach relies upon two quite different methods for localization, one of which is specialized to global position estimation, the other of which is dedicated to position tracking during map learning. Since both attack the same general problem—sensor-based localization under uncertainty—it is desirable to find a single, unified mechanism.
- **Unknown state spaces.** At various occasions throughout this chapter, we made the assumption that the robot environment is Markov and partially observable. In particular, all our probabilistic approaches, such as our methods for mapping and localization, make strong assumptions as to what the non-observable quantities (state) of the environment are that make the environment Markov. Since in general it is difficult to specify what constitutes the "state" of the environment, methods that can discover hidden state and model it from data are clearly desirable (see *e.g.*, [9, 29, 36]).
- **Other sensors.** Integrating sensors other than sonar and cameras into mobile robot navigation is an important problem, since different sensors have different perceptual characteristics. In principle, the general mechanisms for mapping, localization and navigation are not specialized to a particular type sensor. However, incorporating other sensors is not trivial, and we believe many interesting research opportunities will come up from actually trying it.
- **Scaling up.** The largest cycle-free map that has been generated with this approach was approximately 100 meter long; the largest single cycle measured approximately 58 by 20 meter. What happens if the environment is an order of magnitude larger? Clearly, there are intrinsic limits as to how well a robot can localize itself

incrementally, without a global positioning system. We believe that by localizing the robot backwards in time, we can increase the size of the environments that can be mapped reliably. However, the general problem will never disappear, and the best we can hope for are incremental improvements in the size of environments that our software can manage reliably.

As this chapter documents, we have found the map-based paradigm to be surprisingly powerful and reliable. While to date, there exists a variety of successful architecture for mobile robot navigation (such as Brooks's subsumption architecture [5]) each of which is characterized by different advantages and disadvantages, we believe that the map-based paradigm is particularly well-suited for fully autonomous robots that are to perform a multitude of tasks in large indoor environments. Maps are well-understood, and it is easy to specify new navigation tasks using a map. We also believe that probabilistic models, such as the maps described in this chapter, are powerful concepts in robot navigation, as long as they compile percepts into compact representations that capture relevant details and are easy to access.

Acknowledgment

The authors thank the other members of the RHINO team and the XAVIER mobile robot team at CMU for insightful discussion, help and advice. The low-level process communication software TCX [16] was provided by CMU, which is gratefully acknowledged.

One author (S.T.) is sponsored in part by the National Science Foundation under award IRI-9313367, and by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Defense Advanced Research Projects Agency (DARPA) under grant number F33615-93-1-1330 (T. Mitchell, PI). The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of NSF, Wright Laboratory or the United States Government. Another author (T.F.) acknowledges financial support by the European Community under grant CORMORANT 8503 (J. Buhmann, PI).

References

[1] D.H. Ballard and C.M. Brown. *Computer Vision*. Prentice-Hall, 1982.

[2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[3] M. Betke and L. Gurvits. Mobile robot localization using landmarks. Technical Report SCR-94-TR-474, Siemens Corporate Research, Pinceton, December 1993. will also appear in the IEEE Transactions on Robotics and Automation.

[4] J. Borenstein and Koren. Y. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, June 1991.

[5] R. A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2):253, 1989.

[6] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(1), 1995.

[7] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, August 1996. AAAI, AAAI Press/MIT Press.

[8] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Position tracking with position probability grids. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[9] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinction approach. In *Proceedings of 1992 AAAI Conference*, Menlo Park, CA, July 1992. AAAI Press / The MIT Press.

[10] K.L. Chung. *Markov chains with stationary transition probabilities*. Springer Publisher, Berlin, 1960.

[11] J. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 674–680, Scottsdale, AZ, May 1989.

[12] T. L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceeding of Seventh National Conference on Artificial Intelligence AAAI-92*, pages 49–54, Menlo Park, CA, 1988. AAAI, AAAI Press/The MIT Press.

[13] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis*. Wiley, New York, 1973.

[14] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[15] S. Engelson and D. McDermott. Error correction in mobile robot map learning. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 2555–2560, Nice, France, May 1992.

[16] C. Fedor. TCX. An interprocess communication system for building robotic architectures. programmer's guide to version 10.xx. Carnegie Mellon University, Pittsburgh, PA 15213, December 1993.

[17] L. Feng, J. Borenstein, and H.R. Everett. "Where am I?" Sensors and methods for autonomous mobile robot positioning. Technical Report UM-MEAM-94-12, University of Michigan, Ann Arbor, MI, December 1994.

[18] D. Fox, W. Burgard, and S. Thrun. Controlling synchro-drive robots with the dynamic window approach to collision avoidance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[19] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 1996. to appear, also appeared as Technical Report IAI-TR-95-13, University of Bonn, 1995.

[20] T. Fröhlinghaus and J.M. Buhmann. Real-time phase-based stereo for a mobile robot. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[21] T. Fröhlinghaus and J.M. Buhmann. Regularizing phase-based stereo. In *Proceedings of the 13th International Conference on Pattern Recognition*, Vienna, Austria, 1996.

[22] R. Hinkel and T. Knieriemen. Environment perception with a laser radar in a fast moving robot. In *Proceedings of Symposium on Robot Control*, pages 68.1–68.7, Karlsruhe, Germany, October 1988.

[23] I. Horswill. Specialization of perceptual processes. Technical Report AI TR-1511, MIT, AI Lab, Cambridge, MA, September 1994.

[24] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press and Wiley, 1960.

[25] D. Kortenkamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 979–984, Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[26] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Technical report, Department of Computer Science, University of Texas at Austin, Austin, Texas 78712, January 1990.

[27] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[28] M.J. Matarić. Interaction and intelligent behavior. Technical Report AI-TR-1495, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, 1994.

[29] R. A. McCallum. Instance-based state identification for reinforcement learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1995. MIT Press. To appear.

[30] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[31] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.

[32] H. Neven and Schöner G. Dynamics parametrically controlled by image correlations organize robot navigation. *Biological Cybernetics*, 1995. to appear.

[33] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Publisher, Berlin, New York, 1982.

[34] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[35] D. Pierce and B. Kuipers. Learning to explore and build maps. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1264–1271, Menlo Park, July 1994. AAAI, AAAI Press/MIT Press.

[36] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*. IEEE, 1989. IEEE Log Number 8825949.

[37] W.D. Rencken. Concurrent localisation and map building for mobile robots using ultrasonic sensors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2129–2197, Yokohama, Japan, July 1993.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II*. MIT Press, 1986.

[39] T.D. Sanger. Stereo disparity computation using gabor filters. *Biological Cybernetics*, 59:405–418, 1988.

[40] B. Schiele and J. Crowley. A comparison of position estimation techniques using occupancy grids. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1628–1634, San Diego, CA, May 1994.

[41] F. E. Schneider. Sensorinterpretation und Kartenerstellung für mobile Roboter. Master's thesis, Dept. of Computer Science III, University of Bonn, 53117 Bonn, December 1994. In German.

[42] R. Simmons. The 1994 AAAI robot competition and exhibition. *AI Magazine*, 16(1), Spring 1995.

[43] S. Thrun. Exploration and model building in mobile robot domains. In E. Ruspini, editor, *Proceedings of the ICNN-93*, pages 175–180, San Francisco, CA, March 1993. IEEE Neural Network Council.

[44] S. Thrun. A bayesian approach to landmark discovery and active perception for mobile robot navigation. Technical Report CMU-CS-96-122, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213, April 1996.

[45] S. Thrun and A. Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, August 1996. AAAI, AAAI Press/MIT Press.

[46] S. Thrun and A. Bücken. Learning maps for indoor mobile robot navigation. Technical Report CMU-CS-96-121, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213, April 1996.

[47] M. C. Torrance. Natural communication with robots. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1994.

[48] J. W. M. van Dam, B. J. A. Kröse, and F. C. A. Groen. Neural network applications in sensor fusion for an autonomous mobile robot. In L. Dorst, M. van Lambalgen, and F. Voorbraak, editors, *Proc. of Int. Workshop Reasoning with Uncertainty in Robotics*, pages 1–19. Amsterdam, 1995.

[49] S.W. Zucker. Region growing: Childhood and adolescence. *Comput. Graphics Image Processing*, 5:382–399, 1976.