EXPERIENCES WITH AN INTERACTIVE MUSEUM TOUR-GUIDE ROBOT

Wolfram Burgard, Armin B. Cremers, Dieter Fox,
Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz
Walter Steiner, Sebastian Thrun

June 1998
CMU-CS-98-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This article describes the software architecture of an autonomous, interactive tour-guide robot. It presents a modular, distributed software architecture, which integrates localization, mapping, collision avoidance, planning, and various modules concerned with user interaction. The approach does not require any modifications to the environment. To cope with the various challenges in dynamic and ill-structured environments, the software relies on probabilistic computation, on-line learning, any-time algorithms, and distributed control. Special emphasis has been placed on the design of interactive capabilities that appeal to people's intuition. In mid-1997, the robot was successfully deployed in a densely populated museum, demonstrating reliable operation in hazardous public environments, and raising the museum's attendance by more than 50%. In addition, people all over the world controlled the robot through the Web.

Authors' affiliations: W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, D. Schulz, and W. Steiner are with the Computer Science Department III at the University of Bonn. G. Lakemeyer is with the Computer Science Department V at the Technological University of Aachen, both Germany. S. Thrun is with CMU.

This research is sponsored in part by DARPA via AFMSC (contract number F04701-97-C-0022), TACOM (contract number DAAE07-98-C-L032), and Rome Labs (contract number F30602-98-2-0137), and also by the EC (contract number ERBFMRX-CT96-0049) under the TMR programme. 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 DARPA, AFMSC, TACOM, Rome Labs, the United States Government, or the EC.

Keywords: Mobile robotics, probabilistic reasoning, localization, mapping, planning, collision avoidance, logic, human robot interaction, machine learning, entertainment

1 Introduction

Ever since novelist Karel Čapek and science fiction author Isaak Asimov invented the term "robot" [1, 2, 18], the dream of building robots—willing, intelligent and human-like machines that make life pleasant by doing the type work we don't like to do—has been an active dream in people's minds. With universal personal robots still far beyond reach, we are currently witnessing a rapid revolution in robots that directly interact with people and affect their lives [132]. This paper describes one such robot, which is really just a step in this direction. Presented here is the software architecture of an interactive robot named RHINO, which has been built to assist and entertain people in public places, such as museums. RHINO is shown in Figure 1. Its primary task is to give interactive tours through an exhibition, providing multi-modal explanations to the various exhibits along the way (verbal, graphical, sound). In May 1997, RHINO was deployed in the "Deutsches Museum Bonn" (see Figure 2). During a six-day installation period the robot gave tours to more than 2,000 visitors. Through an interactive Web-Interface, people from all over the world could watch the robot's operation and even control its operation—and more than 2,000 did.

On the software side, on which this article focuses, RHINO employs some of the most recent developments in the field of artificial intelligence (AI) and robotics. At its core, RHINO relies upon data-driven probabilistic representation and reasoning to cope with the uncertainties that necessarily arise in complex and highly dynamic environments. RHINO can also learn models (maps) of its environment and change its plans on-the-fly. It is equipped with an easy-to-understand multi-modal user interface, and it can react to the presence of people in various, entertaining ways.

The necessity to employ state-of-the-art AI technology arose from the complexity of the task domain. The majority of RHINO's users were complete novices in robotics; yet, since the typical tour lasted for less than ten minutes, appealing to visitors' intuition was essential for the success of the mission. RHINO's environment, the museum, was densely populated. Most of the time, RHINO was "lucky" in that it lead the way when giving a tour with people following. At times, however, we counted more than a hundred people that surrounded the robot from all sides, making it difficult for the robot to reach the exhibits as planned while not losing its orientation. The museum itself, its geometry and its exhibits, posed further challenges on the software. While there were several narrow passages in the environment in which accurate motion control was essential, most of the museum consisted of wide open spaces that, to a large extent, lacked the necessary structure for the robot to orient itself. One of the key constraints was the necessity to avoid collisions with obstacles at all costs, humans and exhibits alike. Many of the obstacles, however, were literally "invisible," i.e., they could physically not be detected with the robot's sensors. The inability to sense certain obstacles was not necessarily due to the lack of an appropriate sensor suite—in fact, RHINO used four different sensor systems, ranging from laser range finders, sonar, and active infrared sensors to touch-sensitive panels—rather, it was the nature of the obstacles. For example, many exhibits were protected by glass cages, whose very purpose implied that they were not detectable by light-based sensors such as cameras, laser, or infrared, and whose smoothness made it impossible to detect them reliably even with sonar. Other

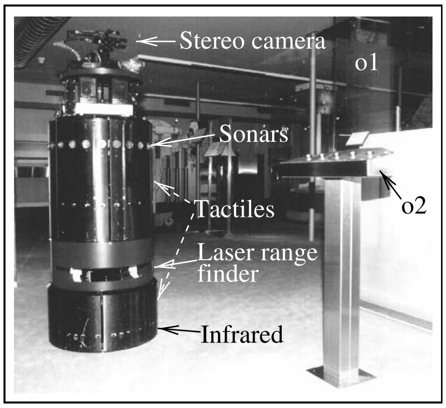

Fig. 1: The robot and its sensors.

Fig. 2: RHINO, pleasing the crowd.

exhibits were placed on solid metal plates, many of which were below the range of our lowest sensors. Not all objects in the museum were static. In particular, the museum provided stools for the visitors to rest, and people tended to leave them behind at random places, preferably close to exhibits. The problem of safe navigation was made more difficult by the speed requirements in the museum. To be interesting to the people, the robot had to move at walking speed whenever possible. At speeds of up to 80 cm/sec the inertia of the robot is significant; turning or stopping on the spot is impossible. Lastly, some of the users were not at all cooperative, imposing further difficulties for the software design. Often, while we were not paying attention, museum visitors tried to "challenge" the robot. For example, by permanently blocking its way, they sometimes sought to make the robot leave the designated exhibition area towards other parts of the museum, where several unmapped and undetectable hazards existed (including a staircase). We quickly learned that one cannot necessarily expect humans to be cooperative, so the safety of the system may not depend on specific behavioral requirements on the side of the users. On the other hand, people are thrilled if robots interact with them—just like they are if people interact with them. Thus, a primary component of a successful tour-guide is the ability to notice the presence of people, and to interact with them in a meaningful, appealing way. In fact, when we interviewed museum visitors, most of them assigned more weight to the robot's interactive capabilities than to its navigational skill.

This article provides an overview of the major components of RHINO's software architecture. As this description of the museum suggests, operating a robot in public environments as complex (and dangerous) as it poses research challenges that go beyond many of those found in most office environments. To cope with them, this paper describes a collection of algorithms which provide the robot with some unique features, such as its ability navigate smoothly and safely at high speed, to determine its location in an unmodified environment and populated, the ability to quickly find detours if paths are blocked, and the ability to engage and interact with people. We believe that these characteristics are

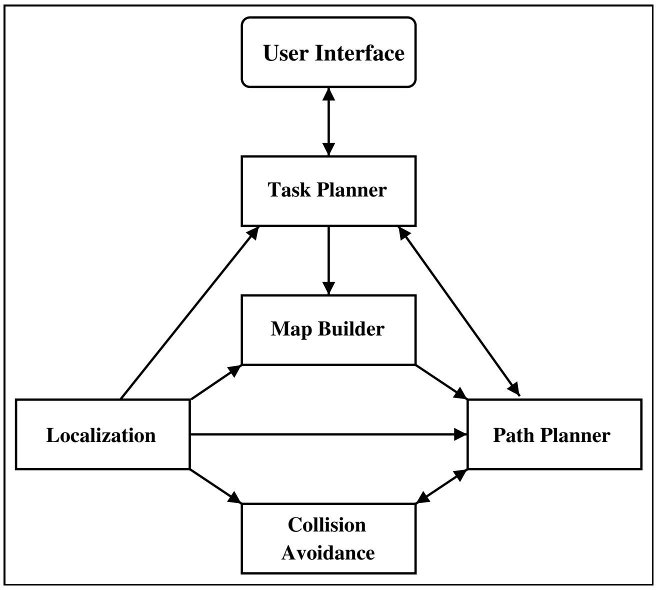

Fig. 3: Major components of the RHINO system and major flow of information.

prototypical for a large variety of application domains for mobile robots, and conjecture that virtually all of the technology described in this paper can be applied to a much larger variety of tasks.

2 Architectural Overview

The overall software architecture consists of approximately 25 modules (processes), which are executed in parallel on 3 on-board PCs and 2 off-board SUN workstations, connected via Ethernet. The software modules communicate using TCX [39], a decentralized communication protocol for point-to-point socket communication. Figure 3 shows the overall software architecture along with the major software modules and the flow of information between them. Similar to other robot control architectures, the RHINO system is also organized in a hierarchical manner, with the device drivers at the lowest level and the user interfaces at the highest. The hierarchy, however, is not strict in the sense that modules would pass information only within the same or across adjacent layers. In RHINO's software, modules often communicate across multiple layer boundaries.

Among the various principles that can be found in RHINO's software system, the following three are the most important ones:

1. **Probabilistic representations, reasoning, and learning.** Robot perception is inaccurate and incomplete. Therefore, robots are inherently unable to determine the state of the world. Probabilistic data structures lend themselves nicely to the inherent uncertainty inside a robot. Instead of extracting just a single interpretation from sensor data, as is traditionally done in the field of robotics, probabilistic methods extract multiple interpretations (often all possible ones), weighted by a numeric plausibility factor that is expressed as a conditional probability. By considering multiple hypotheses, the robot can deal in a mathematically elegant way with ambiguities and uncertainty. In our experience, robots that use probabilistic representations recover easier from false beliefs and therefore exhibit more robust behavior. In addition, probability theory provides nice and elegant ways to integrate evidence from multiple sources over time, and to make optimal decisions under uncertainty. Recently, probabilistic methods have been employed in a variety of successful mobile robots [15, 57, 65, 100, 120], for reasons similar to the ones given here.

2. **Resource flexibility**. Most of RHINO's software can adapt to the available computational resources. For example, modules that consume substantial processing time, such as the motion planner or the localization module, can produce results regardless of the time available for computation. The more processing cycles are available, however, the more accurate, or optimal, the result. In RHINO's software, resource flexibility is achieved by two mechanisms: selective data processing and any-time algorithms [33]. Some modules consider only a subset of the available data, such as the localization routine. Other modules, such as the motion planning module, can quickly draft initial solutions which are then refined incrementally, so that an answer is available when needed.

3. **Distributed, asynchronous processing with decentralized decision making.** RHINO's software does not possess a centralized clock or a centralized communication module. Synchronization of different modules is strictly de-central. Time-critical software (e.g., all device drivers), and software that is important for the safety of the robot (e.g., collision avoidance), are run on the robot's on-board computers. Higher-level software, such as the task control module, is run on the stationary computers. This software organization has been found to yield robust behavior even in the presence of unreliable communication links (specifically the radio link which connected the on-board and off-board computers) and various other events that can temporarily delay the message flow or reduce the available computational time. The modular, decentralized software organization eases the task of software configuration. Each module adds a certain competence, but not all modules are required to run the robot.

The remainder of this paper will describe those software modules that were most essential to RHINO's success.

3 State Estimation

To find its way safely through a populated environment with invisible obstacles, RHINO employs several methods to estimate its current state. State comprises the robot's position and the position of people and obstacles. This section describes RHINO's approach to localization and mapping, both of which use probabilistic estimators for interpreting and integrating sensor evidence.

3.1 Localization

At the core of RHINO's navigation routines is a module that continuously estimates the robot's position in x-y-θ space, where x and y are the coordinates of the robot in a 2D Cartesian coordinate system and θ is its orientation. RHINO employs a variant of *Markov localization*, which is a probabilistic method for robot localization [15, 65, 100, 120, 125]. Its input is a stream of sensor readings from the robot's proximity sensors, interleaved with a sequence of action commands. Throughout this paper, this sequence will be denoted

$$d \quad = \quad \{o^{(1)}, o^{(2)} \ldots, o^{(T)}\} \tag{1}$$

where each $o^{(t)}$ with $t \in \{1, \ldots, T\}$ is either a sensor reading or an action command. The localization module computes, incrementally and in real-time, a probability distribution $P(\xi^{(t)})$ that expresses the robot's belief to be at location $\xi^{(t)}$ at time t where each $\xi^{(t)}$ is a location in the three-dimensional x-y-θ space.

The robot's belief at time t is described by the conditional probability

$$P(\xi^{(t)}) \quad = \quad P(\xi \mid o^{(1)}, o^{(2)} \ldots, o^{(t)}) \tag{2}$$

To compute this probability, three aspects have to be discussed: (1) initialization, (2) sensing, and (3) motion. The latter two, sensing and motion, have opposite effects on the robot's belief. While sensor readings convey information about the robot's position, thereby often decreasing the entropy of $P(\xi^{(t)})$, actions cause a loss of information due to the inaccurate nature of robot motion, and therefore increase the entropy of $P(\xi^{(t)})$.

3.1.1 Initialization

Initially, at time $t = 0$, $P(\xi^{(0)})$ reflects the robot's initial state of knowledge in the absence of any data d. If the robot's initial position is ξ^0 and if it knows exactly where it is, $P(\xi^{(0)})$ is initialized with a Dirac distribution

$$P(\xi^{(0)}) = \begin{cases} 1, & \text{if } \xi = \xi^0 \\ 0, & \text{if } \xi \neq \xi^0 \end{cases} \tag{3}$$

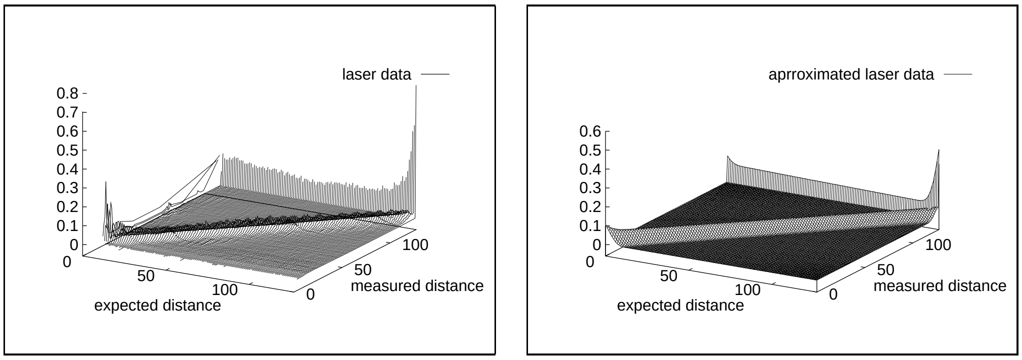

Fig. 4: The conditional probability $P(o \mid dist(\xi))$ obtained from 11,000,000 laser measurements (left) and its approximation using a mixture of a Gaussian, a uniform and a Dirac density (right).

If the robot does not know where it is, $P(\xi)$ is initialized with a uniform distribution. Of particular interest in this paper is the latter case, since the robot was often placed somewhere in the museum without initial knowledge of its position. Thus, the robot had to localize itself under global uncertainty, a problem also known as *global localization* or the *kidnaped robot problem* [37].

3.1.2 Robot Perception

Suppose at time t, the robot receives a sensor measurement $o^{(t)}$. In RHINO's localization module, $o^{(t)}$ is either a laser scan or a sonar scan. This measurement is used to update the internal belief as to where the robot is, according to the following rule:

$$\begin{aligned} & P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t)}) \\ & \quad = \quad \alpha\, P(o^{(t)} \mid \xi^{(t)}, o^{(1)}, \ldots, o^{(t-1)})\; P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)}) \\ & \quad = \quad \alpha\, P(o^{(t)} \mid \xi^{(t)})\; P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)}) \end{aligned} \tag{4}$$

Here α is the Bayes normalizer that ensures that the probabilities on the left-hand side of (4) sum up to 1, and $P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})$ is the robot's belief just before sensing $o^{(t)}$. The first step of the derivation of (4) follows from Bayes rule. The second step rests on the following conditional independence assumption, also called *Markov assumption*:

$$P(o^{(t)} \mid \xi^{(t)}, o^{(1)}, \ldots, o^{(t-1)}) \quad = \quad P(o^{(t)} \mid \xi^{(t)}) \tag{5}$$

This conditional independence assumption states that if the robot's location at time t is known, knowledge of past sensor readings $o^{(1)}, \ldots, o^{(t-1)}$ do not convey any information

(a) laser scan and map (b) probability distribution for different positions

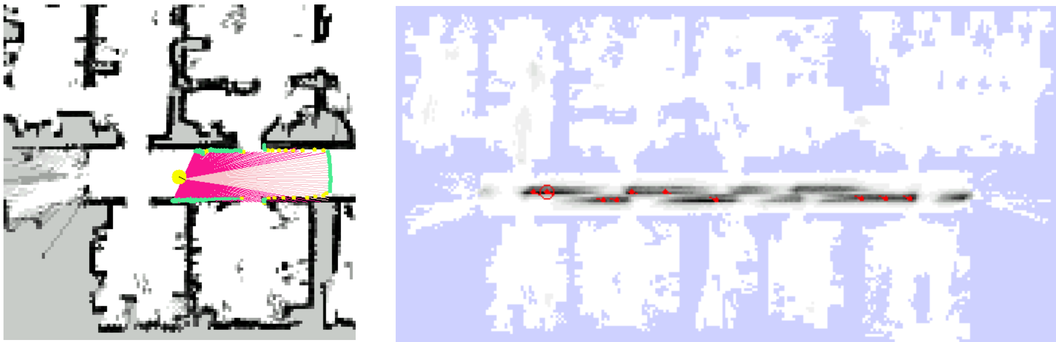

Figure 5: Probabilistic model of perception: (a) Laser range scan, projected into a previously acquired map. (b) The probability $P(o \mid \xi)$, evaluated for all positions ξ and projected into the map (shown in grey). The darker a position, the larger $P(o \mid \xi)$.

relevant to the prediction of $o^{(t)}$. In other words, the Markov property assumes there is no *state* in the environment other than the robot's own location. In most realistic environments such as the museum, this assumption is violated; for example, people often block the robot's sensors for multiple time steps, which makes sensor readings conditionally dependent even if the exact robot location is known. Section 3.1.6 will explicitly address this problem. For now, the Markov assumption will be adopted, as it is mathematically convenient and as it justifies a simple, incremental update rule.

The update equation (4) relies on the probability $P(o^{(t)} \mid \xi^{(t)})$ of observing $o^{(t)}$ at location $\xi^{(t)}$, which henceforth is called the *perceptual model*. The perceptual model does not depend on t; thus, for the reader's convenience we will omit the superscript (t) and write $P(o \mid \xi)$ instead.

RHINO uses its proximity sensors (sonars, lasers) for localization. Its perceptual model is obtained using a generic noise model of the robot's sensors along with a map of the environment. More specifically, $P(o \mid \xi)$ is computed in two steps:

$$P(o \mid \xi) \quad = \quad P(o \mid dist(\xi)) \tag{6}$$

Here the function $dist{:}\Xi \longrightarrow \Re$ computes the *expected* measurement that a noise-free sensor would obtain in a stationary environment. The value of $dist(\xi)$ is computed by ray tracing in a map of the robot's environment. The remaining probability, $P(o \mid dist(\xi))$, models the *noise* in perception. It is learned from data. The left diagram in Figure 4 shows the empirical distribution of $P(o \mid dist(\xi))$ obtained from $11 \cdot 10^6$ measurements; here "expected distance" refers to $dist(\xi)$, "measured distance" refers to o, and the vertical axis plots the probability $P(o \mid dist(\xi))$. In RHINO's software, $P(o \mid dist(\xi))$ is approximated by a mixture of a Gaussian, a uniform, and a Dirac distribution, as shown in the right

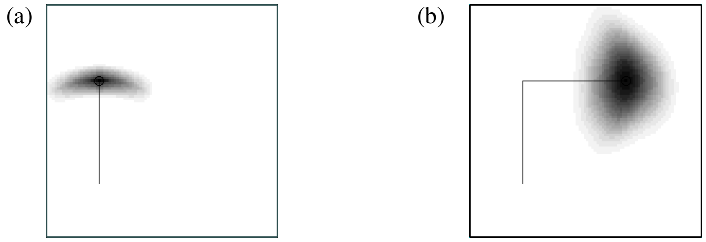

Figure 6: Probabilistic model of robot motion: Accumulated uncertainty after moving (a) 40 meter, (b) 80 meter.

diagram in Figure 4. The coefficients of these distribution are learned from data, using the maximum likelihood estimator [6].

Figure 5 illustrates the perceptual model in practice. An example laser range scan is shown in Figure 5a. Figure 5b shows, for each position ξ, the likelihood $P(o \mid \xi)$ of this specific range scan in a pre-supplied map (projected into 2D). As is easy to be seen, $P(o \mid \xi)$ is high in the main corridor, whereas it is low in the rooms.

3.1.3 Robot Motion

Motion changes the location of the robot. Thus, if $o^{(t)}$ is a motion command, the robot's belief changes according to the following rule:

$$\begin{aligned} & P(\xi^{(t+1)} \mid o^{(1)}, \ldots, o^{(t)}) \\ & \quad = \int P(\xi^{(t+1)} \mid \xi^{(t)}, o^{(1)}, \ldots, o^{(t)})\ P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t)})\ d\xi^{(t)} \\ & \quad = \int P(\xi^{(t+1)} \mid \xi^{(t)}, o^{(t)})\ P(\xi^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})\ d\xi^{(t)} \end{aligned} \tag{7}$$

This update rule is incremental, just like the perceptual update rule (4). The first step in its derivation is obtained using the Theorem of total probability, and the second step is based on a similar Markov assumption as the one above:

$$P(\xi^{(t+1)} \mid \xi^{(t)}, o^{(t)}) \quad = \quad P(\xi^{(t+1)} \mid \xi^{(t)}, o^{(1)}, \ldots, o^{(t)}) \tag{8}$$

In fact, both Markov assumptions described in this section are consequences of a single one, which states that the location of the robot is the only state in the environment.

Equation (7) relies on $P(\xi^{(t+1)} \mid \xi^{(t)}, o^{(t)})$, which is a probabilistic *model of robot motion*. Since the motion model does not depend on t, we will henceforth denote it by $P(\xi \mid \xi', o)$. In our implementation, $P(\xi \mid \xi', o)$ is realized using a mixture of two independent, zero-centered distributions, which model rotational and translational error, respectively [15, 128]. The width of these distributions are proportional to the length of the

1. Initialization: $P(\xi) \longleftarrow Bel_{\text{pri}}(\xi^{(0)})$
2. For each observation o do:

$$P(\xi) \quad \longleftarrow \quad P(o \mid \xi)\; P(\xi) \tag{9}$$

$$P(\xi) \quad \longleftarrow \quad P(\xi)\; \left[\int P(\xi')\; d\xi'\right]^{-1} \quad \text{(normalization)} \tag{10}$$

3. For each action command o do:

$$P(\xi) \quad \longleftarrow \quad \int P(\xi \mid \xi', o)\; P(\xi')\; d\xi' \tag{11}$$

Table 1: Markov localization—the basic algorithm.

motion command. Figure 6 illustrates RHINO's motion model for two example motion commands. Shown there are "banana-shaped" distributions $P(\xi \mid \xi', o)$, which result if the robot starts at ξ' and executes the motion commands specified in the figure caption. Both distributions are of course three-dimensional (in x-y-θ-space); Figure 6 shows their 2D projections into x-y-space.

3.1.4 Grid-based Markov Localization

The generic, incremental Markov localization algorithm is depicted in Table 1. Here the time index is omitted, to emphasize the incremental nature of the algorithm. In experimental tests this method has been demonstrated to localize the robot reliably in static environments even if it does not have any prior knowledge about the robot's position [15, 16, 43].

Recently, different variants of Markov localization have been developed [15, 65, 100, 120]. These methods can be roughly distinguished by the nature of the state space representation. Virtually all published implementations of Markov localization are based on coarse-grained representations of space, often with a spatial resolution of less than one meter and an angular resolution of 90 degrees. For example, in [65, 100, 120] Markov localization is used for landmark-based corridor navigation and the state space is organized according to the topological structure of the environment. Unfortunately, coarse-grained, topological representations are insufficient for navigating in the close vicinity of invisible (but known) obstacles, such as the glass cages described above. Thus, RHINO's localization algorithm differs from previous approaches in that it employs a fine-grained, grid-based decomposition of the state space [15]. In all our experiments reported here, the spatial resolution was 15cm and the angular distribution was $2°$.

The advantage of this approach is that it provides a high accuracy with respect to the position and orientation of the robot. Its disadvantage, however, is the huge state space

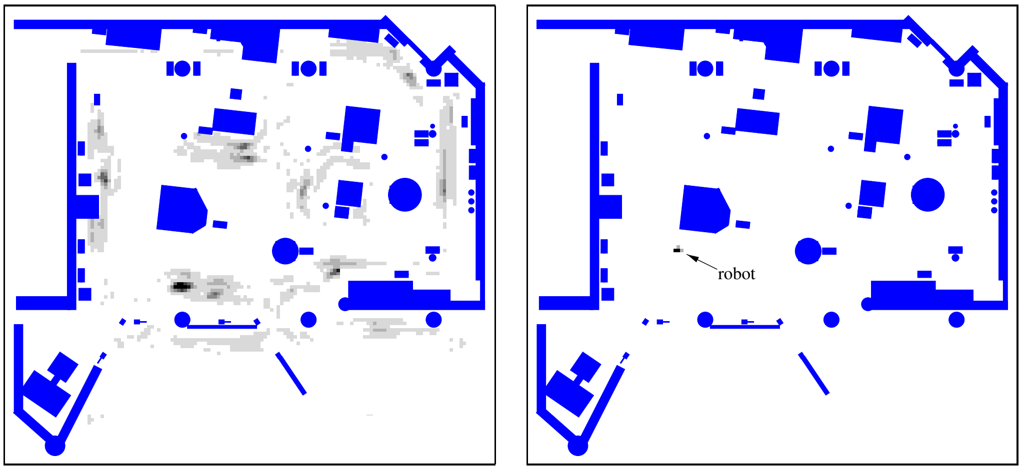

Fig. 7: Global localization in the Deutsches Museum Bonn. The left image shows the belief state after incorporating one laser scan. After incorporating a second scan, the robot uniquely determined its position (right).

which has to be maintained and updated. With such a high resolution, the number of discrete entities is huge, and the basic algorithm cannot be run fast enough on our current low-cost computational hardware for the algorithm to be of practical use.

3.1.5 Selective Computation

To cope with the large numbers of grid cells, RHINO updates them *selectively*. The legitimacy of selectively updating $P(\xi)$—instead of updating all values at all times—is based on the observation that most of the time, the vast majority of grid cells have probability vanishingly close to zero and, thus, can safely be ignored. This is because in most situations, the robot knows its location with high certainty, and only a small number of grid cells close to the true location have probabilities that are significantly different from zero.

In RHINO's localization algorithm, grid cells whose probability are smaller than a threshold θ are not updated. Instead, they are represented by a single value, which uniformly represents the probability of all non-updated grid cells [14]. In the museum exhibit, the threshold θ was set to 0.1% of the a priori position probability. This led to an average savings of two orders of magnitude while not reducing the accuracy of the localization algorithm in any noticeable way.

Figure 7 shows a typical example of global localization in the Deutsches Museum Bonn. RHINO is started with a uniform distribution over its belief state. The probability distribution given after integrating the first sensor scan is shown on the left side of Figure 7. Thus, after incorporating a single sensor scan, the probability mass is readily centered on a much smaller number of grid cells. After incorporating a few more sensor scans, the robot knows its position with high certainty. In the museum exhibit, the localization algorithm was run on a single-processor SUN 170Mhz UltraSparc station, equipped with 256MB RAM. The time required to process a sensor scan varied, depending on the uncertainty in the robot's position. Initially, when the robot was maximally uncertain about its position and therefore had to update every single value in $P(\xi)$, processing a sensor scan required approximately 20 seconds. After the initial localization, the robot's uncertainty was consistently low, which reduced the computational complexity tremendously. The average processing time for processing a sensor scan was approximately 0.5 sec. Since our sensors (sonar and laser) generate approximately 8 scans per second, not every sensor reading was considered in localization. In addition, only a subset of the 360 range readings generated with the laser range finder were considered, since these readings are highly redundant. The practical success of the localization algorithm, however, demonstrates that sufficiently much sensor data was incorporated while the robot was moving.

3.1.6 Entropy Gain Filters: Beyond The Markov Assumption

Unfortunately, the basic Markov localization approach is bound to fail in densely populated environments. Markov localization approaches, by definition, assume that the environment is static—a direct consequence of the underlying Markov assumption. The presence of people violates the Markov assumption by introducing additional state.

In the museum, people often followed the robot closely for extended durations of time. In such situations, the Markov assumption can be fatal. For example, when multiple visitors collaboratively blocked the robot's path, the sensor readings often suggested the presence of a wall in front of the robot. For such readings o, $P(o \mid \xi)$ is maximal for locations ξ next to walls. Since the Markov localization algorithm incorporates $P(o \mid \xi)$ in a multiplicative way every time a sensor reading is taken, multiple iterations of this algorithm will ultimately make the robot believes that it is next to a wall. This property is a direct consequence of the conditional independence assumption (Markov assumption) that was used in the derivation of the Markov localization algorithm,

At first glance, one might attempt to remedy the problem by introducing additional state features in the Markov approach. Instead of estimating the location of the robot as the only state, one could extend Markov localization to simultaneously estimate the locations of the people. With such an enriched state representation, the Markov assumption would be justified and the approach would therefore be applicable. Unfortunately, such an extension is computationally expensive, since the computational and memory complexity increases exponentially in the number of state variables. In addition, such an approach requires probabilistic models of the behavior of the various non-stationary obstacles, such as humans, which are difficult to obtain.

In our approach, we pursued a different line of thought: filtering. The idea is to sort sensor readings into two buckets, one that corresponds to known obstacles such as walls, and one that corresponds to dynamic obstacles such as humans. Only the former readings are incorporated into the position estimation, whereas the latter ones are simply discarded. The filtering approach does not explicitly estimate the full state of the environment; rather, it reduces the damaging effect that arises from state other than the robot's location.

The specific filter used in our implementation is called *entropy gain filter* and works as follows. The *entropy* $H(P)$ of a distribution P is defined by [20]

$$H(P) \quad = \quad -\int P(\xi) \log P(\xi) \; d\xi. \tag{12}$$

Entropy is a measure of uncertainty: The larger the entropy, the higher the robot's uncertainty as to where it is. *Entropy gain* measures the relative change of entropy upon incorporating a sensor reading into P. More specifically, let o denote a sensor scan, and let o_i denote an individual component of the scan (i.e., a single range measurement). The *entropy gain of a probability distribution P with respect to a sensor measurement o_i* is defined as:

$$\Delta H(P \mid o_i) \quad := \quad H(P(\xi^{(t)} \mid o_i^{(t)})) \; - \; H(P(\xi^{(t-1)})) \tag{13}$$

Entropy gain measures the change of certainty. A positive entropy gain indicates that after incorporating o_i, the robot is less certain about its position. A negative entropy gain indicates an increase in certainty upon incorporating o_i.

RHINO's *entropy gain filter* filters out sensor measurements that, if used, would *decrease* the robot's certainty. This is achieved by considering only those o_i for which $\Delta H(P \mid o_i) \leq 0$. The entropy gain filter makes robot perception highly selective, in that only sensor readings are considered that confirm the robot's current belief. The resulting approach does not comply with the original Markov assumption.

Figure 8 shows a prototypical situation which illustrates the entropy gain filter. Shown there are examples where RHINO has been projected into the map at its most likely position. The lines indicate the current proximity measurements, some of which correspond to static obstacles that are part of the map, whereas others are caused by humans (max-range measurements are not shown). The different shading of the measurements demonstrates the result of the entropy gain filter. The black values reduce entropy, whereas the gray values would increase the robot's entropy and are therefore filtered out. Here all measurements of humans are successfully filtered out. These examples are prototypical. In the museum exhibit, we never observed that a reading caused by a dynamic obstacle (such as a human) was not successfully filtered out. We did observe, however, that the robot occasionally filtered our measurements that stemmed from stationary obstacles that were part of the map.

The entropy gain filter proved to be highly effective in identifying non-static obstacles and in filtering sensor readings accordingly. Throughout the complete deployment period, the robot incorporated sufficiently many sensor readings to never lose track of its position. Using the data gathered in the museum, we evaluated the accuracy of our localization algorithm systematically using 118 reference positions, whose coordinates were determined manually [45]. One of the data sets, shown in Figure 9, contains data collected during 4.8

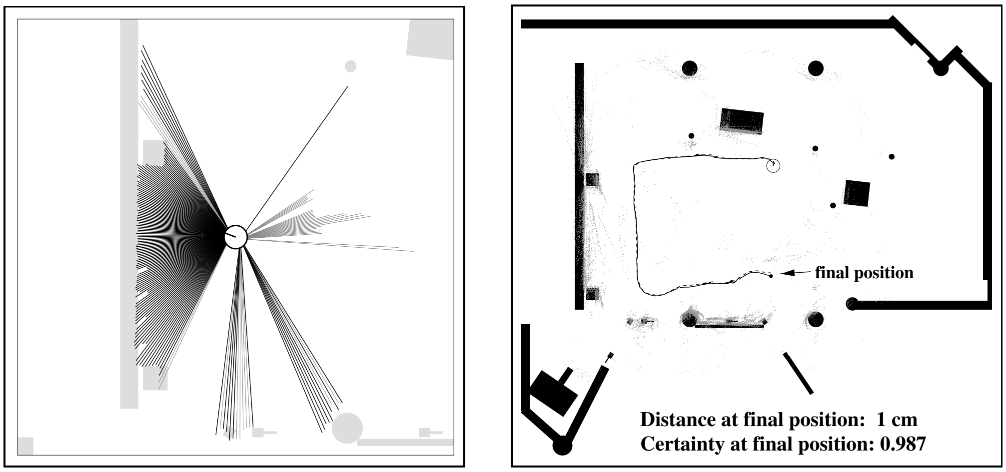

Fig. 8: Sensor measurements (black) selected by the entropy filter from a typical scan (left image) and endpoints of selected scans on a longer trajectory (right image)

hours of robot operation in peak traffic, in which the robot traversed 1,540 meters. In this data set, more than 50% of all sensor readings were corrupted by people for longer periods of time. The average localization error was found to be smaller than 10cm [45]. In only one case did we observe some noticeable error. Here the robot's internal belief deviated approximately 30cm from the real position. As a result, the robot touched a large, invisible metal plate in the center of the museum. The localization error was preceded by a failure of the robot's sonar sensors for an unknown duration of time.

Unfortunately, the basic entropy gain filter also has a disadvantage. When applied as described above, it impairs the robot's ability to recover from large errors in its localization. This is because if the robot's position estimate is wrong, the entropy gain filter might filter out those sensor readings that convey information about its correct position, making a recovery impossible. A successor of the entropy gain filter, which combines the advantages of entropy gain filters while still retaining the ability to globally localize the robot, is described in [45]. As discussed in more depth there, Markov localization combined with the entropy gain filter was able to accurately estimated the position of the robot throughout the entire deployment period, and the entropy filter played a crucial role in its success.

3.1.7 Finding People

As an aside, it is interesting to notice that the entropy gain filter fulfills a secondary purpose in RHINO's software. Sensor measurements o_i with $\Delta H(P \mid o_i) > \gamma$ (with $\gamma > 0$) indicate the presence of an unexpected obstacle, such as people. Thus, the inverse of the entropy gain filter is a filter that can detect people. This filter differs from many other

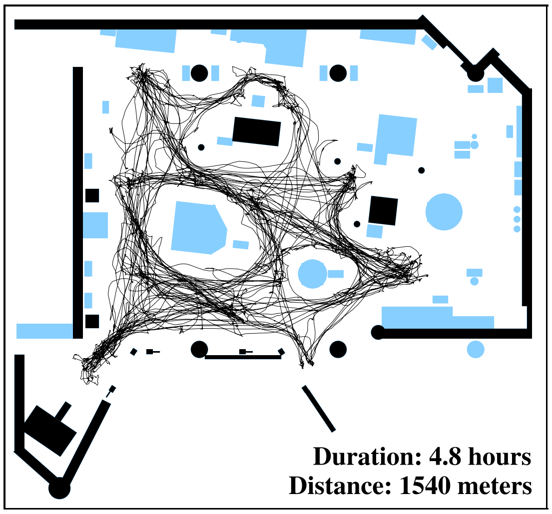

Fig. 9: One of the data sets used to evaluate the accuracy of RHINO's localization in densely populated environments. Here more than half of all sensor readings were corrupted by people; yet the robot managed to keep its average localization error below 10 cm.

approaches in the literature on people detection [63, 67] in that it can find people who do *not* move, and even while the robot itself is in motion. As will be described in more detail below, this filter, in combination with a criterion that measures the robot's progress towards its goal, was used to activate the robot's horn. As a result, the robot blew its horn whenever humans blocked its path; an effect, that most visitors found highly entertaining.

3.2 Mapping

The problem of mapping is the problem of estimating the occupancy of all $\langle x, y \rangle$ locations in the environment [7, 34, 96, 127] from sensor data. Mapping is essential if the environment changes over time, specifically if entire passages can be blocked. In the museum, stools or people often blocked certain regions or passages for extended durations of time. RHINO's ability to acquire maps on-the-fly enabled it to dynamically plan detours, which prevented it from getting stuck in many cases.

The statistical estimation involved in building occupancy maps from sensor data is similar to the probabilistic estimation of the robot's location. Let c_{xy} denote a random variable with events in $\{0, 1\}$ that denotes the occupancy of a location $\langle x, y \rangle$ (in world coordinates). Here 1 stands for *occupied*, and 0 stands for *free*. Then, the problem of mapping is to estimate

$$P(\{c_{xy}\} \mid o^{(1)}, \ldots, o^{(t)}) \tag{14}$$

where the set of all occupancy values $\{c_{xy}\}$ denotes the map. Since the variables to be estimated—the map—is usually extremely high-dimensional (many of our maps contain 10^6 or more grid cells), it is common practice to treat the occupancy estimation problem independently for each location $\langle x, y\rangle$ [96, 127]. This effectively transforms the high-dimensional occupancy estimation problem into a collection of one-dimensional estimation problems

$$\{\ P(c_{xy} \mid o^{(1)}, \ldots, o^{(t)})\ \} \tag{15}$$

which can be tackled very efficiently.

3.2.1 Temporal Integration of Evidence

The temporal integration of sensor evidence is basically analogous to Markov localization. Just like Markov localization, our mapping approach relies on the following Markov assumption

$$P(o^{(t)} \mid c_{xy}, \xi^{(t)}, o^{(t')}) \quad = \quad P(o^{(t)} \mid \xi^{(t)}, c_{xy}) \ \ \text{for } t \neq t' \tag{16}$$

which renders sensor data conditionally independent given the true occupancy of the grid cell $\langle x, y\rangle$. Here $o^{(t)}$ stands for the sensor reading taken at time t. To separate the problem of mapping from that of localization, it is assumed that the robot's location $\xi^{(t)}$ is known[1]; henceforth, $\xi^{(t)}$ will be omitted in the mathematical derivation. In our implementation, the maximum likelihood estimation

$$\hat{\xi}^{(t)} \quad = \quad \underset{\xi}{\operatorname{argmax}}\, P(\xi^{(t)}) \tag{17}$$

is used as the robot's location.

Armed with all necessary assumptions, we are now ready to derive an efficient algorithm for statistical occupancy estimation from data. The probability that a location $\langle x, y\rangle$ is occupied given the data is given by

$$\begin{aligned} P(c_{xy} \mid o^{(1)}, \ldots, o^{(t)}) &= \frac{P(o^{(t)} \mid c_{xy}, o^{(1)}, \ldots, o^{(t-1)})\ P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{P(o^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})} \\ &= \frac{P(o^{(t)} \mid c_{xy})\ P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{P(o^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})} \\ &= \frac{P(c_{xy} \mid o^{(t)})\ P(o^{(t)})\ P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{P(c_{xy})\ P(o^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})} \qquad (19) \end{aligned}$$

[1] See [128] for an approach to concurrent localization and mapping that relaxes these assumption and estimates both the robot's location and the location of the obstacles using a single, mathematical approach.

1. Initialization: $P(c_{xy}) \longleftarrow 0.5$
2. For each observation o do:

$$P(c_{xy}) \quad \longleftarrow \quad 1 - \left(1 + \frac{P(c_{xy} \mid o)}{1 - P(c_{xy} \mid o)} \; \frac{P(c_{xy})}{1 - P(c_{xy})}\right)^{-1} \tag{18}$$

Table 2: Mapping—the basic algorithm.

This transformation is obtained via Bayes rule, followed by applying the Markov assumption and a second application of Bayes rule.

The binary nature of occupancy—a location $\langle x, y\rangle$ is either occupied or not—can be exploited to derive a slightly more compact update equation than in the real-valued case of position estimation [96]. In analogy to (19), the probability that a location $\langle x, y\rangle$ is free (unoccupied) is given by

$$P(\neg c_{xy} \mid o^{(1)}, \ldots, o^{(t)}) \quad = \quad \frac{P(\neg c_{xy} \mid o^{(t)}) \; P(o^{(t)}) \; P(\neg c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{P(\neg c_{xy}) \; P(o^{(t)} \mid o^{(1)}, \ldots, o^{(t-1)})} \tag{20}$$

Dividing (19) by (20) yields the following expression, which is often referred to as the *odds* of $\langle x, y\rangle$ being occupied [103]:

$$\frac{P(c_{xy} \mid o^{(1)}, \ldots, o^{(t)})}{P(\neg c_{xy} \mid o^{(1)}, \ldots, o^{(t)})} \quad = \quad \frac{P(c_{xy} \mid o^{(t)}) \; P(\neg c_{xy}) \; P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{P(\neg c_{xy} \mid o^{(t)}) \; P(c_{xy}) \; P(\neg c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})} \tag{21}$$

it follows that the desired probability is given by

$$\begin{aligned} & P(c_{xy} \mid o^{(1)}, \ldots, o^{(t)}) \\ = \;& 1 - \left(1 + \frac{P(c_{xy} \mid o^{(t)})}{1 - P(c_{xy} \mid o^{(t)})} \; \frac{1 - P(c_{xy})}{P(c_{xy})} \; \frac{P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}{1 - P(c_{xy} \mid o^{(1)}, \ldots, o^{(t-1)})}\right)^{-1} \end{aligned} \tag{22}$$

Here $P(c_{xy})$ represents the *prior distribution* of c_{xy} (in the absence of data), which in out implementation is set to 0.5 and can therefore be ignored.

As is easily seen, the latter estimation equation can be computed incrementally, leading to the mapping algorithm shown in Table 2. The probability $P(c_{xy} \mid o)$ is called the *inverse sensor model* (or *sensor interpretation*), whose description is subject to the next section.

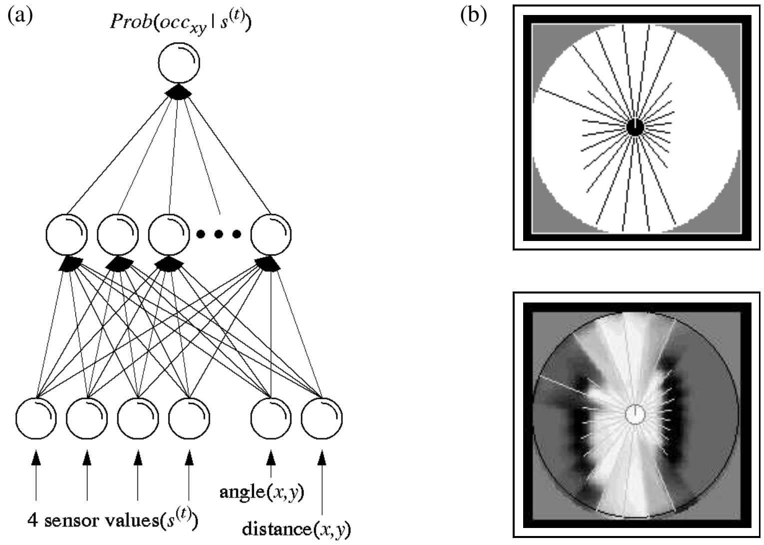

Figure 10: (a) An artificial neural network maps sensor measurements to probabilities of occupancy. (b) An example sonar scan, along with the local map generated by the neural network. The darker a region, the more likely it is to be occupied.

3.2.2 Neural Network Sensor Interpretation

In RHINO's mapping approach, $P(c_{xy} \mid o)$ maps a sensor reading to a conditional probability that location $\langle x, y \rangle$ is occupied (under knowledge of the actual position ξ). In traditional approaches to mobile robot mapping, $P(c_{xy} \mid o)$ is usually crafted by hand, based on knowledge of the characteristics of the respective sensor. In our approach, which is described in more detail in [127], an artificial neural network is trained with Back-Propagation [56, 108] to approximate $P(c_{xy} \mid o)$ from data. This *interpretation network*, which is shown in Figure 10a, accepts as input an encoding of a specific $\langle x, y \rangle$-location, encoded in polar coordinates, relative to the robot's local coordinate system. Part of the input are also the four sensor readings that are geometrically closest to $\langle x, y \rangle$. The output of the network, after training, is an approximation of $P(c_{xy} \mid o)$. Training data for learning $P(c_{xy} \mid o)$ is obtained by placing the robot randomly in a known environment and recording a sensor reading. For each $\langle x, y \rangle$ within the robot's perceptual range (which in our implementation is between 3 and 5 meters), a training pattern is generated, whose label reflects whether or not the $\langle x, y \rangle$ is occupied. After appropriate training [90, 94, 127], the output of the network can be interpreted as the desired conditional probability $P(c_{xy} \mid o)$. Figure 10b shows examples of sonar sensor readings and the corresponding probabilities generated by the trained neural network.

In conjunction with any of the approaches presented in [50, 51, 88, 123, 127, 128, 130], the mapping algorithm is powerful enough to generate consistent maps from scratch. Two example maps are shown in Figure 11. Both maps were constructed in less than 1 hour.

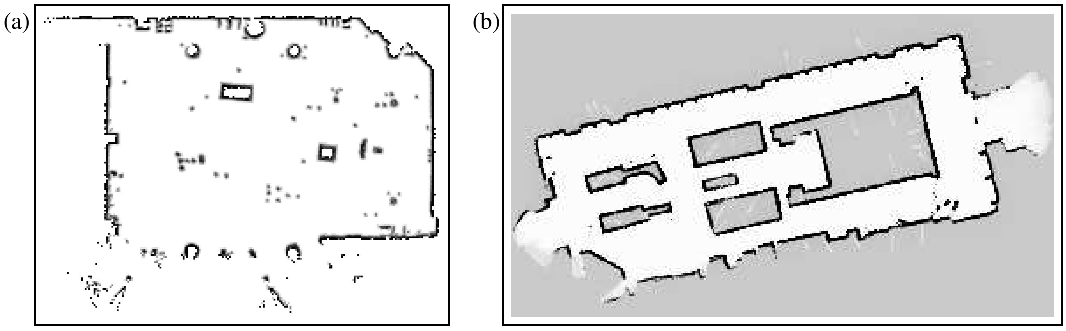

Figure 11: Maps learned from scratch: (a) the Deutsches Museum Bonn, and (b) the Dinosaur Hall of Pittsburgh's Carnegie Museum of Natural History, built in preparation of the installation of a similar tour-guide robot. Both maps were acquired in less than an hour.

The scarceness of the map shown in Figure 11a. however, illustrates the large number of undetectable obstacles (cf. the hand-crafted map shown in Figure 18). Because of this, we chose to provide RHINO with a hand-crafted CAD map instead.

3.2.3 Integration of Multiple Maps

RHINO possesses two major proximity sensor systems, a ring of 24 ultrasonic transducers (sonars) and a pair of laser range finders. Both sensor systems cover a full 360 degree range. Since the perceptual characteristics of both systems are quite different, and since they are mounted at different heights, separate maps are built for each sensor.

From those, and from the hand-supplied CAD map, a single map is compiled using the conservative rule

$$P(c_{xy}^{\rm int}) \quad = \quad \max\left\{P(c_{xy}^{\rm laser}), P(c_{xy}^{\rm sonar}), P(c_{xy}^{\rm CAD})\right\} \tag{23}$$

where the superscript "int" marks the integrated map and the various superscripts on the right hand-side correspond to the respective maps. The integrated map is used for all navigational purposes. The reader may notice that the integration rule (18) is inappropriate for map integration; such a rule could easily dismiss obstacles that are only detectable by one of the sensor systems.

Figure 12 shows an example of the various maps and their integration. Other examples of integrated maps are shown in Figure 13. These examples were recorded during peak traffic hours. In both cases, a massive congestion made it impossible to progress along the original path. The robot's ability to modify its map and hence its paths on-the-fly was absolutely essential for the high reliability with which the robot reached its destinations.

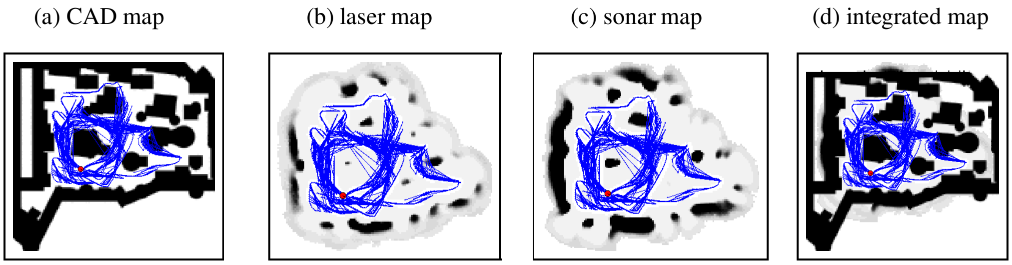

Figure 12: Integrating multiple maps: (a) CAD map, (b) laser map, (c) sonar map, and (d) the integrated map. The scarceness of the sensor-based maps, when compared to the CAD map, indicates how few of the obstacles are actually detectable.

4 Planning and Execution

RHINO motion control is implemented hierarchically, using three different modules for generating control. These are, in increasing levels of abstraction:

1. **Collision avoidance.** This module directly sets the velocity and the motion direction of the robot so as to move in the direction of a target location while avoiding collisions with obstacles. It is the only module that considers the dynamics of the robot.

2. **Motion planner.** The motion planner consults the map to find shortest paths to an exhibit. The path is communicated to the collision avoidance module for execution. Since maps are updated continuously, the motion planner continuously revises its plans.

3. **Task control module.** The task control module coordinates the various robot activities related to motion and interaction. For example, it determines the sequence at which exhibits are visited during a tour, and it also determines the sequence of steps involved in the dialogue with the user.

The hierarchical organization is fairly classical [75]. Each module has its own way to monitor the execution and react accordingly. In the museum, the robot was always in motion—unless, of course, it intentionally stopped to explain an exhibit.

4.1 Collision Avoidance

The task of the collision avoidance module is to determine the actual motion direction and velocity of the robot so as to operate the robot safely while maximizing its progress towards its goal location. The majority of literature on mobile robot collision avoidance suffers from two limitations, both of which are critical in environments like the museum.

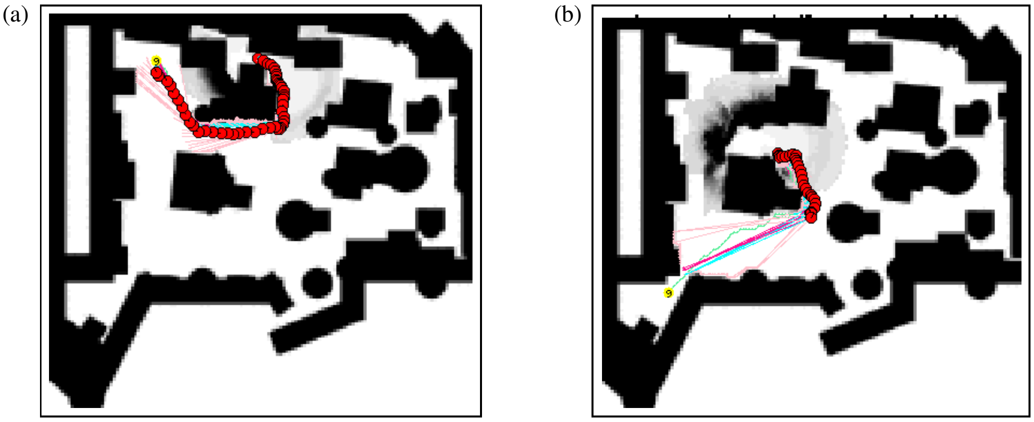

Figure 13: Two integrated maps, acquired in situations where a massive congestion of the museum forced the robot to take a detour.

1. **Inability to handle invisible obstacles.** Virtually all existing methods for collision avoidance are purely sensor-based, i.e., they only consult the robot's sensors to determine collision-free motion [10, 42, 62, 68, 69, 118]. If all obstacles can be sensed, such strategies suffice. However, since some of the obstacles in the museum were invisible, a purely sensor-based approach would have been likely to fail.

2. **Inability to consider dynamics.** With few exceptions [42, 118], existing approaches model only the kinematics of the robot and ignore dynamic effects such as inertia. At lower speeds (such as 20 cm/sec), the dynamics of mobile robots can safely be ignored. At higher speeds (such as 80 cm/sec), however, the robot's inertia can prohibit certain maneuvers, such as sudden stops or sharp turns. Since one of the requirements in the museum was to operate at walking speed, it was essential that the robot's dynamics were taken into account.

RHINO's collision avoidance module, which is called μDWA (short for: model-based dynamic window algorithm), specifically addresses these limitations [44]. μDWA consults a hand-supplied map of the environment to avoid collisions with obstacles that cannot be sensed. The map is also used to bound the area in which the robot operates. To ensure safe motion at high speed, constraints imposed by the robot's dynamics are explicitly considered.

4.1.1 The Dynamic Window Algorithm

The key idea of μDWA is to choose control directly in the *velocity space* of the robot, that is the *translational* and *rotational* velocity. As shown in [42], robots with fixed velocity always travel on a circular trajectory whose diameter is determined by the ratio of translational and rotational velocity. Motor current (torque) change the velocity of the robot and,

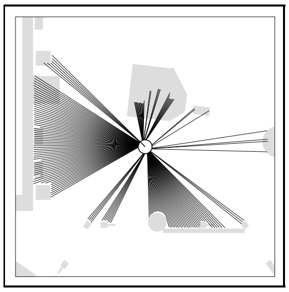

Fig. 14: Scan of the laser sensors, missing much of the large center obstacle.

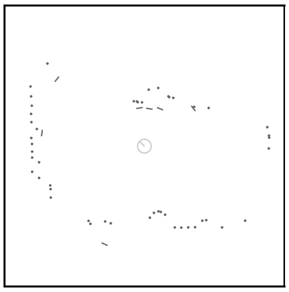

Fig. 15: Obstacle line field, purely sensor-based.

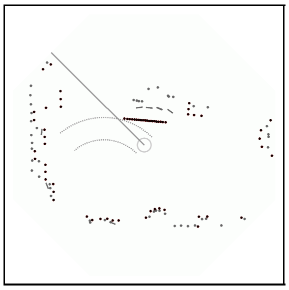

Fig. 16: Obstacle line field enriched using virtual sensors. The circular trajectory visualizes the control, as chosen by μDWA.

as a consequence, its motion direction. The problem of collision avoidance is, thus, the problem of selecting appropriate velocities for translation and rotation.

In regular time intervals (every .25 seconds), μDWA chooses velocities so as to best obey various hard and soft constraints:

1. **Hard constraints** are vital for a robot's safety and are imposed by torque limits. μDWA considers two types of hard constraints: *torque constraints* and *safety constraints*. Torque constraints rule out velocities that physically cannot be attained (e.g., a fast moving robot cannot take a 90 degree turn). Safety constraints rule out velocity settings that would inevitably lead to a collision with an obstacle. Notice that hard constraints do not specify preferences among the different control options; neither do they take into account the robot's goal.

2. **Soft constraints** express *preferences* for both the motion direction and the velocity of the robot. μDWA measures the progress towards the goal by trading off three different soft constraints, which measure (1) translational velocity, (2) heading towards the target position, and (3) forward clearance. If combined in the right ratio, these criteria lead to goal-directed behavior while graciously avoiding collisions.

Consider the situation depicted in Figure 14 in which the robot is nearly in straight motion at a translational speed of about 40cm/sec. Figure 17 depicts the whole velocity space, in which each axis corresponds to a velocity (translational and rotational). The robot's current velocity is in the center of the small rectangular box in the diagram, called the *dynamic window*. This window includes all velocities that can be attained in the next 0.25 seconds under the robot's torque limits. Nearby obstacles carve out regions in the diagram (shown there in white), as those velocities would inevitably lead to a collision. The remaining velocities are then evaluated according to a superposition of the three soft constraints listed

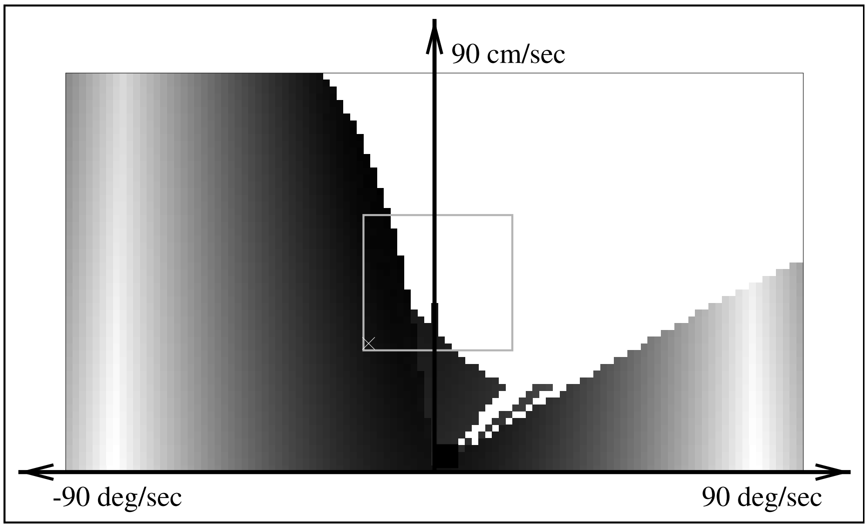

Fig. 17: Each control is a combination of translational (y-axis) and rotational (x-axis) velocity. The darker a control, the higher its value, Also shown here is the *dynamic window* of velocities that can actually be attained. The cross marks the control chosen by μDWA.

above, which favors velocity vectors with high translational velocity and for which the robot's heading direction points towards the goal. The overall evaluation of each velocity pair is represented by its grey level, where darker values correspond to velocity pairs with higher value. The cross marks μDWA's final selection, which makes the robot follow the (circular) trajectory shown in Figure 16.

4.1.2 Integrating Sensor Readings and Maps

μDWA integrates "real" proximity measurements, obtained from the robot's various sensors (tactile, infrared, sonar, laser), with "virtual" proximity measurements, generated using a map of the environment. Figure 18 shows the map that was used in the museum for this purpose. This map marks as dark grey all regions that contain obstacles that cannot be sensed. This map was also used to limit the robot's operational range. By adding appropriate virtual obstacles (shown in light grey) it can be ensured that the robot does not accidentally leave the area where it is supposed to operate.

Just like the real sensors (tactile, infrared, sonar, laser), the virtual sensors in μDWA are assumed to be mounted on a circular array around the robot. The generation of virtual measurements is not straightforward, as the robot never knows exactly where it is; instead, it is given the belief $P(\xi)$ that assigns conditional probabilities to the various locations ξ.

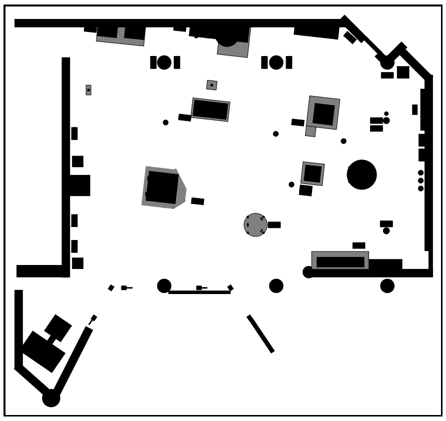

Fig. 18: This map depicts, in dark grey, obstacles which could be detected, and in light grey, the boundary of the robot's operational area.

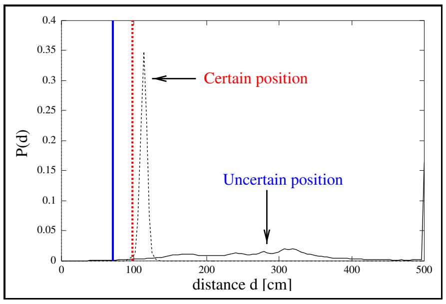

Fig. 19: Probability of measuring a given distance under different degrees of certainty

At first glance, one might want to use the maximum likelihood position

$$\xi^* \quad = \quad \underset{\xi}{\operatorname{argmax}}\, P(\xi) \tag{24}$$

to generate virtual proximity sensor measurements. However, such an approach would be brittle, since it ignores the robot's uncertainty. μDWA uses a more conservative rule which takes uncertainty into account, by generating virtual measurements so that with high likelihood (e.g., 99%), virtual measurements underestimate the actual distance to the nearest object. To explain how this is done, let us first consider situations in which the robot position ξ is known. Recall that $dist(\xi)$ denotes the distance an ideal (noise-free) sensor would measure if the robot's position were ξ, and let X denote a random variable that models the measurement of this ideal sensor. Obviously, the probability $P(X = o \mid \xi)$ is given by a Dirac distribution:

$$P(X = o \mid \xi) \quad = \quad \begin{cases} 1, & \text{if } o = dist(\xi) \\ 0, & \text{if } o \neq dist(\xi) \end{cases} \tag{25}$$

In our case, the robot only has a probabilistic belief $P(\xi)$ as to where it might be. Under this belief, the probability that the sensor returns a value o is given by

$$P(X = o) \quad = \quad \int P(X = o \mid \xi)\ P(\xi)\ d\xi. \tag{26}$$

Consequently, the probability that the sensor measures a value larger than o is given by

$$P(X > o) \quad = \quad \int_{o' > o} P(X = o' \mid \xi)\ P(\xi)\ do' \tag{27}$$

μDWA generates virtual measurements using a conservative rule: The measurement of a virtual sensor is the largest distance that, with 99% probability, underestimates the true distance.

$$o^* \quad = \quad \max\{o : P(X > o) \geq .99\} \tag{28}$$

Let us illustrate this approach using two examples. Figure 7 show two situations, one in which the robot is uncertain about its position, and one in which it is fairly certain. Both situations induce different densities $P(X = o)$, which are shown in Figure 19. The solid curve depicts $P(X = o)$ in the uncertain case, whereas the dashed curve illustrates $P(X = o)$ when the robot is certain. As is easy to be seen, $P(X = o)$ is fairly unspecific in the uncertain case, whereas it is narrow in the certain case. The vertical lines (solid and dashed) indicate the virtual reading that μDWA generates in either situation. With 99% probability, the real distance is larger than the distance suggested by the virtual reading. This conservative rule ensures that the robot does not collide with any of the invisible obstacles, unless it assigns less than 1% probability to its actual position.

Both virtual and real measurements form the basis for determining the robot's motion direction and velocity. Figure 16 shows the integrated sensor information (real and virtual). Figure 16 also shows the trajectory chosen by μDWA, which safely avoids collision with the center obstacle. This figure demonstrates that a purely sensor-based approach is inappropriate.

The collision avoidance module proved to be highly effective in the museum. Because the unified approach to setting speed and motion direction, the approach often maintained walking speed even in cluttered environments. The robot reacted quickly when people blocked its way, which prevented visitors from perceiving the robot as a potential threat. We never observed that parents kept their children—many of whom were much shorter than the robot—from approaching the robot.

4.2 Motion Planning

The collision avoidance module only considers local constraints. As any local motion planning method, cannot cope with U-shaped obstacles. RHINO's motion planning module takes a more global view. Its task is to determine globally shortest paths to arbitrary target points. Paths generated by the motion planner are then communicated to the collision avoidance routine for execution.

The idea for path planning is to let the robot always move on a minimum-cost path to the next exhibit. The cost for traversing a grid cell $\langle x, y\rangle$ is proportional to its occupancy value $P(c_{xy}^{\text{int}})$ (cf. Equation (23)). The minimum-cost path is computed using a modified version of *value iteration*, a popular dynamic programming algorithm [4, 61]:

1. **Initialization.** The grid cell that contains the target location is initialized with 0, all

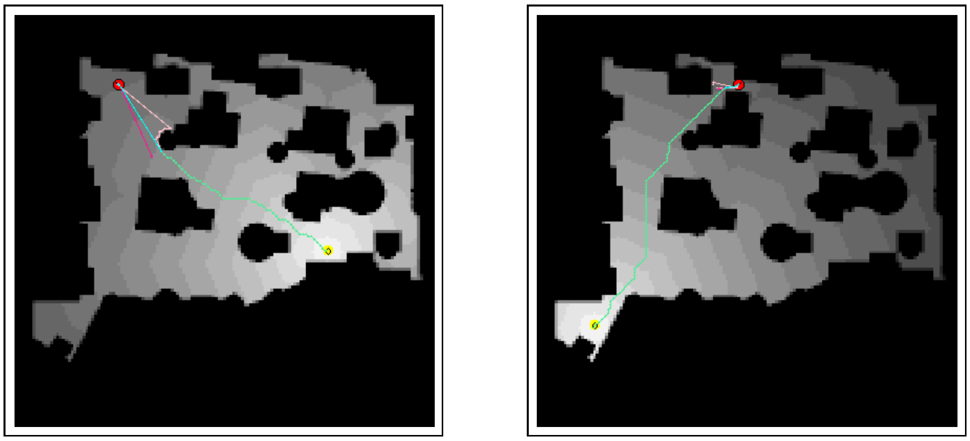

Fig. 20: The motion planner uses dynamic programming to compute the shortest path to the nearest goal(s) for every location in the unoccupied space, as indicated by the gray shading. Once the distance has been computed, paths are generated by hill-climbing in distance space. An additional post-processing step increases the side-clearance to obstacles.

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:

$$V_{x,y} \longleftarrow \min_{\substack{\Delta x=-1,0,1 \\ \Delta y=-1,0,1}} \{V_{x+\Delta x, y+\Delta y} + P(c_{x+\Delta x, y+\Delta y})\}$$

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* [99]). Cost is here equivalent to the probability $P(c_{x,y})$ 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. Determining the motion direction is done in regular time intervals and is fully interleaved with updating V.

Figure 20 shows V after convergence for a typical situation in the museum, using the map shown in Figure 18. The grey-level indicates the cumulative costs V for moving towards the goal point. Notice that every local minimum in the value function corresponds to a goal, if multiple goals exist. 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, *increasing* too small a value can be arbitrarily slow [123, 127]. 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:

$$V_{x,y} \longleftarrow \infty \qquad \text{if} \qquad V_{x,y} < \min_{\substack{\Delta x=-1,0,1 \\ \Delta y=-1,0,1}} \{V_{x+\Delta x, y+\Delta y} + P(c_{x+\Delta x, y+\Delta y})\}$$

   Notice that the remaining $V_{x,y}$-values are not affected. Resetting the value table bears close resemblance to 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 determined 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 [95].

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

- **Any-time algorithm.** Value iteration can be understood as an any-time planner [33], since it allows the generation of a robot action at (almost) any time, long before value iteration has converged. It 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 collision avoidance module adjusts the motion direction commanded by the planner based on sensor readings and dynamic constraints.

In the museum, the motion planner was fast enough for real-time operation. In grid maps of size 30 by 30 meter and with a spatial resolution of 15cm, optimized value iteration, done from scratch, requires approximately one to five 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 much less than a second. Figure 13 shows situations in which a passage is temporarily blocked, along with partially executed plans generated by the motion planner. Such situations occurred frequently in the museum, and without the ability to dynamically change the map and generate new plans, the robot would have frequently been stuck for extended durations of time. The motion planner, together with the collision avoidance and the various state estimation modules described above, provided the robot with the ability to safely move from one exhibit to another, while adjusting the velocity to the circumstances, shipping around obstacles when possible, but choosing completely different trajectories if passages were blocked.

4.3 High-Level Task Control

The task control module coordinates the various robot activities related to motion and user interaction at the highest level. It transforms abstract, user-level commands (such as: "give tour number three") into a sequence of appropriate actions, where actions either correspond to motion commands (e.g., "move to exhibit number five") or control the robot's user interface (e.g., "display image four" and "play pre-recorded message number seventeen"). The task control module also monitors the execution and modifies task-level plans if necessary.

In the museum exhibit, the primary role of the task control module was to determine the order at which exhibits were visited, and to control the user interaction to ensure the robot functioned in accordance to the user's demands. When tours were given to real visitors, the job of the task control module was to monitor and control the dialogue with the visitor, and to monitor plan execution. Internet users were able to compose tours by selecting individual tour items. Since multiple Internet users often sent commands at the same time, there was a combinatorial problem of sequencing exhibits appropriately.

RHINO's task control monitor is an augmented version of GOLOG, which has been described in depth elsewhere [85]. GOLOG is a first-order logical language that represents knowledge in the situation calculus [86]. It uses a built-in theorem prover to generate plans and to verify their correctness [93]. Programs (and plans) in GOLOG are sequences of elemental actions expressed in a logical language using *if-then-else rules* and *recursive procedures*. GOLOG also requires the programmer to provide an abstract model of the robot's environment (a *domain theory*), describing the effects of its actions. The key benefit

proc *internet-tourguide*
  while ($\exists$ *exhibit*) *request(exhibit)* $\wedge$ *next(exhibit)* $\wedge$ $\neg$*visited(exhibit)* **do**
    (π *exhibit*)*.goto(exhibit)* ; *explain(exhibit)*
  endWhile
  goto(parking-position)
endProc

proc *goto(loc)*
  if *robotLocation(robotloc)* $\wedge$ *robotloc* $\neq$ *loc* **then** *drive(loc)* **endIf**
endProc

Table 3: Fraction of the GOLOG program for the Internet tour-guide robot

of GOLOG is that it facilitates designing high-level controllers by seamlessly integrating programming and problem solving [85, 86]. Table 3 depicts an example GOLOG program for scheduling requests by Internet users. It basically specifies that the robot shall serve all pending requests by moving to the corresponding position and explaining it, and return to its homing position thereafter.

Unfortunately, GOLOG, in its current form, suffers from several limitations such as the lack of sensing, interaction, and execution monitoring.[2] Also, there is a mismatch between the level of abstraction of GOLOG actions and those the robot is able to perform, thus making it difficult to directly control the low-level software from GOLOG:

Sensing and Interaction: GOLOG is unable to accept and react to exogenous events. It cannot handle plans conditioned on events not known in the beginning of program execution. In the museum, the robot's actions are, of course, conditioned on user input and various other circumstances, and the ability to react to exogenous events is essential.

Execution monitoring: By default, GOLOG assumes that actions always succeed if their preconditions are met. It does not monitor the execution of its actions. In practice, however, actions can fail, and it is important that the robot reacts adequately. For example, the action `wait_for_user_request()` often does not result in a user response, and timeout mechanisms have to be employed to avoid getting stuck indefinitely.

Level of Abstraction: The primitive actions provided by RHINO's low-level software components are too fine-grained to be used directly in GOLOG. For example, the action *goto(exhibit)* involves a collection of low-level control directives, such as setting track-points for the cameras, setting target locations for the motion planner, turning

[2] In recent work, which was not available when RHINO's software was developed, extensions of GOLOG were proposed which address these shortcomings [31, 32, 81].

the robot towards the exhibit after arrival, etc. While in principle, GOLOG can cope with any granularity, dealing with low-level action sequencing at the most abstract level would make GOLOG programs cumbersome and difficult to understand.

These difficulties are overcome by an intermediate software layer, called GOLEX [54], a runtime and execution monitoring system for GOLOG, which extends GOLOG in three aspects:

- GOLEX integrates sensing and user interaction capabilities into GOLOG. It enables the programmer to formulate conditional programs. Action sequences can be conditioned upon exogenous events (such as the arrival of a tour item request) and timer events. If necessary, GOLEX can activate GOLOG's planner to generate new plans in reaction to unexpected events.
- GOLEX permanently monitors the execution of the various concurrent actions of the underlying robot control system. If it detects a failure, it chooses appropriate actions for recovery and updates the internal state accordingly, so that GOLOG can resume its operation.
- GOLEX decomposes the primitive actions specified in GOLOG into a macro-like sequence of appropriate directives for the robot control system, thereby bridging the gap between GOLOG's abstract task-level programs, and the remaining robot control software.

Table 4 shows an excerpt of RHINO's GOLEX program. This program segment implements, in a Prolog-like notation, the re-scheduling of new tour items by calling GOLOG to compute a new plan, and the various commands involved when moving from one item to another. This program illustrates how high-level actions, such as *drive(location)*, are decomposed into sequences of lower level actions. In particular, the *drive(location)* action involves setting the appropriate tracking point for the cameras, blowing its horn, playing music thereafter, initiating the motion by informing the motion planner, turning the robot towards the exhibit upon arrival, and continuing with the next action in the schedule. The program segment also illustrates how GOLEX can react to exogenous events (in this case: requests for tour items) and change the execution accordingly.

As an example, consider a situation where RHINO is waiting in its parking position and receives a tour request for the exhibits 1 and 12. GOLOG then generates the plan

$$do(drive(p), do(explain(e12), do(drive(e12), do(explain(e1), do(drive(e1), s0))))).$$

which is graphically illustrated by Figure 22. Now suppose RHINO receives a new request for exhibit 5 while it explains exhibit 1. In this case GOLEX uses the predicate `updateSchedule` to initiate replanning using GOLOG's planner. Since exhibit 5 is closer than exhibit 12, the plan is revised to

$$do(drive(p), do(explain(e12), do(drive(e12), do(explain(e5), do(drive(e5), \ldots))))).$$

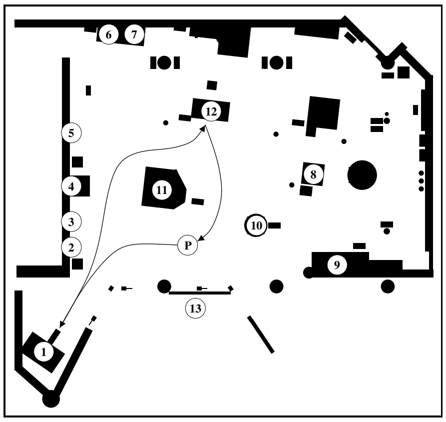

Fig. 21: Schedule of tour items for the exhibits 1 and 12.

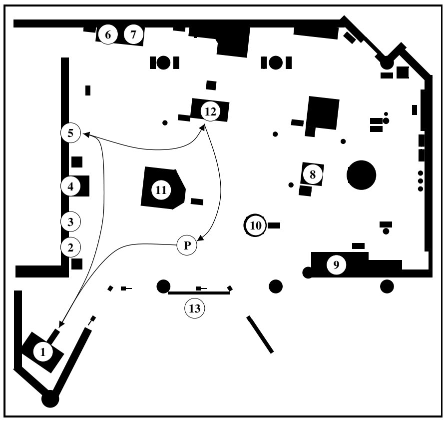

Fig. 22: Re-scheduled plan after receiving a request for a exhibit 5.

The resulting plan for the tour-guide is illustrated in figure 22. By providing these methods for executing primitive actions, reacting to user requests, and monitoring the progress of the robot, GOLEX provides the necessary "glue" between GOLOG and the rest of the robot software.

5 Human-Robot Interaction

An important aspect of the tour-guide robot is its interactive component. User interfaces are of great importance for robots that are to interact with "normal" people. In settings such as the museum, where people typically do not spend extensive amounts of time with the robot, two criteria are most important: ease of use, and interestingness. The user interfaces must be intuitive, so that untrained and non-technical users can operate the system without instruction. Interestingness is an important factor in capturing people's attention.

RHINO possesses two user interfaces, one on-board interface to interact with people directly, and one on the Web. The on-board interface is a mixed-media interface that integrates graphics, sound, and motion. The Web-based interface uses graphics and text. The interactive component was critical for RHINO's user acceptance in the museum. Visitors of the museum paid considerably little attention to the fact that the robot navigated safely from exhibit to exhibit. Instead, many seemed to be most intrigued when the robot interacted with them in some noticeable way. Some of RHINO's most enthusiastic users were less than six years old; others were over 80. The vast majority of users had not been exposed to robots prior to visiting the museum. Since the majority of visitors stayed less than 15 minutes, it was critical that RHINO's interface was easy-to-use and robust.

```
exec( [], Done, Schedule ).

exec([explain(Exhibit)|ToDo], Done, Schedule ) :-
   explain( Exhibit ),
   updateSchedule( [explain(Exhibit)|Done], Schedule,
                   NewToDo, NewSchedule ),
   exec( NewToDo, [explain(Exhibit)|Done], NewSchedule ).

exec([drive(L)|ToDo], Done, Schedule ) :-
   position(L, (X, Y)),
   panTiltSetTrackPoint((X, Y)),
   soundPlay(horn),
   soundPlay(jamesBondTheme)
   robotDrivePath([(X, Y)]),
   robotTurnToPoint((X, Y)),
   exec(ToDo, [drive(L)|Done], Schedule).
```

Table 4: Implementation of primitive actions in GOLEX.

5.1 Mixed-Media User Interface

RHINO's on-board control interface integrates graphics, motion, and spoken, pre-recorded language and sound.

1. Initially, visitors can select a tour or, alternative, listen to a brief, pre-recorded explanation of the system (the "help text"). They indicate their choice by pressing one out of four colored buttons, shown in Figure 23.

2. When RHINO moves towards an exhibit, it displays an announcement on its screen. It also uses its camera-head to indicate the direction of its destination, by continually pointing the camera towards the next exhibit. While in motion, the robot plays music in order to entertain the people.

3. At each exhibit, the robot first plays a brief pre-recorded verbal explanation. Users are then given the choice to listen to more text or to move on to the next exhibit. Users are informed about their choice through the graphical display, and they can indicate their selection by pressing one of two lit buttons next to the screen. If no user presses a button within five seconds, the robot defaults to the fast version of the tour where a minimum of verbal explanations are provided.

4. When a tour is finished, the robot returns to a pre-defined starting position in the entrance area where it waits for new visitors.

Figure 23 illustrates the interaction between visitors in the museum and the robot. The left images shows an example screen of the graphical display. All text was in German. The

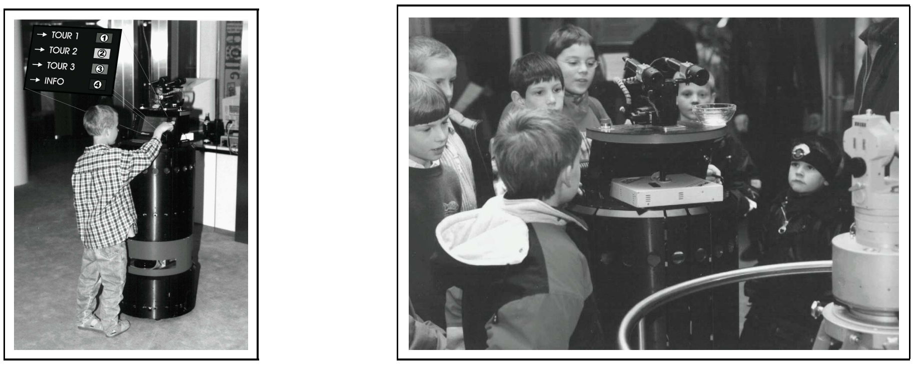

Fig. 23: On-board interface of the tour-guide robot. The users were required to press the buttons right form the display to choose between different options (left image). During the tour the robot used its camera to point to the exhibits. It showed text and graphics on its display and played pre-recorded sounds from CD to explain the exhibits (right image).

---

four colored buttons adjacent to the screen were the sole input interface. In the museum, people consistently grasped the main aspects of the interface within seconds. They were able to select a tour without instruction. Often, they did not realize immediately that they were given the choice to obtain more detailed explanations, once a tour was started. After visiting the third exhibit or so, users were usually aware of this option and made use of it.

5.2 Web Interface

RHINO's Web interface consists of a collection of Web pages[3], which serves four main purposes.

1. Monitoring.
2. Control.
3. Providing background information.
4. Providing a discussion forum.

The interface enabled remote users to control the robot and to observe it, along with the museum and the people therein.

Three of the most frequently visited pages of the Web interface are shown in Figure 24. The page on the left enables users to observe the robot's operation on-line. The left image includes camera images obtained with one of RHINO's cameras (left) and taken with a

[3] See http://www.cs.uni-bonn.de/˜rhino/tourguide/

Fig. 24: Popular pages of the Web interface. The left image shows a page with on-line images from the museum, including background information about the exhibits and further hyper links. The middle image contains a screen dump of a page for smooth animations of the robot's trajectory and actions based on a Java applet. The right image shows the control page, where Web-users can specify where they want the robot to go.

fixed, wall-mounted camera (right). The center of this page shows a map of the robot's environment from a bird's eye perspective, along with the actual location of the robot and the exhibits. The bottom portion of this page has two functions. When the robot is moving, it indicates which exhibit RHINO is moving towards. Once an exhibit is reached, information is provided about this specific exhibit, including hyper-links to more detailed background information. All information on this page is updated synchronously in approximately five second intervals, or when the robot reaches or leaves an exhibit. Each user's Web browser automatically reloads this page in periodic time intervals. The update rate can be specified by the user in accordance to the speed of his communication link (the default is 15 seconds).

To provide more frequent updates of the robot's state we additionally provided a Java page illustrating the current position of the robot and explaining the robot's current action (see middle image of Figure 24). This Java applet directly connects to a dedicated server and updates the position of the robot every 0.3 seconds, thereby providing smooth animations of the robot's motion in the map. At the bottom of the map this applet also scrolls text explaining the current robot mode (e.g., "approaching exhibit 5").

The middle image of Figure 24 serves as the remote control interface of the robot. To send the robot to an exhibit, users can click on exhibits directly in the map or, alternatively, highlight one or more exhibits in the list on the left side of this Web page. For four of the exhibits (9—12) the users could additionally specify the heading from which they wanted to see the images. Up to 12 different viewpoints were admissible for these exhibits. A particular viewpoint could be chosen by clicking on the appropriate region closely around the exhibit. When hitting the "select" button, the selected exhibits are queued. Users can

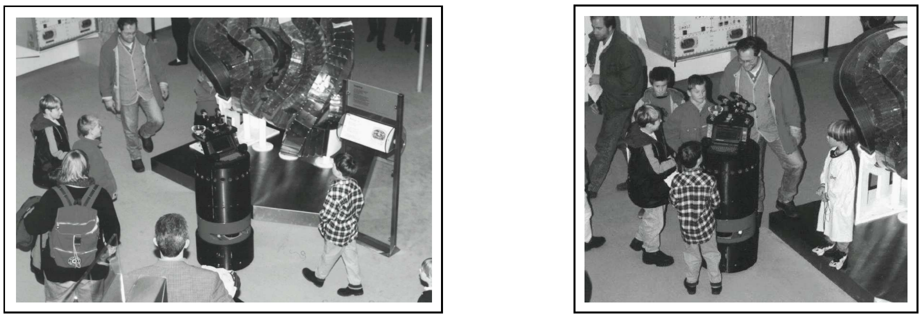

Fig. 25: Typical situation in which visitors try to challenge the robot, here by intentionally blocking its path.

also identify themselves by typing their name (or acronyms) into the name box at the top of this page. This name is then displayed in the queue. The task planner schedules exhibits so as to minimize the travel distance. Requests of multiple users for the same exhibits are accommodated in parallel, so that the exhibit is visited only once. Thus, the length of the queue is limited by the number of different exhibits (13 exhibits plus 36 view points in the museum). At times, there were over 100 pending requests of different individual users. Particularly popular was the tour item "the crew," which was only available to Web users. Upon sending the robot to the crew, the robot positioned itself so that it could see the off-board computers and, at rare occasions, some of us.

When the robot was controlled through the Internet, we quickly learned that the robot was too fast to convince some Web-users that there was a physical machine. This was specifically the case during a so-called "Internet night," a scheduled event that took place outside the opening hours of the museum. Because the museum was empty, most of the time the robot traveled close to its maximum speed of 80cm/sec. With an update rate of 15 seconds per update, Web users saw mostly images of exhibits (where the robot waited for 30 seconds), but they rarely saw the robot traveling from one exhibit to another. This problem was remedied by lowering the maximum speed of the robot to 30 cm/sec. Now Web users saw images recorded along the way, raising their confidence that there might actually be a real machine (as if we couldn't have pre-recorded those images as well).

5.3 Reaction to People

RHINO's ability to react directly to people was among the most fascinating aspects of the entire system. If people stepped in the robot's way, RHINO reacted in multiple ways, each of which was characterized by a different time constant:

1. The first behavior of the robot is initiated by its collision avoidance, slowing the robot down so as to avoid a collision. If the obstacle is not contained in the map (and thus

most likely a person), the robot blows its horn. Such obstacles are detected using the inverse entropy gain filter, as described in Section 3.1.7.

2. The next visible behavior is an attempt to find a local detour around the person. This behavior is also driven by the collision avoidance method. At the same time, the map is modified.

3. If the blockage persists and no local detour exists, the modification of the map leads, after a few seconds, to a global detour. The robot then turns away and chooses the second best global path.

In the museum, people were all but cooperative. Many shared an ambition to "break the system." Attempts to do so included intentionally blocking the robot's way for extensive periods of time, by trying to "push" it outside its operational boundary (e.g., close to the hazardous staircase), or by lining up in a way that looked like a wall to the robot, in order to confuse its sense of location. Typical examples of such a situations are show in Figure 25. Luckily none of these attempts succeeded. We attribute this robustness to the extensive use of methods that are non-reactive, in the sense that they did not base their decisions on the most sensor readings only, as advocated elsewhere [11, 27].

RHINO's ability to react to people proved to be one of the most entertaining aspects, which contributed enormously to its popularity. Many visitors were amazed by the fact that the robot acknowledged their presence by blowing its horn, and repeatedly stepped in its way to get the acoustic "reward." The ability of RHINO to decelerate in the presence of people and to "ask" for clearance proved to be one of the most entertaining aspects of the entire systems.

6 Statistics

The results of the six day deployment are summarized in Table 5 [13]. RHINO operated for approximately 47 hours without any significant downtime. Over this period of time, the robot traveled approximately 18.6km. More than 2,000 real visitors and over 2,000 "virtual" Web-based visitors were guided by RHINO. We counted over 200,000 accesses to RHINO's Web site. The robot served a total of 2,400 tour requests by real and virtual visitors of the museum. Only six requests were not fulfilled, mostly due to scheduled battery changes at the time of the request. Thus, RHINO's overall success-rate was 99.75%. Whenever possible, RHINO chose its maximum speed (80 cm/sec when guiding real people, between 30 and 50 cm/sec when controlled through the Web). The discrepancy between the top and the average speed (36.6cm/sec), however, was due to the fact that in the presence of obstacles, the collision avoidance module was forced to slow the robot down.

During its 47 hours of operation, RHINO suffered a total of six collisions with obstacles, all of which occurred at low speed and did not cause any damage. Only one of these collisions was caused by a software failure. Here the localization module failed to compute

hours of operation	47
number of visitors	>2,000
number of Web visitors	2,060
total distance	18.6 km
maximum speed	>80 cm/sec
average speed during motion	36.6 cm/sec
number of collisions	6
exhibits explained	2,400
success rate	99.75%
increased attendance	>50%

Table 5: Summary of the robot's six-day deployment period.

the robot's position with the necessary accuracy. All other collisions were results of various hardware failures (which were usually caused by neglect on our side to exchange the batteries in time) and by omissions in the manually generated map (which were fixed after the problem was observed).

Overall, RHINO was received with enthusiasm in all age groups. We estimate that more than 90% of the museum's visitors followed the robot for at least a fraction of a tour. Kids often followed the robot for more than an hour. According to the director of the museum, RHINO raised the overall attendance by at least 50%.

7 Related Work

7.1 Localization

Mobile robot localization has frequently been recognized as a key problem in robotics with significant practical importance. Cox [30] noted that "Using sensory information to locate the robot in its environment is the most fundamental problem to providing a mobile robot with autonomous capabilities." A recent book by Borenstein, Everett, and Feng [8] provides an excellent overview of the state-of-the-art in localization. Localization plays a key role in various successful mobile robot architectures [28, 46, 58, 72, 83, 84, 98, 104, 107, 110, 120, 119, 134] and various chapters in [75]. While some localization approaches, such as those described in [60, 78, 120] localize the robot relative to some landmarks in a topological map, RHINO's approach localizes the robot in a metric space, just like those methods proposed in [5, 125, 128].

The vast majority of approaches is incapable of localizing a robot globally; instead, they are designed to track the robot's position by compensating small odometric errors. Thus, they differ from the approach described here in that they require knowledge of the robot's initial position; and they are not able to recover from global localizing failures. Probably the most popular method for tracking a robot's position is Kalman filtering [51, 52, 88, 92, 110, 121], which represent uncertainty by single-modal distributions. These approaches are unable to localize robots under global uncertainty—a problem which Engelson called the "kidnaped robot problem" [37]. Recently, several researchers proposed *Markov localization*, which enables robots to localize themselves under global uncertainty [15, 65, 100, 120]. Global approaches have two important advantages over local ones: First, the initial location of the robot does not have to be specified and, second, they provide an additional level of robustness, due to their ability to recover from localization failures. Unfortunately, none of these methods were appropriate for the museum. This is because previous Markov localization methods relied on extremely coarse-grained, *topological* representations of the environment, making it impossible to navigate close to "invisible" obstacles. By using a fine-grained, metric representation of space, our approach can localize a robot with much higher accuracy, and it can also deal with a wider variety of environments, including those that do not possess obvious geometric features such as corridors, intersections and doors.

In addition, the vast majority of approaches differ from the method described here in that they can only cope with *static* environments, that is, environments that do not possess measurable state other than the robot's location. These approaches are typically brittle in dynamic environments. The approach described in [70] uses cameras pointed towards the ceiling and thus cannot perceive most of the changes that occur in typical office environments. Unfortunately, such an approach is only applicable if the ceiling contains enough structure for accurate position estimation. RHINO's approach, by filtering out sensor data, has been demonstrated to function even in highly dynamic environments. The results obtained in the museum illustrate that it is capable of reliably localizing a robot even if more than 50% of all sensor readings are corrupted by people (see also [45]).

Finally, most existing approaches are restricted in the type features that they consider. Many approaches reviewed in [8], a recent book on this topic, are limited in that they require modifications of the environment. Some require artificial landmarks such as bar-code reflectors [38], reflecting tape, ultrasonic beacons, or visual patterns that are easy to recognize, such as black rectangles with white dots [7]. Of course, modifying the environment is not an option in many application domains. Some of the more advanced approaches use more natural landmarks that do not require modifications of the environment. For example, the approaches of Kortenkamp and Weymouth [78] and Matarić [91] use gateways, doors, walls, and other vertical objects to determine the robot's position. The Helpmate robot uses ceiling lights to position itself [70]. Dark/bright regions and vertical edges are used in [26, 135], and hallways, openings and doors are used by the approach described in [72, 116, 119]. Others have proposed methods for learning what feature to extract, through a training phase in which the robot it told its location [49, 101, 125, 126]. These are just a few representative examples of many different features used for localization.

RHINO's approach differs from all these approaches in that it does not extract predefined features from the sensor values. Instead, it directly processes raw sensor data. Such an approach has two key advantages: First, it is more universally applicable since fewer assumptions are made on the nature of the environment; and second, it can utilize all sensor information, typically yielding more accurate results. Other approaches that process raw sensor data can be found in [51, 52, 88].

7.2 Mapping

RHINO's mapping approach is a variant of the occupancy grid method. Occupancy grids have originally been proposed by Elfes and Moravec [34, 35, 96] and since been adopted in numerous robotic systems (e.g., [10, 12, 53, 112, 136, 137]). To date, occupancy grids have become the most successful metric approach to mobile robot map acquisition. Our approach differs from previous ones in that neural networks are used to learn the mapping from sensors to occupancy values; as a result, sensor readings are interpreted in the context of their neighbors, which increases the accuracy of the resulting maps [123, 127].

Occupancy grids, however, are not the only approach to mobile robot mapping. Chatila and Laumond [21] proposed to represent objects by polyhedra in a global coordinate frame. Cox [29] proposed to construct probabilistic trees to represent different, alternative models of the environment. In his work, Kalman filters and Bayesian methods are used for handling uncertainty. Lu, Milios and Gutmann [50, 88, 89] presented an approach that memorizes raw proximity sensor data in a metric coordinate system, using an alignment procedure that extracts lines from laser range finer data. Jeeves [124], an award-winning robot at the 1996 AAAI mobile robot competition [77], constructs geometric maps incrementally by concatenating wall segments detected in temporal sequences of sonar measurements. Jeeves's design was strongly inspired by the work presented here; its inability to handle dynamic environments and its strong commitment to parallel/orthogonal walls make its software approach more brittle than the approach described here.

A statistical approach, which addresses both mobile robot localization and metric mapping, has been proposed in [128, 129, 130]. This approach uses efficient statistical estimators to interleave localization and mapping. It is specifically tailored towards building maps of large, indoor environments, where the natural drift of the robot makes it difficult to maintain an accurate sense of a robot's position. In the current application, this is not an issue, as an initial map of the environment is readily available.

All approaches discussed thus far fall into the *metric* paradigm [127]. There exists a second, major paradigm to mapping, called *topological* methods. This family of algorithms represents environments by graphs, where nodes correspond to places and arcs correspond to actions for moving from one place to another. Often, topological graphs are enriched by local metric information to facilitate the navigation from one place to another. Among the earliest successful work in this field is an approach by Kuipers and Byun [79, 80]. In their approach, topological places are defined as points that maximize the number of equidistant obstacles (a similar idea can be found in Choset's work, who refers to such points as "meetpoints" [22, 23, 24]). Topological places are connected by arcs, which contain metric information for locally moving from one place to another. The approach disambiguates different places by local sensor information (taken at a single node or, if necessary, at a small number of neighboring nodes). In systematic simulations, this approach has been found to reliably learn large maps of indoor environments, even if sensor data is noisy. However, in these experiments the robot was equipped with a compass, which simplifies the localization problem significantly.

A similar approach was proposed by Matarić [91]. Her algorithm acquires topological maps of the environment in which nodes correspond to pre-defined landmarks such as straight wall segments. Neighboring topological entities are connected by links. The topological representation is enriched by distance information to help keeping track of the location of the robot. The approach was evaluated on a physical robot and was found to be robust in practice. Its inability to maintain an exact position estimate imposes intrinsic scaling limitations. Moreover, since the recognition of landmarks in this approach involve robot motion, the approach might have severe difficulties in recognizing previously visited locations when approaching them from different directions (e.g., T-junctions).

Shatkay and Kaelbling proposed a method that learns topological map from landmark observations [115]. Their work extends work by Koenig and Simmons [72], who investigated the problem of learning topological maps if a topological sketch of the environment is readily available. In Shatkay and Kaelbling's work, no such assumption is made. Their approach considers local topological information along with landmark label information (which is assumed to be observable), to disambiguate different locations. A key feature of their approach is the use of a recursive estimation routine (the Baum-Welsh algorithm [105]) that enables the topological mapper to refine position estimates backwards in time. As a result, their approach has built fairly large topological maps. Their work also predates the work reported in [128, 130], which also employs the Baum-Welsh algorithm to interleave mapping and localization. Other topological approaches can be found in [25, 97, 131, 138].

7.3 Collision Avoidance

In the field of collision avoidance for mobile robots, potential field methods [69] are highly popular. They determine the steering direction of the robot by (hypothetically) assuming that obstacles assert negative forces on the robot and that the target location asserts a positive force. By restricting the field of view to the close vicinity of the robot, these methods are computationally highly efficient. While the physical analogy of considering the robot as a free-flying object is attractive, Borenstein and Koren [74] identified that in practice, such methods often fail to find trajectories between narrowly spaced obstacles; they also can produce oscillatory behavior in narrow corridors. An extended version of the potential field approach is introduced in [68]. By modifying the potential function the motion of the robot becomes more efficient and different behaviors such as wall following and tracking can be achieved.

The *virtual force field histogram* algorithm [9], which closely resembles potential field methods, uses occupancy grids to represent information about obstacles close to the robot. This grid is generated and updated continuously using ultrasonic proximity sensors. Borenstein later extended this approach to the *vector field histogram* [10]. In this algorithm. occupancy information is transformed into a histogram description of the free space around the robot, which is used to compute the motion direction and velocity for the robot.

All of the approaches discussed above generate motion commands for mobile robots under the assumption that infinite forces can be asserted on the robot—a common assumption in the field of collision avoidance for indoor robots, as most robots are operated at low speed where inertial effect can be neglected. However, to operate robots safely at higher speeds (such as walking speed), it is necessary to take the robot dynamics into account. Both the dynamic window approach [12, 41] and the "curvature velocity method" [118], which despite their similarity were developed independently, are designed to deal with the dynamics of mobile robots.

To deal with obstacles that cannot be detected by the robot's sensors it is necessary to integrate model-based information into reactive collision avoidance. Only considerably small attention has been payed to this problem in the literature. In [111], an approach to motion planning is proposed which in principle could solve this problem. The emphasis of their work lies in the combination of global path planning and local collision avoidance. Here, motion commands are generated based on a global model of the environment, which is updated based on sensory input. They propose a method which efficiently extracts a path to a goal point based on such a map. Unfortunately, the authors do not discuss the problem of robot dynamics and uncertain position estimation. Furthermore it is not clear how the static model of the environment is combined with the fast-changing information obtained on-the-fly.

RHINO's current method, called μDWA [45], is specifically designed for environments where not all obstacles can be perceived, guaranteeing safe navigation with high probability even if the robot is not certain as to where it is. To the best of our knowledge, this feature is unique in the literature on mobile robot collision avoidance.

7.4 Motion Planning

Robot motion planning has been subject to intense research, as documented by a large body of literature on this topic (see e.g., [82, 114]). The majority of work addresses more complicated problems than the one addressed in this article, such as motion planning in higher-dimensional and continuous space. Motion planning for holonomic mobile robots is usually performed in 2D, ignoring costs of rotation and the dynamics of the robot. Such a methodology yields only sub-optimal results, but greatly reduces the complexity of motion planning, which is known to be exponential in the number of degrees of freedom [17, 106].

A popular algorithm for robot motion planning is A* [99], which bears close resemblance to the value iteration approach proposed here (see [71]). Both approaches can be used to find shortest paths to multiple goals, and they also generate values for arbitrary locations, facilitating rapid replanning in the face of unexpected events. The issue of efficient re-planning has previously been addressed by Stentz [122], whose D* (dynamic A*) planning algorithm is designed to quickly repair motion plans if the map is updated. RHINO's motion planner does the same; given sufficient computation time, it is guaranteed to generate optimal plans with respect to the underlying representation.

In the AI community, conditional planning has also been a subject of intense research. A recent textbook [109] provides a collection of references to the most important work in this field. Almost all of the work here is focused on domains where the model is specified by a collection of logical axioms, often expressed as STRIPS-like operators [99]. This work usually does not exploit the highly geometric nature of robot motion planning, and is therefore not directly applicable. State-based representations, such as the one used here, are commonly used in the literature on partially observable Markov decision processes (POMDPs) [64, 66, 87]. POMDPs are the mathematically the most appropriate framework to formulate the motion planning problem, since they can encompass uncertainty both in the map and the robot's location. Unfortunately, the complexity of the domain does not permit efficient planning even if today's best POMDP algorithms were employed.

7.5 Task Planning and Control

Recent research has produced a variety of frameworks for task-level control (see e.g., [40, 117] and various chapters in [75]). To our knowledge, GOLOG is unique in its seamless integration of a programming language and a powerful theorem prover for planning. Other special-purpose programming languages for mobile robots, such as COLBERT [73], do not feature built-in problem solvers. COLBERT also stays below the level of abstraction of GOLOG in that it is designed to deal with raw sensor input and motor commands. However, since the expressive power of the situation calculus exceeds that of STRIPS-like planners (which rely on an implicit close-world assumption), GOLOG is less efficient than most existing planning algorithms (see [109]).

Historically, Nilsson's SHAKEY robot was the first, successful demonstration of the synthesis of symbolic, AI-type problem solving and low-level control. While SHAKEY was a milestone in the history of mobile robotics, it suffered from a lack of robustness on the low-level side, making it too brittle to operate in domains as dynamic as the one considered in this paper. More recently, Haigh integrated PRODIGY [19] with a suite of low-level software developed for CMU's XAVIER project [119]. Her system, called ROUGE [55], uses PRODIGY to generate cost-optimal motion plans for XAVIER, a robot navigating in the corridors of an office building. This approach differs from ours in that it does not offer a programming option on the task control level, i.e., all plans are generated by the planner, not by the programmer.

7.6 Human Robot Interfaces

Most existing robots don't possess user-friendly interfaces. This is because mobile robotics research tends to focus on navigational issues, not on human robot interaction. Nevertheless, the need for more effective human robot interfaces has clearly been recognized. For example, in his M.Sc. thesis, Torrance developed a natural language interface for teaching mobile robots names of places in an indoor environment [131]. Due to the lack of a speech recognition system, his interface still required the user to operate a keyboard; however, the natural language component made instructing the robot significantly easier. More recently, Asoh and colleagues [3] developed an interface that integrates a speech recognition system into a phrase-based natural language interface. They successfully instructed their "office-conversant" robot to navigate to office doors and other significant places in their environment, using verbal commands. Other researchers have proposed vision-based interfaces that allow people to instruct mobile robots via arm gestures. For example, Kortenkamp and colleagues [76] recently developed a gesture-based interface, which is capable of recognizing arm poses such as pointing towards a location on the ground. In a similar effort, Kahn and colleagues [67] developed a gesture-based interface which has been demonstrated to reliably recognize static arm poses (pose gestures) such as pointing. This interface was successfully integrated into Firby's reactive plan-execution system RAP [40], where it enabled people to instruct a robot to pick up free-standing objects. A recent paper by Waldherr and colleagues [133] extends these approaches to *dynamic* gestures, i.e., gestures which are defined through motion, not just poses. Motion gestures, which are commonly used for communication among people, provide additional freedom in the design of gestures. In addition, they reduce the chances of accidentally classifying arm poses as gestures that were not intended as such.

Unfortunately, while these interfaces are important steps in the right direction, they are not quite suited for application domains as museums, where people typically interact with a robot for an extremely short duration. Most visitors spent less than 15 minutes following the robot through the museum, and even RHINO's most enthusiastic supporters stayed rarely for more than two hours. Under such circumstances, it is important that the robot appeals to the "intuitions" of its users. It is generally undesirable that users have to learn how to interact with a robot, even if the nature of the interface (language, gestures) facilitates this process. We believe that we currently lack a convincing methodology for "intuitive" human robot interaction.

7.7 Integrations and Applications

Various researchers have devised integrated systems similar to the one described here [75, 113]. A good survey of fielded systems is provided in a recent book by Schraft and Schmierer [113]. For example, Siemens Corp. (Germany) has recently developed a mobile cleaning robot, which it successfully deployed in a supermarket [36]. Unfortunately, much of their technology is proprietary. The robot differs from RHINO in that does not interact with people; it also moves at much lower speed. We suspect that the basic software architecture is similar to the one used by RHINO. A second, prominent example of a successfully deployed robot is Helpmate Inc.'s Helpmate robot [70]. This robot has been deployed at dozens of hospitals worldwide, where it carries food and other items through hospital corridors. The Helpmate does not interact with people either, but it features an easy-to-use interface which enables nurses to tell the robot where to go.

To the best of our knowledge, the concept of a tour-guide robot was first proposed by Horswill [59, 60], who built a robot that guided visitors through the AI Lab at MIT. This robot, designed to demonstrate Horswill's fast vision routines, did not require modifications of the environment, just like RHINO. However, it lacked the ability to modify its map; neither could it avoid collisions with invisible obstacles. RHINO's approach relies to a much larger degree on internal state (location, map), which accounts for its enhanced level of robustness and capability required in environments such as the one considered here.

7.8 Robots on the Web

In recent years, several research teams have connected robots to the Web, enabling people all over the world to command robots remotely. One of the most prominent examples is CMU's XAVIER robot [119], which predates the work reported here. XAVIER is equipped with an interface similar to the one described here. Web-users can schedule requests for moving to pre-specified locations where the robot tells a user-selected "joke," and they can watch camera images recorded in regular time intervals. RHINO's interface offers a choice between JPEG images and Java applets; the latter option enables Web users to run a robot simulator on their own machine, in which the robot's location is animated continuously. Users command both robots at an abstract level, and the low-level navigation software ensures the robot's safety.

Others have designed interfaces for the control of robot arms. The MERCURY system [47], which started its operation in 1994, was one of the first tele-operated manipulators on the Web. It enabled Web users to excavate artifacts buried in a sand-filled terrarium. This system required users to assume exclusive control over the manipulator. The TELE-GARDEN, successor of MERCURY, enabled people to plant flowers [48]. Just like RHINO, this system was able to coordinate requests by multiple users. The "Mechanical Gaze Project" [102] enables visitors of a museum to examine exhibits from various viewpoints, and with different distances. Other Web-connected robots include a tele-operated "Eyebot", a robot that carries a camera whose pictures are posted on the Web (`http://www.dma.nl/eyebot`), and "KhepOnTheWeb", a table-top robot that Web users can manually move through a maze (`http://KhepOnTheWeb.epfl.ch`).

8 Summary

This article described the software architecture of a fully autonomous mobile robot designed to interact with people. The robot has been proven to function reliably in unmodified and densely populated environments, and it has equally been proven to interact successfully with people. To reach this level of reliability, our approach extends a collection of well-known algorithms for mobile robot state estimation, control, and user interaction, and integrates them into a single system. While most of the work reported here is well-grounded in previously approaches to mobile robotics, some critical innovations were necessary to provide the robot with the necessary level of robustness.

RHINO's localization method is based on Markov localization, a popular probabilistic localization method. Our approach extends the Markov localization paradigm by a method for filtering out noisy sensor data, which makes it extremely robust in highly dynamic environments. It also estimates the location of a robot at much higher resolution; a necessary requirement for navigating close to obstacles that cannot be sensed. The robot inherits its mapping algorithm from the decade-old literature on occupancy grid maps. It employs artificial neural networks for sensor interpretation, which allow it to interpret sensor readings in the context of its neighbors. As argued elsewhere [127], such an approach improves the robustness of the mapping algorithm to noise (such as spectral reflection). RHINO also integrates maps from different sensor modalities.

RHINO employs a hybrid collision avoidance method (μDWA), specifically designed to operate in environments where not all obstacles can be sensed reliably. In addition to the sensor readings, this approach assembles "virtual" sensor readings using a map of the environment. A key feature is its ability to generate safe motion even if the robot does not quite know where it is. This contrasts previous methods, which are typically purely sensor-based and would thus fail in an environment as the museum. RHINO's motion planner is a version of dynamic programming, which, by spreading activation through free-space, effectively pre-plans for all possible exceptions. To accommodate the continuous change of the map, the path planner contains a method for identifying where and when replanning becomes necessary. Task-level control is performed by GOLOG, an integrated programming language and theorem prover embedded in the situation calculus. While this approach provides a powerful framework for designing high-level controllers, it alone is inappropriate to deal with various contingencies arising in robot control. We therefore designed GOLEX, an interface which provides the glue between GOLOG and the rest of the software. GOLEX provides macros, conditional plans and an execution monitor.

RHINO exhibits a degree of interactiveness typically not found on other mobile robots. It reacts to the presence of people in various ways—an aspect that we found to be essential to spark people's enthusiasm. Both its command interface and its interface to the Web was specifically designed to appeal to novices. All these components are integrated through a modular software architecture, designed to accommodate the various bottlenecks in distributed computer networks. All computation is carried out completely asynchronously, and the different modules communicate by sending messages to each other. Computationally-intense modules were capable of adapting their requirements to the available resources, which is critical in a network that was often subject to lengthy communication delays.

During its six-day installation period, the robot performed reliably and excited the visitors. The robot guided thousands of users to exhibits with almost perfect reliability, traversing more than 18.5km at an average speed of almost 40 cm/sec. We did not modify the museum in any way that would facilitate the robot's operation; in fact, RHINO's software successfully coped with various "invisible" obstacles and the large number of people. We believe that RHINO's reliability and effectiveness in complex environments is unprecedented. The robot also raised the museum's attendance by more than 50%, suggesting that robotic applications in the field of entertainment and education might be commercially viable.

9 Discussion

One of the key limitations of the current approach is its reliance on an accurate map of the environment. In the Deutsches Museum Bonn, it took us about a week to manually acquire the map, partially because of the non-orthogonal arrangement of exhibits and walls. To remedy this problem, we recently devised a family of more powerful mapping techniques [128, 130]. These techniques make it possible to acquire maps of large-scale, cyclic environments from raw sensor data, collected while joy-sticking the robot through its environments (see also Section 7.2 and Figure 11). Augmenting such maps with "invisible" obstacles is straightforward, as the robot's localization methods can be used to accurately determine its position. These new techniques overcome an important limitation of the approach described here, effectively reducing the installation time for a robot by more than an order of magnitude.

Another limitation arises from the fact that the entropy gain filter, used to filter out corrupted sensor readings, impairs the robot's ability to recover from global localization failure. As noted above, once the robot has lost track of its position entirely—which luckily never happened in the museum—, the entropy gain filter may filter out *all* authentic sensor readings, making it impossible for the robot to re-localize itself. Recently, we proposed *novelty filters* [45], an extension of entropy gain filters that takes into account the nature of the surprise in a sensor reading. In a systematic study using the data collected in the museum we found that novelty filters do not suffer this limitation, while still retaining the full advantage of the entropy gain filter. We envision that this new filter will lead to more robust behavior in highly dynamic environments.

RHINO is just one out of a series of recent successful mobile robots [113]. Recent research in the field of mobile robotics has led to significant progress along various dimensions. Applications such as robots that guide blind or mentally handicapped people, robots that clean large office buildings and department stores, robots that assist people in recreational activities, etc., are clearly in reach, and for many of those target domains prototypes are readily available [113]. This recent, ongoing revolution has been triggered by advances along various dimensions. Robotic hardware has steadily become cheaper and more reliable. Robotic software has matured, reaching critical levels of reliability, robustness, and flexibility.

We believe that the museum tour-guide is a prototypical example of a new generation of mobile service robots. Many of the challenges in the museum tour-guide domain apply to a wide variety of mobile robot applications: the necessity to navigate through highly dynamic and unstructured environments; the necessity to interact with people; and the necessity to operate in environments that cannot be modified. It is quite likely that robots similar to the one described here will soon be deployed in shopping malls, amusement centers, technology fairs, etc., where they will be receptionists, information kiosks, waiters, guides, but most of all: magnets that attract people. Similar robots may soon perform janitorial services, operate at sites too dangerous for humans, or assist people in various aspects of their lives.

Through developing RHINO's software and watching it operate in the Deutsches Museum, we learned more than we can possibly describe in a single article. Among the most important lessons is the recognition that mobile robot navigation has progressed to a level at which robots can now navigate reliably even in densely populated spaces. In most aspects, such robots can now be installed in new sites within days, without having to modify the environment (see also [128]). We also learned that human robot interfaces are key prerequisites if robots are to become part of people's everyday lives. We found that adaptive mechanisms are essential for operating robots in highly dynamic and unpredictable environments, as without the capability to revise its plans, the robot would often have been stuck. Finally, we learned that entertainment is a highly promising application domain for mobile robotics. In most envisioned service robot applications, robots have to compete with human labor, whereas in the entertainment sector, robots may generate revenue by simply exploiting the fact that they differ.

The following is a highly subjective list of what we believe include some of the most promising research directions for indoor mobile robotics research:

- **Long living robots.** How can we build software that enables robots to operate reliably for many years? Such robots must be able to adapt to changes in their environments, changes in their physical properties, and they must also be able to perform new tasks. To what extent can we build robots that survive sensor and actuator failures? Most of today's mobile service robots are only deployed for short periods of time and are only capable of performing a single task. A science of software mechanisms that enable robots to operate for extended stretches of time, and to perform many tasks, would almost certainly lead to revolutionary changes in mobile robotics.

- **Human robot interaction/cooperation.** How can we build robots that can be installed and instructed by non-expert users? How can we design robots that can interact with people, and support them in their everyday activities? Most of today's mobile robots are installed and operated by experts, and their interactive capabilities are still poorly developed. To bring robots into the everyday life of people, advances in various aspects of human robot interaction are urgently needed.

- **Taskable robot teams.** How can we build large teams of mobile robots to collaboratively perform interesting tasks? Such software must be able to deal with individual robot failures and limited communication channels. We specifically lack methods that makes teams of robots taskable, i.e., that enable teams of robots to perform diverse families of user-specified tasks.

- **Mobile manipulation.** The integration of mobility and manipulation is essential for a wide range of service tasks. However, mobile robotics and robotic manipulations have largely remained separate fields, and integrations of both are often brittle at best. How can we build highly dextrous robots that can both manipulate their environment and navigate therein?
- **Flexible software architectures.** While recent research has led to a large corpus of isolated component technologies, we still lack effective methods for their integration. How can we build software architectures that facilitate the assembly of large-scale robotics software? Learning, in particular, appears to be promising for providing the "glue" between different components that otherwise may not fit together.

While this list is highly subjective, it seeks to identify some of the most promising new research directions in mobile robotics. Progress along any of those dimensions would almost certainly lead to interesting new science, along with practical algorithms with high societal impact.

Acknowledgment

The authors thank Peter Frieß and his staff from the Deutsches Museum Bonn for their enthusiastic support of this project.

References

[1] I. Asimov. *Runaround*. Faucett Crest, New York, 1942.

[2] I. Asimov. *I, Robot*. Doubleday, 1950.

[3] H. Asoh, S. Hayamizu, I. Hara, Y. Motomura, S. Akaho, and T. Matsui. Socially embedded learning of office-conversant robot jijo-2. In *Proceedings of IJCAI-97*. IJCAI, Inc., 1997.

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

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

[6] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK, 1994.

[7] J. Borenstein. *The Nursing Robot System*. PhD thesis, Technion, Haifa, Israel, June 1987.

[8] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[9] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 572–577, 1990.

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

[11] R.A. Brooks. A robust layered control system for a mobile robot. Technical Report AI memo 864, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[12] 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(2):31–38, Summer 1995.

[13] W. Burgard, A.B. Cremers, D. Fox, G. Lakemeyer, D. Hähnel, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc.of the Fifteenth National Conference on Artificial Intelligence*, 1998. To appear.

[14] W. Burgard, A. Derr, D. Fox, and A.B. Cremers. Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach. In *Proc. of the IEEE/RSJ International Conference ond Intelligent Robots and Systems (IROS'98)*, 1998. To appear.

[15] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the Fourteenth National Conference on Artificial Intelligence*, pages 896–901, 1996.

[16] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. In *Proc. of the Fifteenth International Conference on Artificial Intelligence (IJCAI-97)*, 1997.

[17] J. Canny. *The Complexity of Robot Motion Planning*. The MIT Press, Cambridge, MA, 1987.

[18] K. Čapek. *R.U.R. (Rossum's Universal Robots)*. Out of print, 1921.

[19] J.G. Carbonell, C.A. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. Van Lehn, editor, *Architectures for Intelligence*. Erlbaum, 1990.

[20] G.C. Casella and R.L. Berger. *Statistical Inference*. Wadsworth & Brooks, Pacific Grove, CA, 1990.

[21] R. Chatila and J.-P. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, 1985.

[22] H. Choset. *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. PhD thesis, California Institute of Technology, 1996.

[23] H. Choset, I. Konuksven, and J.W. Burdick. Sensor Based Planning for a Planar Rod Robot. In *Proc. IEEE/SICE/RSJ Int. Conf. on Multisensor Fusion on Multisensor Fusion and Integration for Intelligent Systems*, Washington, DC, 1996.

[24] H. Choset, I. Konuksven, and A. Rizzi. Sensor Based Planning: A Control Law for Generating the Generalized Voronoi Graph. In *Submitted to Proc. IEEE Int. Advanced Robotics*, Washington, DC, 1996.

[25] E. Chown, S. Kaplan, and D. Kortenkamp. Prototypes, location, and associative networks (plan): Towards a unified theory of cognitive mapping. *Cognitive Science*, 19:1–51, 1995.

[26] T.S. Collet and B.A. Cartwright. Landmark learning in bees. *Journal of Comparative Physiology*, January 1985.

[27] J. Connell. *Minimalist Mobile Robotics*. Academic Press, Boston, 1990.

[28] I.J. Cox. Blanche—an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

[29] I.J. Cox. Modeling a dynamic environment using a bayesian multiple hypothesis approach. *Artificial Intelligence*, 66:311–344, 1994.

[30] I.J. Cox and G.T. Wilfong, editors. *Autonomous Robot Vehicles*. Springer Verlag, 1990.

[31] G. de Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. In *AAAI 1998 Fall Symposium on Cognitive Robotics*, 1998. To appear.

[32] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, 1998. To appear.

[33] 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.

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

[35] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[36] H. Endres, W. Feiten, and G. Lawitzky. Field test of a navigation system: Autonomous cleaning in supermarkets. In *Proc. of the 1998 IEEE International Conference on Robotics & Automation (ICRA 98)*, 1998.

[37] S. Engelson. *Passive Map Learning and Visual Place Recognition*. PhD thesis, Department of Computer Science, Yale University, 1994.

[38] H.R. Everett, D.W. Gage, G.A. Gilbreth, R.T. Laird, and R.P. Smurlo. Real-world issues in warehouse navigation. In *Proceedings of the SPIE Conference on Mobile Robots IX*, Boston, MA, November 1994. Volume 2352.

[39] C. Fedor. *TCX. An interprocess communication system for building robotic architectures. Programmer's guide to version 10.xx.* Carnegie Mellon University, Pittsburgh, PA 15213, 12 1993.

[40] R.J. Firby, R.E. Kahn, P.N. Prokopowicz, and M.J. Swain. An architecture for active vision and action. In *Proceedings of IJCAI-95*, pages 72–79, 1995.

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

[42] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 1997.

[43] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 1998. Accepted for publication.

[44] D. Fox, W. Burgard, S. Thrun, and A.B. Cremers. A hybrid collision avoidance method for mobile robots. In *Proc. of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium, 1998.

[45] D. Fox, W. Burgard, S. Thrun, and A.B. Cremers. Position estimation for mobile robots in dynamic environments. In *Proc.of the Fifteenth National Conference on Artificial Intelligence*, 1998. To appear.

[46] T. Fukuda, S. Ito, N. Oota, F. Arai, Y. Abe, K. Tanake, and Y. Tanaka. Navigation system based on ceiling landmark recognition for autonomous mobile robot. In *Proc. Int'l Conference on Industrial Electronics Control and Instrumentation (IECON'93)*, volume 1, pages 1466 – 1471, 1993.

[47] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, and J. Wiegley. Desktop tele-operation via the world wide web. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1995.

[48] K. Goldberg, J. Santarromana, G. Bekey, S. Gentner, R. Morris, J. Wiegley, and E. Berger. The telegarden. In *Proc. of ACM SIGGRAPH*, 1995.

[49] R. Greiner and R. Isukapalli. Learning to select useful landmarks. In *Proceedings of 1994 AAAI Conference*, pages 1251–1256, Menlo Park, CA, 1994. AAAI Press / The MIT Press.

[50] J.-S. Gutmann. Vergleich von Algorithmen zur Selbstlokalisierung eines mobilen Roboters. Master's thesis, University of Ulm, Ulm, Germany, 1996. In German.

[51] J.-S. Gutmann and B. Nebel. Navigation mobiler Roboter mit Laserscans. In *Autonome Mobile Systeme*. Springer Verlag, Berlin, 1997. In German.

[52] J.-S. Gutmann and C. Schlegel. AMOS: Comparison of scan matching approaches for self-localization in indoor environments. In *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[53] D. Guzzoni, A. Cheyer, L. Julia, and K. Konolige. Many robots make short work. *AI Magazine*, 18(1):55–64, 1997.

[54] D. Hähnel, W. Burgard, and G. Lakemeyer. GOLEX — bridging the gap between logic (GOLOG) and a real robot. In *Proc. of the 22nd German Conference on Artificial Intelligence (KI'98)*, LNCS. Springer Verlag, 1998. To appear.

[55] K.Z. Haigh and M.M. Veloso. High-level planning and low-level execution: towards a complete robotic agent. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, February 1997.

[56] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley Pub. Co., Redwood City, California, 1991.

[57] J. Hertzberg and F. Kirchner. Landmark-based autonomous navigation in sewerage pipes. In *Proc. of the First Euromicro Workshop on Advanced Mobile Robots*, pages 68–73. IEEE Computer Society Press, 1996.

[58] 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.

[59] I. Horswill. Polly: A vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*. MIT Press, 1993.

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

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

[62] H. Hu and M. Brady. A Bayesian approach to real-time obstacle avoidance for a mobile robot. In *Autonomous Robots*, volume 1, pages 69–92. Kluwer Academic Publishers, Boston, 1994.

[63] E. Huber and D. Kortenkamp. Using stereo vision to pursue moving agents with a mobile robot. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1995.

[64] T. Jaakkola95a, S.P. Singh, and M.I. Jordan. Reinforcement learning algorithm for partially observable decision problems. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1995. MIT Press.

[65] L.P. Kaelbling, A.R. Cassandra, and J.A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[66] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. 1996.

[67] R.E. Kahn, M.J. Swain, P.N. Prokopowicz, and R.J. Firby. Gesture recognition using the perseus architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 734–741, San Francisco, CA, 1996.

[68] M. Khatib and R. Chatila. An extended potential field approach for mobile robot sensor-based motions. In *Proc. International Conference on Intelligent Autonomous Systems (IAS'4)*, 1995.

[69] O. Khatib. Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[70] S. King and C. Weiman. Helpmate autonomous mobile robot navigation system. In *Proceedings of the SPIE Conference on Mobile Robots*, pages 190–198, Boston, MA, November 1990. Volume 2352.

[71] S. Koenig. The complexity of real-time search. Technical Report CMU-CS-92-145, Carnegie Mellon University, April 1992.

[72] S. Koenig and R. Simmons. Passive distance learning for robot navigation. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, 1996.

[73] K. Konolige. Colbert: A language for reactive control in saphira. In *KI-97: Advances in Artificial Intelligence*, LNAI, pages 31–52. Springer Verlag, 1997.

[74] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proc. IEEE Int. Conf. Robotics and Automation*, April 1991.

[75] D. Kortenkamp, R.P. Bonassi, and R. Murphy, editors. *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA, 1998. MIT Press.

[76] D. Kortenkamp, E. Huber, and P. Bonassi. Recognizing and interpreting gestures on a mobile robot. In *Proceedings of AAAI-96*, pages 915–921. AAAI Press/The MIT Press, 1996.

[77] D. Kortenkamp, I. Nourbakhsh, and D. Hinkle. The 1996 AAAI mobile robot competition and exhibition. *AI Magazine*, 18(1):25–32, 1997.

[78] 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.

[79] B. Kuipers and Y.-T. Byun. A robust qualitative method for spatial learning in unknown environments. In *Proceeding of Eighth National Conference on Artificial Intelligence AAAI-88*, Menlo Park, Cambridge, 1988. AAAI Press / The MIT Press.

[80] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[81] G. Lakemeyer. On sensing and off-line interpreting in golog. In *AAAI 1998 Fall Symposium on Cognitive Robotics*, 1998. To appear.

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

[83] J.J. Leonard and H.F. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic Publishers, Boston, MA, 1992.

[84] J.J. Leonard, H.F. Durrant-Whyte, and I.J. Cox. Dynamic map building for an autonomous mobile robot. *International Journal of Robotics Research*, 11(4):89–96, 1992.

[85] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

[86] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation, special issue on actions and processes*, 4:665–678, 1994.

[87] M.L. Littman, A.R. Cassandra, and L.P. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

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

[89] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 1998. To appear.

[90] D.J.C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, Pasadena, California, 1992.

[91] M.J. Matarić. A distributed model for mobile robot environment-learning and navigation. Master's thesis, MIT, Cambridge, MA, January 1990. also available as MIT AI Lab Tech Report AITR-1228.

[92] P.S. Maybeck. The Kalman filter: An introduction to concepts. In Cox and Wilfong [30].

[93] J. McCarthy. Situations, actions and causal laws. In *Semantic Information Processing*, pages 410–417. MIT Press, 1968.

[94] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

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

[97] U. Nehmzow, T. Smithers, and J. Hallam. Location recognition in a mobile robot using self-organizing feature maps. In G. Schmidt, editor, *Information Processing in Autonomous Mobile Robots*. springer Verlag, 1991.

[98] H. Neven and G. Schöner. Dynamics parametrically controlled by image correlations organize robot navigation. *Biological Cybernetics*, 75:293–307, 1996.

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

[100] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2):53–60, Summer 1995.

[101] S. Oore, G.E. Hinton, and G. Dudek. A mobile robot that learns its place. *Neural Computation*, 9:683–699, 1997.

[102] E. Paulos and J. Canny. Delivering real reality to the world wide web via telerobotics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

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

[104] L. Peters, H. Surmann, S. Guo, K. Beck, and J. Huser. Moria fuzzy logik gesteuertes, autonomes fahrzeug. 1994.

[105] 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.

[106] J.H. Reif. Complexity of the mover's problem and generalizations. In *Proceesings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[107] 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.

[108] 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.

[109] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[110] B. Schiele and J.L. Crowley. A comparison of position estimation techniques using occupancy grids. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1628–1634, 1994.

[111] K. Schmidt and K. Azarm. Mobile robot navigation in a dynamic world using an unsteady diffusion equation strategy. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992.

[112] 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.

[113] R.D. Schraft and G. Schmierer. *Serviceroboter*. Springer Verlag, 1998. In German.

[114] J.T. Schwartz, M. Scharir, and J. Hopcroft. *Planning, Geometry and Complexity of Robot Motion*. Ablex Publishing Corporation, Norwood, NJ, 1987.

[115] H Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of IJCAI-97*. IJCAI, Inc., 1997.

[116] H. Shatkay and L.P. Kaelbling. Learning hidden markov models with geometric information. Technical Report CS-97-04, Computer Science Department, Brown University, Providence, RI, April 1997.

[117] R. Simmons. Concurrent planning and execution for autonomous robots. *IEEE Control Systems*, 12(1):46–50, February 1992.

[118] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc. of the IEEE International Conference on Robotics and Au tomation*, 1996.

[119] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, February 1997.

[120] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. International Joint Conference on Artificial Intelligence*, 1995.

[121] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[122] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of IJCAI-95*, 1995.

[123] S. Thrun. Exploration and model building in mobile robot domains. In *Proceedings of the ICNN-93*, pages 175–180, 1993.

[124] S. Thrun. To know or not to know: On the utility of models in mobile robotics. *AI Magazine*, 18(1):47–54, 1997.

[125] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 1998. To appear.

[126] S. Thrun. Finding landmarks for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[127] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[128] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53, 1998. Also appeard in *Autonomous Robots* 5, pp. 253–271, joint issue.

[129] S. Thrun, D. Fox, and W. Burgard. Probabilistic mapping of an environment by a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1998.

[130] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proceedings of AAAI-98*. AAAI Press/The MIT Press, 1998. To appear.

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

[132] G. von Randow. *Roboter: Unsere Nächsten Verwandten*. Rohwolt Verlag, Reinbek, Germany, 1997.

[133] S. Waldherr, S. Thrun, R. Romero, and D. Margaritis. Template-based recognition of pose and motion gestures on a mobile robot. In *Proceedings of AAAI-98*. AAAI Press/The MIT Press, 1998. To appear.

[134] G. Weiß, C. Wetzler, and E. von Puttkamer. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 595–601, 1994.

[135] E. Wolfart, R.B. Fisher, and A. Walker. Position refinement for a navigating robot using motion information based on honey bee strategies. In *Proceesings of the International Symposium on Robotic Systems (SIR 95)*, Pisa, Italy, 1995. also appeared as DAI Research Paper No 751, University of Edinburgh, Department of Artificial Intelligence, U.K.

[136] B. Yamauchi and P. Langley. Place recognition in dynamic environments. *Journal of Robotic Systems*, Special Issue on Mobile Robots, To appear.

[137] B. Yamauchi, P. Langley, A. C. Schultz, J. Grefenstette, and W. Adams. Magellan: An integrated adaptive architecture for mobile robots. Technical Report 98-2, Institute for the Study of Learning and Expertise (ISLE), Palo Alto, CA, May 1998.

[138] U.R. Zimmer. Robust world-modeling and navigation in a real world. *Neurocomputing*, 13(2–4), 1996.