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[‡]

Computer Science Department III, University of Bonn, Germany

[†]*Computer Science Department V, Technological University of Aachen, Germany*

[‡]*Computer Science Department and Robotics Institute, Carnegie Mellon University, Pittsburgh, PA*

Abstract

This article describes the software architecture of an autonomous, interactive tour-guide robot. It presents a modular and distributed software architecture, which integrates localization, mapping, collision avoidance, planning, and various modules concerned with user interaction and Web-based telepresence. At its heart, the software approach relies on probabilistic computation, on-line learning, and any-time algorithms. It enables robots to operate safely, reliably, and at high speeds in highly dynamic environments, and does not require any modifications of the environment to aid the robot's operation. Special emphasis is placed on the design of interactive capabilities that appeal to people's intuition. The interface provides new means for human-robot interaction with crowds of people in public places, and it also provides people all around the world with the ability to establish a "virtual telepresence" using the Web. To illustrate our approach, results are reported obtained in mid-1997, when our robot "RHINO" was deployed for a period of six days in a densely populated museum. The empirical results demonstrate reliable operation in public environments. The robot successfully raised the museum's attendance by more than 50%. In addition, thousands of people all over the world controlled the robot through the Web. We conjecture that these innovations transcend to a much larger range of application domains for service robots.

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

1 Introduction

Ever since the Czech novelist Karel Čapek invented the term "robot" [154]—which was later popularized by Isaak Asimov [2,3]—the dream of building autonomous

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 (see, e.g., [128,155]). 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 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 concept. 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 track of 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 cases, 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

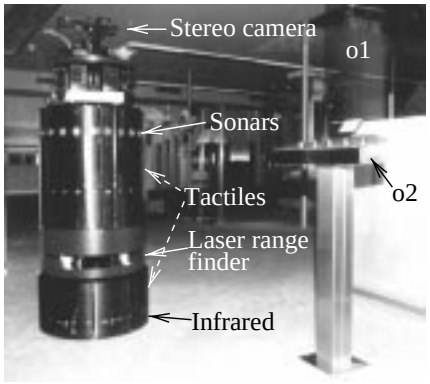

Fig. 1: The robot and its sensors.

Fig. 2: RHINO, pleasing the crowd.

with sonar. Other 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, usually 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 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 ability to navigate.

These challenges mandated the development of a collection of new, innovative software solutions. Our past work (e.g., [16,149]) focused primarily on office navigation—which involved moving through static corridors and into offices with well-detectable obstacles and cooperative people—where many of the difficulties simply did not exist. Many of the assumptions underlying this work do not apply in populated public places, such as a museum. As a result, we had to develop several new techniques and, more importantly, changed our view on several well-established methodologies in the field.

For example, most successful mobile robot architectures employ sensor-based, reactive methods for real-time collision avoidance [85]. The typical paradigm is to consider a short time window of past sensor readings when setting the speed and final motion direction of the robot. Of course, the purely sensor-based approach is inappropriate if obstacles cannot be sensed. RHINO employs a hybrid approach, which incorporates a map of the environment in addition to sensor readings. The map contains "invisible" obstacles and hazards such as staircases. For the map to be useful, however, the robot has to know where it is with high accuracy. RHINO employs an efficient, fine-grain variant of Markov localization, capable of tracking the robot's position with high accuracy. Localization proved more difficult in the museum than in our office environment. Markov localization relies on the assumption that the robot's location is the only state in the world. However, large numbers of people that followed the robot closely, thereby blocking most of its sensors constantly violate this assumption. As demonstrated here and elsewhere [53], the basic Markov localization algorithm would have failed under such conditions. RHINO employs an extended localization method that filters out corrupted sensor readings. This approach, described in detail below, considers only sensor data that reduce the uncertainty of the robot during localization. As a result, the robot pays only attention to sensor data that confirms its current belief while ignoring all other data. While this approach critically departs from the basic Markov approach, it makes localization succeed in this highly non-Markovian environment.

Most navigation approaches either rely on a fixed, pre-given map of the environment, or learn such a map by themselves. Neither of these two options was appropriate—a fixed map would have made it impossible to navigate successfully when obstacles blocked the robot's path persistently (such as the stools). A pure learning approach, which was used successfully in a prototype tour-guide robot that was previously installed in our university building [149], was inappropriate since the robot was unable to map those invisible obstacles. RHINO's software integrates both types maps, using a hand-crafted CAD map as a starting point and a map learned on-the-fly from sensor data. To cope with changing maps, RHINO uses an efficient motion planner that can quickly react to changes in the map.

Finally, a key ingredient of any robot that interacts with people in public places is its interactive component. To interact with people, a method for finding people is called for. Unfortunately, sensor differencing (e.g., image differencing) is inapplicable since it typically assumes that the people move and the robot doesn't—in our case, the robot was almost always in motion, while people often didn't move. Thus, RHINO finds people by comparing the map with its range measurements, using the inverse of the filter described above. The robot then invokes a series of means that inform people of the robot's intentions and goals. RHINO also possesses two user interfaces, one for visitors and one for Web-users, which are designed to be simple enough for people to operate even without prior exposure to robotics.

This article provides an overview of the major components of RHINO's software

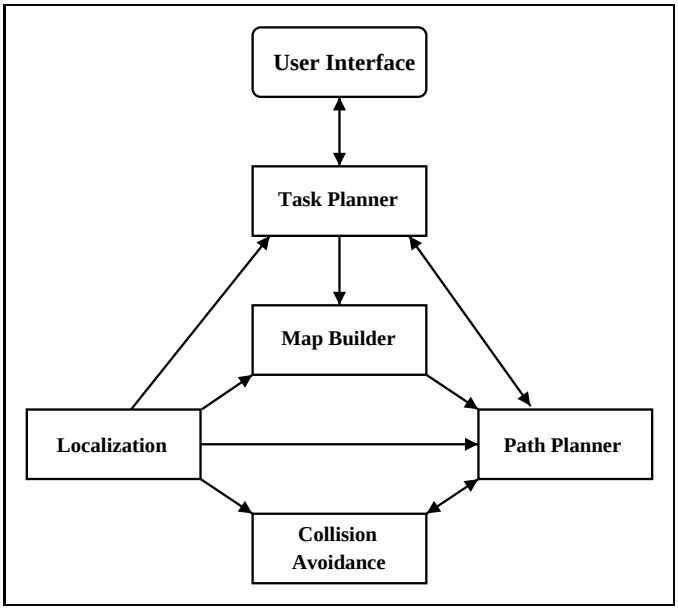

Fig. 3: Major components of the RHINO system and major flow of information.

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 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 20 modules (processes) listed in Table 1, 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 [46], 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 [55,138], 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,

Communication		
tcxServer		TCP/IP communication interface
router		communication pipeline for low-bandwidth radio communication
Hardware Interfaces		
baseServer	(*)	interface to motors and sonar sensors
laserServer	(*)	interface to laser range finder
buttons	(*)	interface to buttons
sound	(*)	interface to on-board CD player
showpic	(*)	interface to on-board graphics display
pantilt	(*)	interface to pan/tilt unit
meteor	(*)	interface to on-board cameras
scenecam		interface to off-board camera
meteorServer		module for synchronizing images from the on-board and the off-board camera
Navigation, Planning, and Control		
colliServer	(**)	collision avoidance module
collgraph		display for collision avoidance
obstacle-server		provider of "virtual" (map-based) obstacles
localize		localization module
laserint		conversion of raw sensor data into local maps
map		global mapper
plan		motion planner
hli		interface between high-level mission planner, user interface, and lower level software
eclipse		high-level planner
User and Web Interfaces		
detection		module for people detection
positionServer		auxiliary module which communicates robot positions to to the Java applets (Web)
picServer(2)		two auxiliary modules, both of which communicate images and goal locations to the Java applets (Web), one of which located in the museum, and the other is located at a remote high-speed node
map-applet		module for displaying the map and the robot's status in the Web
explanation-applet		control of in-time explanations of exhibits and robot actions/intentions
tour_select.cgi		module for processing input by Web users, and interface to the high-level mission planner
frame.cgi		module for the selection of image update rates (by Web user)
commander		command interface for on-line monitoring and tele-operation

Table 1
Complete listing of all software modules. Modules labeled "(*)" must be and those labeled "(**)" should be run on board the robot. The exact on-board configuration during the exhibition varied; however, a dual Pentium computer is sufficient to execute all major components on-board.

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 [19,66,74,111,139], 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 better or more accurate the result. In RHINO's software, resource flexibility is achieved by two mechanisms: selective data processing and any-time algorithms [38,162]. 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 decentral. 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 idea of decentralized, distributed decision making has been at the core of research on behavior-based robotics over the last decade [1,15,121], but here modules are typically much lower in complexity (e.g., simple finite state machines).

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 [19,74,111,139,145]. 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

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

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

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

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 generally cause a loss of information due to the inaccurate nature of robot motion, and increase the entropy of $P(\xi^{(t)})$. The entropy of $P(\xi^{(t)})$ will be discussed further below in Section 3.1.6.

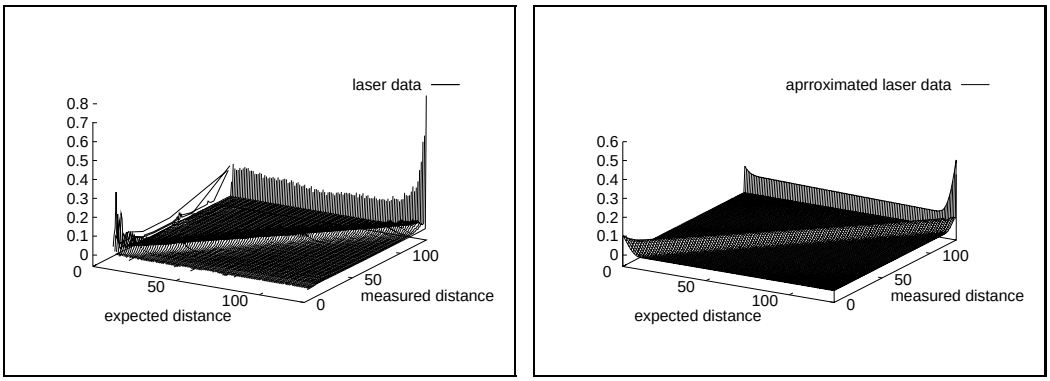

Fig. 4: The conditional probability $P(o \mid dist(\xi))$ obtained from 11,000,000 laser measurements (left) and its approximation using a mixture of a Gaussian, a uniform and a Dirac density (right).

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

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

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 *kidnapped robot problem* [44].

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:

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

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

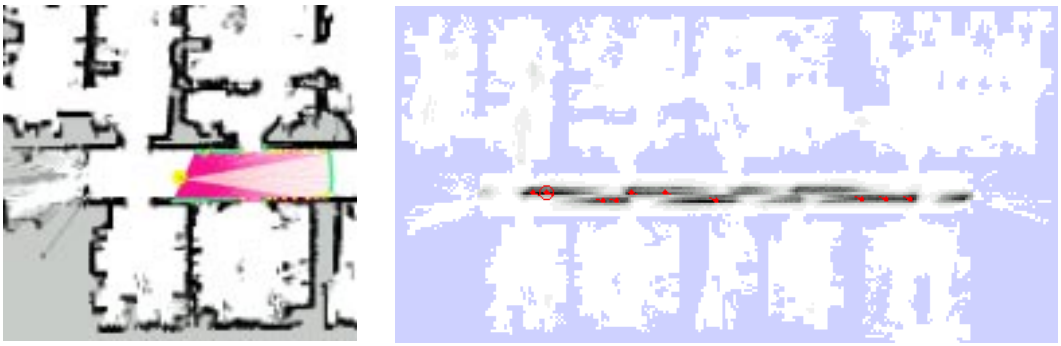

Fig. 5. Probabilistic model of perception: (a) Laser range scan, projected into a previously acquired map. (b) The probability $P(o \mid \xi)$, evaluated for all positions ξ and projected into the map (shown in grey). The darker a position, the larger $P(o \mid \xi)$.

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*:

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

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 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:

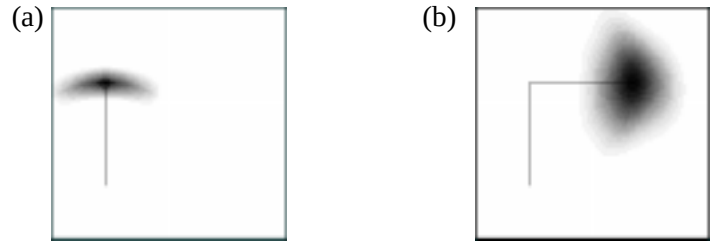

Fig. 6. Probabilistic model of robot motion: Accumulated uncertainty after moving (a) 40 meter, (b) 80 meter.

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

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 geometric, and a Dirac distribution, as shown in the right diagram in Figure 4. The coefficients of these distribution are learned from data, using the maximum likelihood estimator [8].

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.

In our implementation, the parameters of the perceptual model are obtained through maximum likelihood estimation from data, i.e., pairs of measurements o and "true" distances $dist(\xi)$. Since such data is difficult to obtain—it requires knowledge of the exact robot location, a bootstrapping algorithm was used to automatically derive position data. More specifically, our approach relies with position labels derived using an approximate perpetual model, and used these approximate positions to optimize the model parameters. Once the model parameters have been fit, new position labels are computed using the improved perceptual model. This approach is iterated in an EM-like fashion [40], leading to increasingly better data fits. Notice that this approach bears close similarity to a rich body on research on learning from labeled and unlabeled data [9,24,25,113,109,132], and is commonly used in other data-intense fields such as speech recognition [156]. As a result, our approach can effortlessly use millions of data items gathered during everyday operation, for building a highly accurate perceptual model.

3.1.3 Robot Motion

Motion changes the location of the robot. 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:

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

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 *kinematic 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 [19,150]. The width of these distributions are proportional to the length of the 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 2. 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 [19,20,51].

Recently, different variants of Markov localization have been developed [19,74,111,139]. These methods can be roughly distinguished by the nature of the state space representation. Virtually all published implementations of Markov localization, with

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

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

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

(3) For each action command o do:

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

Table 2
Markov localization—the basic algorithm.

the more recent exception of [39,48], 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 [74,111,139] 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 cases 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 [19]. 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 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 computer 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 differ significantly from zero.

In RHINO's localization algorithm, grid cells whose probability are smaller than a

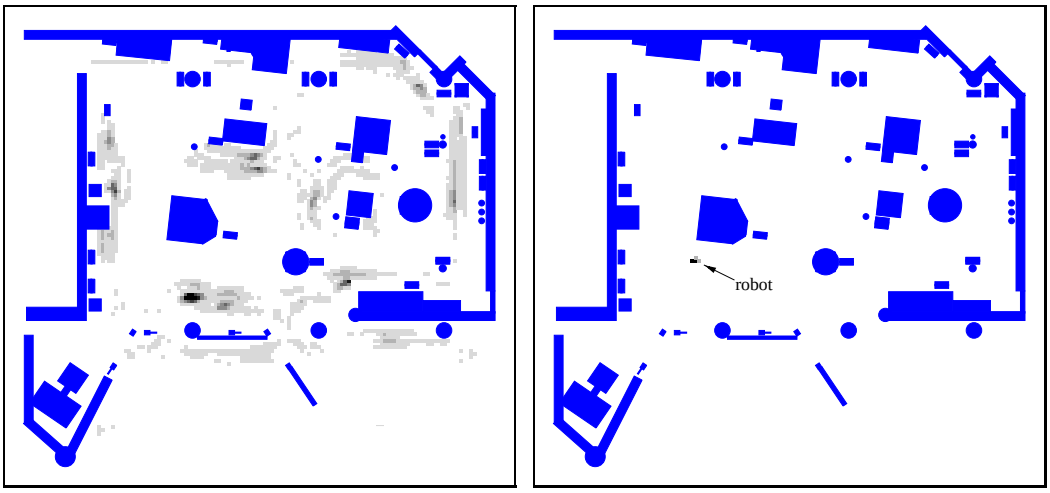

Fig. 7: Global localization in the Deutsches Museum Bonn. The left image shows the belief state after incorporating one laser scan. After incorporating a second scan, the robot uniquely determined its position (right).

---

threshold θ are not updated. Instead, they are represented by a single value, which uniformly represents the probability of all non-updated grid cells [18]. 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 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 believe 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 may be 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* [53] and works as follows. The *entropy* $H(P)$ of a distribution P is defined by [23]

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

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

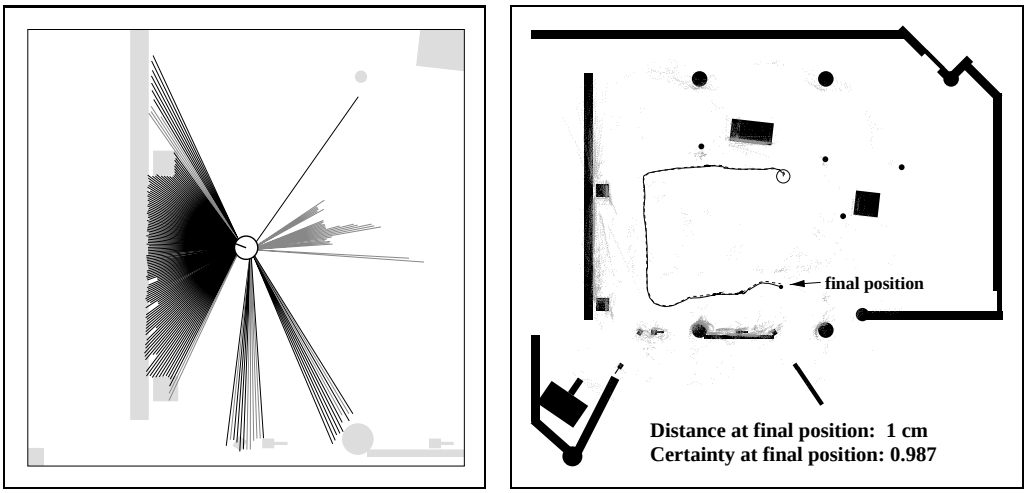

Fig. 8: Sensor measurements (black) selected by the entropy filter from a typical scan (left image) and endpoints of selected scans on a longer trajectory (right image)

---

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:

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

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

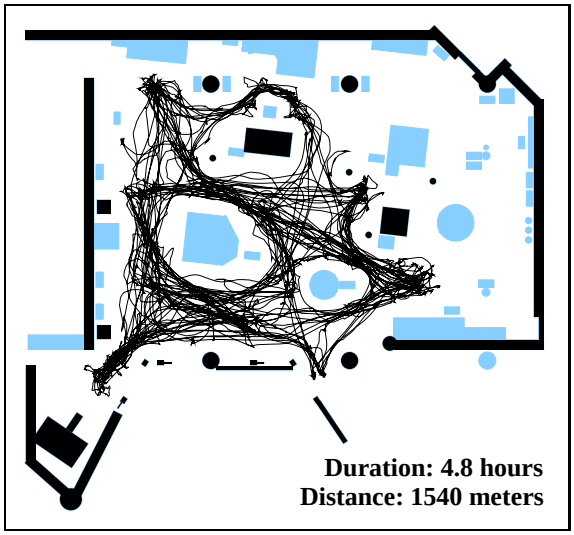

Fig. 9: One of the data sets used to evaluate the accuracy of RHINO's localization in densely populated environments. Here more than half of all sensor readings were corrupted by people; yet the robot managed to keep its average localization error below 10 cm.

---

successfully filtered out. We did observe, however, that the robot occasionally filtered out 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 that it never lost 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 [53]. One of the data sets, shown in Figure 9, contains data collected during 4.8 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 extended periods of time. The average localization error was found to be smaller than 10cm [53]. 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. To solve this problem we always incorporated a randomly chosen set of readings. A successor of the entropy gain

filter, which is better suited to proximity sensors and outperforms the entropy gain filter, is described in [53]. As discussed in more depth there, Markov localization combined with the entropy gain filter was able to accurately estimate the position of the robot throughout the entire deployment period, and the entropy filter played a crucial role in its success. Additional comparative experimental results, using the data obtained in the museum, can be found in [53].

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 and other objects not contained in the map. Thus, the inverse of the entropy gain filter is a filter that can detect people. This filter differs from many other approaches in the literature on people detection [72,76] in that it can find people who do *not* move, and it can do this 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 [10,41,106,147] 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 corresponds to the occupancy of a location $\langle x, y \rangle$ (in world coordinates). Here 1 stands for *occupied*, and 0 stands for *free*. 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 set of 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 $\langle x, y \rangle$ [106,147]. This

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

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

Table 3
Mapping—the basic algorithm.

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 efficiently.

3.2.1 Temporal Integration of Evidence

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

$$P(o^{(t)} \mid c_{xy}, \xi^{(t)}, d \backslash o^{(t)}) = P(o^{(t)} \mid \xi^{(t)}, c_{xy}) \tag{16}$$

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 the localization problem, it is assumed that the robot's location $\xi^{(t)}$ is known [1] ; henceforth, the estimation of $\xi^{(t)}$ will be omitted in the mathematical derivation. In our implementation, the maximum likelihood estimation

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

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

[1] See [150] 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.

location $\langle x, y \rangle$ is occupied given the data is given by

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

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 more compact update equation than in the real-valued case of position estimation [106]. In analogy to (19), the probability that a location $\langle x, y \rangle$ is free (unoccupied) is given by

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

To see, just replace every occurrence of c_{xy} by $\neg c_{xy}$ in Equation (19).

Dividing (19) by (20) yields the following expression, which is often referred to as the *odds* of $\langle x, y \rangle$ being occupied [116]:

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

it follows that the desired probability is given by

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

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 3. The probability $P(c_{xy} \mid o)$ is called the *inverse sensor model* (or *sensor interpretation*), whose description is subject to the next section.

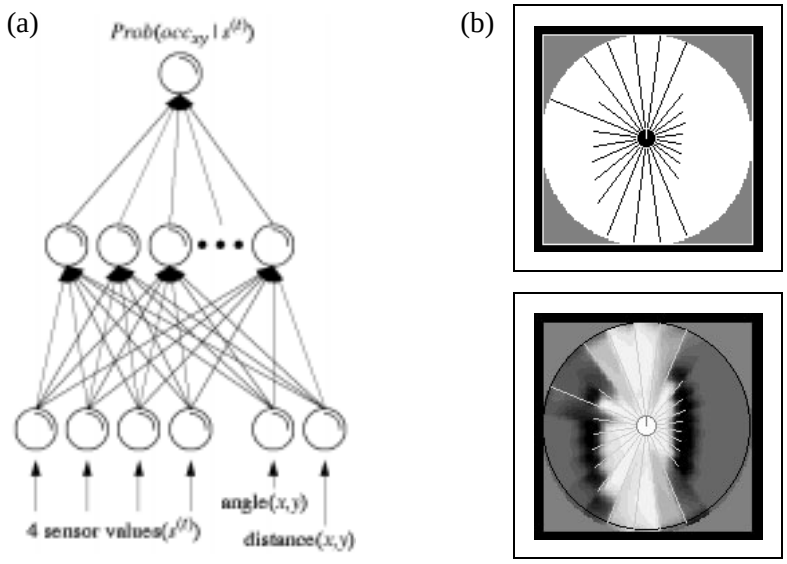

Fig. 10. (a) An artificial neural network maps sensor measurements to probabilities of occupancy. (b) An example sonar scan, along with the local map generated by the neural network. The darker a region, the more likely it is to be occupied.

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 [147], an artificial neural network is trained with Back-Propagation [65,123] 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 at random places in a known environment, and recording its sensor readings. 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 [100,104,147], which in RHINO's case is carried out using a simulator [147], 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 [59,60,98,143,147,150,152], 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

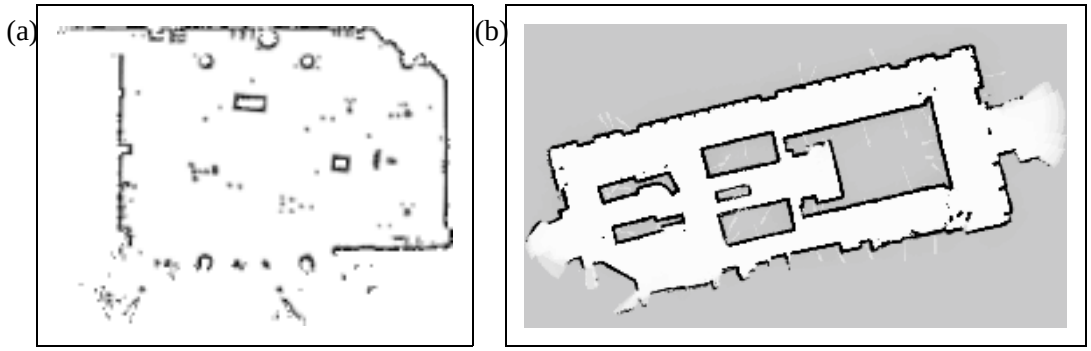

Fig. 11. Maps learned from scratch: (a) the Deutsches Museum Bonn, and (b) the Dinosaur Hall of Pittsburgh's Carnegie Museum of Natural History, built in preparation of the installation of a similar tour-guide robot. Both maps were acquired in less than an hour.

---

than one 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

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

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 navigation. The reader may notice that the integration rule (18) is inapplicable if different sensors detect different obstacles, which is the case for the specific sensor systems considered in this article.

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 make 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.