The Interactive Museum Tour-Guide Robot

Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hähnel,
Gerhard Lakemeyer[†], Dirk Schulz, Walter Steiner, and Sebastian Thrun[‡]

Computer Science Department III
University of Bonn
Bonn, Germany

[†]Computer Science Department
Aachen University of Technology
Aachen, Germany

[‡]School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Abstract

This paper describes the software architecture of an autonomous tour-guide/tutor robot. This robot was recently deployed in the "Deutsches Museum Bonn," were it guided hundreds of visitors through the museum during a six-day deployment period. The robot's control software integrates low-level probabilistic reasoning with high-level problem solving embedded in first order logic. A collection of software innovations, described in this paper, enabled the robot to navigate at high speeds through dense crowds, while reliably avoiding collisions with obstacles—some of which could not even be perceived. Also described in this paper is a user interface tailored towards non-expert users, which was essential for the robot's success in the museum. Based on these experiences, this paper argues that time is ripe for the development of AI-based commercial service robots that assist people in everyday life.

Introduction

Building autonomous robots that assist people in everyday life has been a long-standing goal of research in artificial intelligence and robotics. This paper describes an autonomous mobile robot, called RHINO, which has recently been deployed at the "Deutsches Museum" in Bonn, Germany. The robot's primary task was to provide interactive tours to visitors of the museum. In addition, the robot enabled people all around the world to establish a "virtual presence" in the museum, using a Web interface through which they could watch the robot operate and send it to specific target locations.

The application domain posed a variety of challenges, which we did not face in our previous research, carried out mostly in office-like buildings. The two primary challenges were (1) navigating safely and reliably through crowds, and (2) interacting with people in an intuitive and appealing way.

1. **Safe and reliable navigation.** It was of ultimate importance that the robot navigated at approximate walking speed, while not colliding with any of the various obstacles (exhibits, people). Three aspects made this problem difficult. First, RHINO often had to navigate through dense crowds of people (c.f. Figures 1 and 2). People often blocked virtually all sensors of the robot for extended durations of time. Second, various exhibits were "invisible" to the robot, that is, the robot could not perceive them with its sensors. This problem existed despite the fact that our robot possessed four state-of-the-art sensor systems (laser, sonar, infrared, and tactile). Invisible obstacles included glass cages put up to protect exhibits, metal bars at various heights, and metal plates on which exhibits were placed (c.f., the glass cage labeled "o1" and the control panel labeled "o2" in Figure 3). Navigating safely was particularly important since many of the exhibits in the museum were rather expensive and easily damaged by the 300 pound machine. Third, the configuration of the environment changed frequently. For example, the museum possessed a large number of stools, and people tended to leave them behind at random places, sometimes blocking entire passages. Museum visitors often tried to "trick" the robot into an area beyond its intended operational range, where unmodeled and invisible hazards existed, such as staircases. For the robot to be successful, it had to have the ability to quickly detect such situations and to find new paths to exhibits if possible.
2. **Intuitive and appealing user interaction.** Tutoring robots, by definition, interact with people. Visitors of the museum were between 2 and 80 years old. The average visitor had no prior exposure to robotics or AI and spent less than 15 minutes with the robot, in which he/she was unable to gain any technical understanding whatsoever. Thus, a key challenge was to design a robot that was "intuitive" and user friendly. This challenge included the design of easy-to-use interfaces (both for real visitors and for the Web), as well as finding mechanisms to communicate intent to people. RHINO's user interfaces were an important component in its practical success.

The specific application domain was chosen for two pri-

Fig. 1: RHINO gives a tour

Fig. 2: Interaction with people

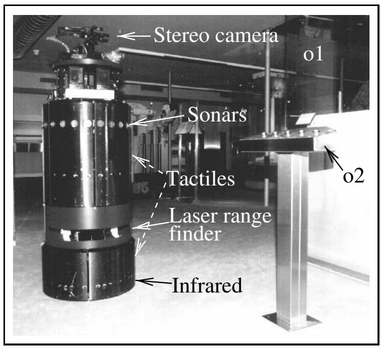

Fig. 3: The Robot and its sensors.

mary reasons. First, we believe that these two challenges, safe navigation and easy-to-use interfaces, are prototypical for a large range of upcoming service robot applications. Second, installing a robot in a museum gave us a unique and exciting opportunity to bridge the gap between academic research and the public.

This paper describes the key components of RHINO's software architecture. The software performs, in real-time, tasks such as collision avoidance, mapping, localization, path planning, mission planning, and user interface control. The overall software architecture consists of approximately 25 independent modules (processes), which were executed in parallel on three on-board PCs and three off-board SUN workstations, connected via a tetherless Ethernet bridge (Thrun *et al.* 1998a). The software modules communicate using TCX (Fedor 1993), a decentralized communication protocol for point-to-point socket communication.

RHINO's control software applies several design principles, the most important of which are:

1. **Probabilistic representations, reasoning, and learning.** Since perception is inherently inaccurate and incomplete, and since environments such as museums change frequently in rather unpredictable ways, RHINO pervasively employs probabilistic mechanisms to represent state and reason about change of state.
2. **Resource adaptability.** Several resource-intensive software components, such as the motion planner or the localization module, can adapt themselves to the available computational resources. The more processing time is available, the better and more accurate are the results.
3. **Distributed, asynchronous processing with decentralized decision making.** RHINO's software is highly distributed. There is no centralized clock to synchronize the different models, and the robot can function even if major parts of its software fail or are temporarily unavailable (e.g., due to radio link failure).

The remainder of this paper will describe those software modules that were most essential to RHINO's success. In particular, it will present the major navigation components and the interactive components of the robots.

Navigation

RHINO's navigation modules performed two basic functions: perception and control. The primary perceptual modules were concerned with localization (estimating a robot's location) and mapping (estimating the locations of the surrounding obstacles). The primary control components performed real-time collision avoidance, path planning, and mission planning.

Localization

Localization is the problem of estimating a robot's coordinates (also called: *pose* or *location*) 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. The problem of localization is generally regarded to be difficult (Cox 1991; Borenstein, Everett, & Feng 1996). It is specifically hard in densely populated domains where people may block the robot's sensors for extended periods of time.

RHINO employs a version of *Markov localization*, a method that has been employed successfully by several teams (Nourbakhsh, Powers, & Birchfield 1995; Simmons & Koenig 1995; Kaelbling, Cassandra, & Kurien 1996). RHINO utilizes a *metric* version of this approach (Burgard *et al.* 1996), in which poses are estimated in x-y-θ space.

Markov localization maintains a probabilistic belief as to where the robot currently is. Let $P(l)$ denote this belief, where l is a pose in x-y-θ space. Initially, when the pose of the robot is unknown, $P(l)$ is initialized by a uniform distribution. To update $P(l)$, Markov location employs two probabilistic models, a perceptual model and a motion model. The perceptual model, denoted $P(s \mid l)$, denotes the probability that the robot observes s when its pose is l. The motion model, denoted $P(l' \mid l, a)$, denotes the probability that the robot's pose is l' if it previously executed action a at location l.

Both models are used to update the robot's belief $P(l)$

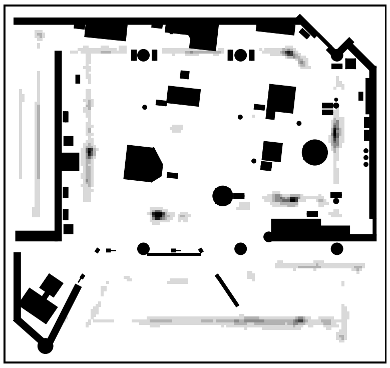

Fig. 4: Global localization. Obstacles are shown in black; probabilities $P(l)$ in gray.

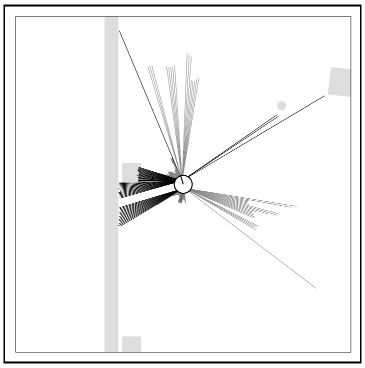

Fig. 5: Typical laser scan in the presence of people. The gray-shaded sensor beams correspond to people and are filtered out.

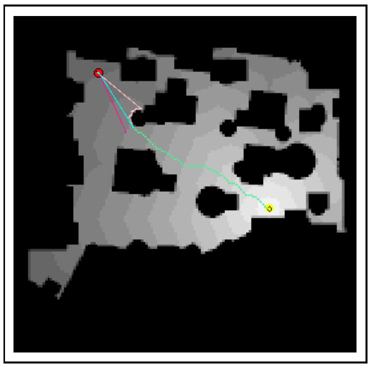

Fig. 6: RHINO's path planner: The grey-scale indicates the proximity of the goal.

when the robot senses, and when it moves, respectively. Suppose the robot just sensed s. Markov localization then updates $P(l)$ according to Bayes rule:

$$P(l \mid s) \quad = \quad \alpha \; P(s \mid l) \; P(l) \tag{1}$$

where α is a normalizer that ensures that the resulting probabilities sum up to one. When the robot moves, Markov localization updates $P(l)$ using the Theorem of total probability:

$$P(l') \quad = \quad \int P(l' \mid a, l) \; P(l) \; dl \tag{2}$$

Here a denotes an action command. These two update equations form the basis of Markov localization. Strictly speaking, they are only applicable if the environment meets a *conditional independence assumption* (Markov assumption), which specifies that the robot's pose is the *only* state therein. Put differently, Markov localization applies only to static environments.

Unfortunately, the standard Markov localization approach is prone to fail in densely populated environments, since those violate the underlying Markov assumption. In the museum, people frequently blocked the robot's sensors, as illustrated in Figure 1. Figuratively speaking, if people line up as a "wall" in front of the robot—which they often did—, the basic Markov localization paradigm makes the robot eventually believe that it is indeed in front of a wall.

To remedy this problem, RHINO employs an "entropy filter" (Fox *et al.* 1998b). This filter, which is applied to all proximity measurements individually, sorts measurements into two buckets: one that is assumed to contain all corrupted sensor readings, and one that is assumed to contain only authentic (non-corrupted) ones. To determine which sensor reading is corrupted and which one is not, the entropy filter measures the relative entropy of the belief state *before* and *after* incorporating a proximity measurement:

$$\Delta H(l, s) := \tag{3}$$

$$- \int_l P(l) \; \log P(l) \; dl + \int_l P(l \mid s) \log P(l \mid s) \; dl$$

Sensor readings that increase the robot's certainty ($\Delta H(l, s) > 0$) are assumed to be authentic. All other sensor readings are assumed to be corrupted and are therefore not incorporated into the robot's belief. In the museum, certainty filters reliably identified sensor readings that were corrupted by the presence of people, as long as the robot knew its approximate pose. Unfortunately, the entropy filter can prevent recovery once the robot looses its position entirely. To prevent this problem, our approach also incorporates a small number of randomly chosen sensor readings in addition to those selected by the entropy filter. See (Fox *et al.* 1998b) for an alternative solution to this problem.

In practice, our approach proved to be robust in the museum. RHINO's localization module was able (1) to globally localize the robot in the morning when the robot was switched on (see Figure 4 for a typical belief state during this procedure), and (2) to keep track of the robot's position reliably and accurately (see Figure 5). RHINO's localization error, measured over a set of 118 randomly selected reference locations, was consistently below 15 cm. This accuracy was essential for safe navigation. In the entire six-day deployment period, we are only aware of a single occasion in which the robot suffered a localization error larger than its 30cm safety margin, and this incident was preceded by partial loss of sensing. A comparison between our approach and plain Markov localization (i.e., our approach without the entropy filter) showed that the standard Markov algorithm would have lost track of the robot's position frequently, whenever large crowds surrounded the robot (Fox *et al.* 1998b).

Mapping

Mapping addresses the problem of determining the locations of the obstacles in global world-coordinates. In the museum domain, the problem of mapping was simplified significantly, since an accurate metric map of the museum was provided to the robot beforehand. However, the configuration of the museum changed frequently—people moved stools around and sometimes people blocked entire passages—, making necessary the revision of the map in real-time.

RHINO employed a probabilistic *occupancy grid algorithm* for modifying the initial map, a technique that was originally developed in the mid-eighties and since has been used successfully in many mobile robot applications (Moravec 1988; Elfes 1989). Occupancy grid techniques approximate the environment by a 2D grid, where each grid cell is annotated by a numerical "occupancy value" that represents the probability that this cell contains an obstacle.

In our approach (Thrun 1998), occupancy maps are generated using a Backpropagation-style network, which maps sensor measurements to local occupancy maps. The network is trained off-line, using labeled data obtained in environments where the exact locations of all obstacles are known. After training, the network generates conditional probability estimates for occupancy, denoted $P(o_{x,y} \mid s)$, for grid cells $\langle x, y \rangle$ that lie in the sensors' perceptual range. Figure 7a shows an example sonar scan, taken in a hallway, along with the local map. The shading encodes the likelihood of occupancy: The darker a location, the more likely it is to be occupied. As is easy to be seen, the two walls are detected with high certainty, as is the free space in between. Behind the walls, however, the network outputs 0.5, to indicate maximum uncertainty. Local maps are integrated over time using Bayes rule and likelihood ratios, as described in (Moravec 1988; Thrun 1998):

$$P(o_{x,y} \mid s^1, s^2, \ldots, s^T) = 1 - \left(1 + \frac{P(o)}{1-P(o)} \prod_\tau \frac{P(o_{x,y} \mid s^\tau)}{1-P(o_{x,y} | s^\tau)} \frac{1-P(o)}{P(o)}\right)^{-1} \quad (4)$$

Here s^τ denotes the sensor reading at time τ, and $P(o)$ denotes the *prior* probability for occupancy.

Figure 7 shows an example map, in which a narrow passage (upper left) has been blocked, forcing the robot to take a detour. When RHINO reaches a tour item, it reset its map to the original, pre-supplied one. This strategy ensures that passages, once blocked, will not be avoided indefinitely. The ability to modify the map on-the-fly was essential for RHINO's success. In some occasions, lack thereof would have caused the robot to be "stuck" for extended periods of time. RHINO frequently surprised people by successfully carrying out its mission even in the presence of major, permanent obstacles.

Collision Avoidance with "Invisible" Obstacles

RHINO's collision avoidance protects the robot from colliding with obstacles—exhibits and humans alike. It does this by controlling the actual motion direction and the speed of the robot, based on sensor input and based on a "target location" prescribed by the motion planner.

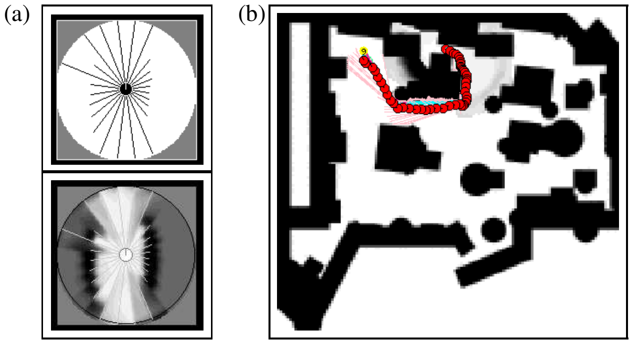

Fig. 7: (a) Sonar scan and local map, generated by the neural network. (b) This map is partially pre-supplied, partially learned. Here the motion planner chooses to take a detour to avoid a blocked passage on the top of this figure.

RHINO employed an extension of the dynamic window algorithm (DWA) (Fox, Burgard, & Thrun 1997; Fox *et al.* 1998a). In a nutshell, DWA controls—four times a second—the *translational* and *rotational* velocity of the robot. It does this in accordance with several hard and soft constraints. Hard constraints restrict the space of velocities. They are imposed to avoid velocities that, if chosen, would inevitably lead to a collision, taking into account the robot's inertia and torque limits. Soft constraints, on the other hand, express preferences in the space of control. DWA employs essentially two soft constraints, one which encourages the robot to make progress towards its target location, and one which seeks to maximize the robot's velocity (thereby favoring motion directions towards uncluttered space). Together, these soft constraints lead to a behavior that makes the robot navigate smoothly around obstacles while making progress towards its target whenever possible. In particular, the second constraint ensures that the robot moves at high speeds whenever the situation permits.

The complicating factor in the museum was the presence of the various "invisible" obstacles, as described in the introduction to this article. The basic DWA, just like any other sensor-based collision avoidance approach, does not allow for preventing collisions with such obstacles. To avoid collisions with those, the robot had to consult its map. Of course, obstacles in the map are specified in world coordinates, whereas DWA requires the location of those obstacles in robo-centric coordinates. Thus, we extended the DWA approach to use the maximum likelihood position estimate $l^* = \text{argmax}_l\, P(l)$ produced by RHINO's localization module. Given the robot's location in the map, the μDWA algorithm (Fox *et al.* 1998a) generates "virtual" proximity measurements and integrates them with the "real" proximity measurements, obtained from the robot's various sensors (tactile, infrared, sonar, laser). This extension of the dynamic window approach proved to be highly effective in the museum. Figure 10 shows a trajectory of the robot travelled during the opening hours of the museum. The length of this trajectory is 1.6km. Apparently, the robot safely avoids "invisible" objects which are marked by the grey-shaded areas. The same mechanism was used to limit the operational range of the robot. With it, the robot successfully withstood all attacks by visitors to force it into unmapped terrain. The collision avoidance routine was also instrumental in quickly finding ways around unanticipated obstacles, such as humans that stepped in the robot's path, or stools left behind by visitors.

In order to deal with situations in which RHINO's localization module assigns high probability to multiple poses, we recently developed a more conservative strategy, which was not ready at the time of the museum installation, but which is now part of μDWA (Fox *et al.* 1998a). This approach generates "virtual" sensor readings that are with 99% probability *shorter* than the actual distances to the nearest invisible obstacles. Let s be a proximity measurement that one would expect if all invisible obstacles were actually detectable. Then

$$P(s) \quad = \quad \int P(s \mid l) \; P(l) \; dl. \tag{5}$$

is the probability distribution over all s that is induced by the map and the uncertainty in the location l. Suppose the robot's virtual sensor were set to one such value, say s^*. With probability

$$\mu(s^*) \quad := \quad \int_{s>s^*} P(s) \; ds \tag{6}$$

this value s^* will be *smaller* than the real value, that is, with probability μ_{s^*}, the proximity returned by the virtual sensor will be underestimating the true distance to an invisible obstacle, and the robot will be safe. In our implementation of μDWA, s^* is chosen so that $\mu(s^*) \geq .99$, that is, the virtual sensor measurements prevent collisions with invisible obstacles with 99% probability. Empirical results presented elsewhere demonstrate that this extension yields safer control in situations with high uncertainty.

Path Planning

RHINO's path planner generates paths from one exhibit to another. In the museum, such paths could not be predetermined in advance, since the environment was highly dynamic and the map changed continuously. This made it imperative that the robot adapted its path to the situation on-the-fly.

RHINO's path planner is based on *value iteration*, a popular dynamic programming/reinforcement learning algorithm (Bellman 1957; Howard 1960). Value iteration computes values $V_{x,y}$ for each grid cell $\langle x, y \rangle$. Initially, the grid cell containing the goal is set to 0, and all others are set to ∞. Value iteration then updates the values of all unoccupied grid cells by the value of their best unoccupied neighbor plus by the costs of moving there. After convergence, each value $V_{x,y}$ corresponds to the distance between $\langle x, y \rangle$ and the goal location. Figure 6 shows a value function after convergence. Here the goal is located at the bottom right (white), the robot's location is shown on the top left, and the shading of the space indicates the values $V_{x,y}$. Notice how the values "spread" through the free-space. After convergence, steepest decent in the value function leads to the shortest path to the goal.

As described in more detail in (Thrun 1998), RHINO's path planner employs two additional mechanisms, aimed to increase its run-time efficiency:

1. It uses a bounding box technique to focus computation on regions where it matters.
2. It uses an algorithm similar to value iteration, to identify areas that require re-planning when the map changes. Such changes are often local, leading to a high degree of re-use when planning in dynamic environments.

Paths generated by the path planner are not executed directly; instead, they are passed on to the collision avoidance routine, using visibility considerations to generate intermediate "target locations." When computing these target locations, paths are also post-processed to maximize the robot's side-clearance.

RHINO's path planner is an *any-time* algorithm, i.e., it returns an answer at any time, whenever needed (Dean & Boddy 1988). As a pleasing consequence, the robot never halts to wait for its planner to generate a plan; instead, it moves continuously—unless, of course, it explains an exhibit.

In the museum, running value iteration to completion and finding the *optimal* path usually took less than one second. The path planner proved to be highly effective in adapting to new situations. Figure 7 shows an example, in which a passage is blocked by stools, forcing the robot and the people following it to take a detour. All these decisions are made in real-time, and people usually do not notice the change in venue.

Task Planning

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

Fig. 8: RHINO's on-board user interface: A screen and four buttons.

Fig. 9: On-line image page for the Web.

Fig. 10: Path of the robot during a single 4.5 hour run (1.6 km).

RHINO's task planner uses GOLOG (Levesque *et al.* 1997), which is an extension of the situation calculus (McCarthy 1968). GOLOG is a language for specifying complex actions using structures like if-then-else or recursive procedures. Based on these specifications, GOLOG generates, in response to a user request, a sequence of elementary (symbolic) actions which provably fulfil this request. To achieve robust and coherent behavior, and to allow the correction of possible errors of the underlying components, we have developed GOLEX. GOLEX complements GOLOG by supplying a run-time component that turns linear plans constructed by GOLOG into (1) hierarchical and (2) conditional plans, and (3) that also monitors their execution. None of these extensions are universal (they are in fact quite limited), but they are all necessary in the context of mobile robot control.

1. **Hierarchical plans.** GOLEX converts each GOLOG action into a pre-specified sequence of elementary commands for RHINO's lower level software. For example, the GOLOG action "move to exhibit number five" is translated into a sequence of robot motion commands ("move to a position next to exhibit number five," "turn towards exhibit number five"), pan/tilt control commands ("track exhibit number five"), and graphical and acoustical display commands ("announce exhibit number five"). The advantage of the hierarchical decomposition is an enormous reduction in the complexity of the task planning problem.
2. **Conditional plans.** Some of GOLOG's actions are translated into pre-specified conditional plans, i.e., plans that are conditioned on the outcome of sensing actions. In GOLEX, conditional plans must always succeed; however, the specific sequence of sub-actions may vary. For example, the action "explain exhibit number five" is translated into a conditional plan that incorporates user feedback (e.g. the user chooses the level of detail in the robot's explanation).
3. **Execution monitor.** GOLEX monitors the execution of its plans. Using time-out mechanisms, it is able to invoke new actions and to plan pro-actively. For example, in the museum, people often pushed no button when they were asked to make a decision. In such situations, GOLEX's execution monitor made a default decision (e.g. it made the robot move to the next tour item) after waiting for a certain amount of time.

GOLEX provided the necessary "glue" between the high-level and the rest of the robot software, thereby bridging the gap between AI-style symbolic reasoning and numerical, sub-symbolic navigation software.

User Interaction

An important aspect of the tour-guide robot is its interactive component. User interfaces are generally of great importance for robots that interact with people. In application domains such as museums, the user interfaces must be intuitive, so that untrained, non-technical users can operate the system without instruction. It must also be appealing, so that people feel attracted to the robot and participate in a tour. While navigation has been studied extensively in the mobile robotics literature, the similarly important issue of human robot interaction has received considerably little attention. RHINO possesses two primary user interfaces, one on-board interface for interacting with people directly, and one on the Web.

On-board Interface

The on-board interface is a mixed-media interface that integrates text, graphics, pre-recorded speech, and sound. When visitors approach the robot, they can choose a tour or, alternatively, listen to a brief, pre-recorded explanation (the "help text"). They indicate their choice by pressing one of four colored buttons shown in Figure 8. When RHINO moves towards an exhibit, it displays an announcement on its screen. It also indicates the direction of its intended destination by continually pointing the camera in the direction of the next exhibit. At each exhibit, the robot plays a brief

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
number of requests	2,400
success rate	99.75%

Table 1: Survey of key results.

pre-recorded verbal explanation. Users can then request further information about the specific exhibit or, alternatively, make the robot proceed. When a tour is finished, the robot returns to a pre-defined starting position in the entrance area where it awaits new visitors.

An important aspect of RHINO's interactive component is the robot's physical reaction to people. RHINO uses body and head motion and sound to express *intent* and *dissatisfaction*. As mentioned above, RHINO's camera head is used to communicate the intended motion direction. In addition, RHINO uses a modified version of the entropy filter (a probabilistic *novelty filter* (Fox *et al.* 1998b)) to detect people or other unexpected obstacles. If such obstacles block the robot's path, it uses its horn to indicate its "dissatisfaction." In the museum, people usually cleared the robot's path once they heard the horn. Some even blocked the robot's path intentionally in expectation of an acoustic "reward."

Web Interface

RHINO's Web interface consists of a collection of Web pages, some of which are interactive, others provide background information. In addition to previous Web interfaces (see e.g., (Simmons *et al.* 1997)), which basically rely on client-pull/server-push mechanisms, RHINO's interface also offers Java applets for instant update of information (both state and intent) as the robot moves. One of the main pages of the Web interface is shown in Figure 9. This page enables Web users to observe the robot's operation on-line. The camera image on the left is obtained with one of RHINO's cameras. The right image is taken with one of two fixed, wall-mounted cameras. The center display shows a map of the robot's environment from a bird's eye perspective, along with the actual location of the robot. The bottom portion of this page contains information about RHINO's current actions. When RHINO is explaining an exhibit, this area displays information about the exhibit, including hyperlinks to more detailed background information. Information may be updated synchronously in user-specified time intervals. Alternatively, certain information (such as the robot's position in its map) can be visualized smoothly, using a Java applet that "simulates" the robot.

Statistics

RHINO was deployed for a period of six days, in which it operated for approximately 47 hours without any significant downtime (see Table 1). Over this period of time, the robot travelled approximately 18.6km. More than 2,000 real visitors and over 600 "virtual" Web-based visitors were guided by RHINO. The robot fulfilled 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, 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 the robot's pose 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%.

Discussion

This paper described the major components of a software architecture of a successfully deployed mobile robot. The robot's task was to provide interactive tours to visitors of a museum. In a six-day deployment period in the Deutsches Museum Bonn, the robot gave tours to a large number of visitors, safely finding its way through dense crowds. RHINO's software integrates a distributed navigation package and modules dedicated to human robot interaction. While the approach is mostly based on existing AI methods of various flavors, it contains several innovations that were specifically developed to cope with environments as complex and as dynamic as the museum. These include a method for localization in dense crowds, an approach for collision avoidance with "invisible" obstacles, a method (GOLEX) that glues together first order logic and numerical robot control, and a new, task-specific user interface. Among the various design principles that came to bear, three stick out as the most important ones. We strongly believe that the use of probabilistic representations for on-line state estimation and learning, the use of resource-adaptive algorithms for state estimation and planning, and the decentralized and distributed nature of the control scheme were all essential to produce the level of robustness and reliability demonstrated here.

What lessons did we learn? From hours of watching the robot operate, we concluded that interaction with people is an essential element in the success of future robots of this and similar kinds. We believe that there is a real need for research in the area of human robot interaction. We are also interested in methods accelerating the installation procedure. For example, one of us spent a week constructing a map of the museum by hand. Based on work by (Lu & Milios 1997; Gutmann & Schlegel 1996; Thrun, Fox, & Burgard 1998), we now have clear evidence that this task can be automated (Thrun *et al.* 1998b), and the time for acquiring such a map from scratch can be reduced to a few hours.

We see a significant potential to leapfrog much of the technology developed in this and similar projects (e.g., (King & Weiman 1990)) into other service robot applications, such as applications in the areas of health care, cleaning, inspection, surveillance, recreation etc. We conjecture that time is ripe for developing AI-based commercial service robots that assist people in everyday life.

References

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

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

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

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

Dean, T. L., and Boddy, M. 1988. An analysis of time-dependent planning. In *Proceeding of Seventh National Conference on Artificial Intelligence AAAI-92*, 49–54. Menlo Park, CA: AAAI.

Elfes, A. 1989. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. Ph.D. Dissertation, ECE, CMU.

Fedor, C. 1993. *TCX. An interprocess communication system for building robotic architectures. Programmer's guide to version 10.xx.* CMU.

Fox, D.; Burgard, W.; Thrun, S.; and Cremers, A. 1998a. A hybrid collision avoidance method for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Fox, D.; Burgard, W.; Thrun, S.; and Cremers, A. 1998b. Position estimation for mobile robots in dynamic environments. In *Proceedings of AAAI-98*. AAAI Press/The MIT Press.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation* 4(1).

Gutmann, J.-S., and Schlegel, C. 1996. 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.

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

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

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

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

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

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

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

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

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

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

Simmons, R.; Goodwin, R.; Haigh, K.; Koenig, S.; and O'Sullivan, J. 1997. A layered architecture for office delivery robots. In *Proceedings of the First International Conference on Autonomous Agents*.

Thrun, S.; Bücken, A.; Burgard, W.; Fox, D.; Fröhlinghaus, T.; Hennig, D.; Hofmann, T.; Krell, M.; and Schimdt, T. 1998a. Map learning and high-speed navigation in RHINO. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *AI-based Mobile Robots: Case studies of successful robot systems*. Cambridge, MA: MIT Press.

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

Thrun, S.; Fox, D.; and Burgard, W. 1998. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots (joint issue)*. to appear.

Thrun, S. 1998. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*. to appear.