Integrating Active Localization into High-level Robot Control Systems

Michael Beetz, Wolfram Burgard, Dieter Fox, Armin B. Cremers

Dept. of Computer Science III, University of Bonn,
Römerstr. 164, D-53117 Bonn, Germany
{beetz,wolfram,fox,abc}@cs.uni-bonn.de

Abstract

High-level control systems are designed to enable mobile robots to successfully perform complex missions such as office delivery and surveillance tasks. For that purpose they have to control, coordinate, and monitor different kinds of subtasks like navigation, manipulation, and perception. An important aspect of the effectiveness of high-level control systems is the ability to cope with failures that occur during the execution of such subtasks. In this paper we focus on the particular subtask of estimating the position of the robot and show how to achieve its robust integration into the high-level control system. The principle of this integration is to monitor the certainty of the position estimation and to autonomously relocalize the robot whenever the uncertainty grows too large. We present a localization approach which accurately and efficiently keeps track of the robot's position. Furthermore, it provides a measure for detecting localization failures and it is able to autonomously relocalize the robot in such situations. In addition to this, we introduce structured reactive plans, which can be interrupted by such active localization processes at any point in time and allow the robot to complete its mission afterwards. Our method has been implemented and shown to be robust in long-term experiments involving a typical office delivery scenario.

Key words: Robot Position Estimation, High-level Robot Control, Autonomous Service Robots

1 Introduction

To successfully perform complex missions such as office delivery and surveillance tasks, mobile robots must reliably fulfill and coordinate different kinds of subtasks like navigation, manipulation, and perception. Many service robots employ high-level control systems, which control, coordinate, and monitor these subtasks during execution to ensure the robot's operational effectiveness [TBB+98,SGH+,KMRS97]. An important aspect of this effectiveness is the ability of the high-level control system to cope with failures that occur during the execution of such subtasks. In this paper we focus on a particular aspect of such robot control systems, namely the interplay between localization, the estimation of the position of the robot within its environment, and the high-level control system. Since possession of accurate knowledge of the robot's location is one of the fundamental preconditions for successful robot navigation, much research has concentrated on position estimation and has produced impressively reliable localization methods. However, due to environmental conditions and the inherent limitations of sensors, localization methods applied in populated environments that are not specifically engineered for the robot will inevitably have occasional failures. As a consequence, to reach a high degree of autonomy, mobile robots must be able to master even those situations in which they do not know where they are. Despite its importance, surprisingly little attention has been paid to the problem of how to deal with situations in which the robot loses track of its position during the execution of its missions.

Our approach to handling these situations is to interrupt the mission-specific activities, autonomously relocalize the robot, and then continue with the mission. To accomplish this behavior, the following requirements have to be met by the control system:

Cognizant position estimation: To allow for the detection of situations in which the position has been lost, the localization method must provide a measure for the degree of certainty in the position estimation.

Active localization: In order to relocalize the robot if the position has been lost, the position estimation procedure must be able to globally localize the robot from scratch. Since most environments contain symmetries, this can in most cases only be achieved by *actively* guiding the robot to places where it can uniquely determine its position [BFT97].

Primary activities and policies: The control system must distinguish and concurrently execute *primary activities* (the actions taken to accomplish the robot's mission) and *policies* (in which those conditions necessary for the successful execution of the primary activities are both monitored and maintained). Primary activities include for example the navigation to places where objects are to be picked up and delivered. One of the policies monitors the localization system and invokes active localization whenever necessary.

Mission interruptability and continuation: Many primary activities have to be suspended immediately on localization failure because they cannot complete successfully without knowing the robot's position. Thus primary activities have to handle interrupts and, due to the possible side-effects of active relocalization, these activities have to make suitable preparations for their successful continuation after reactivation.

Our method for integrating localization into robot control systems meets all these requirements. To meet the first two requirements, we apply an extended version of Markov localization to estimate the robot's position. This technique has recently been proposed and implemented with considerable success by several groups [NPB95,SK95,KCK96,BFHS96,HK96]. Markov localization maintains a probability density for all possible locations in the environment, thus permitting the computation of the overall certainty in the position estimate. This representation is well suited to representing degree-of-belief and to the handling of ambiguous situations, which is particularly important during global localization. While Markov localization is passive, we propose an extension of this technique which is able take over the control of the robot so as to minimize the uncertainty in the position estimation.

To address the third requirement, the high-level control system is implemented in RPL (Reactive Plan Language), a notation for specifying *structured reactive plans (SRPs)*. SRPs provide a means of synchronizing concurrent activities, monitoring aspects of the robot and its environment, locally recovering from execution failures, and avoiding the transmission of inconsistent commands to sensors and effectors.

To achieve mission interruptability and continuation, primary actions include specifications of both (sub-)actions that have to be performed whenever the primary actions are interrupted and also of the (sub-)actions that have to be performed after such an interruption. For example, the navigation process, responsible for computing the next target points and monitoring the progress towards the given target, generates a new plan whenever it is reactivated after a relocalization of the robot.

We propose a control paradigm that causes the robot to exhibit the following behavior: as long as the robot knows its position sufficiently well, it tracks its position passively while accomplishing its tasks. Whenever the robot is switched on or loses track of its position, it starts to actively (re-)localize itself before it begins or continues its mission.

In the field of robot control systems, other approaches deal with the problem of selecting appropriate actions under position uncertainty include [SGH$^+$,GCJK97,KCK96]. Simmons et al. [SGH$^+$] choose the optimal navigation plan with respect to the most likely position of the robot. In contrast to this method, our approach takes into account the degree of uncertainty in the position estimation and considers dedicated actions for reducing uncertainty in position estimation. Guzzoni et al. [GCJK97] detect situations in which the position of the robot has been lost. Whereas their approach relies on the user to specify the robot's current position to recover from such situations, our method is able to autonomously relocalize the robot. Kaelbling et al. [KCK96] propose a decision-theoretic approach that takes uncertainty

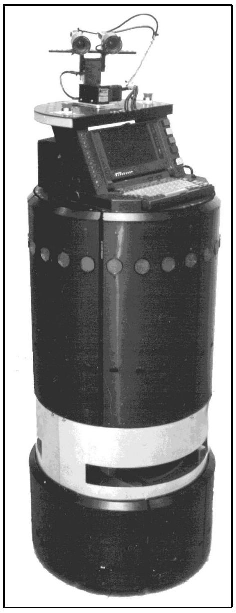

Fig. 1: The mobile robot Rhino, an RWI B21

and actions for relocalization into account. They do not distinguish between primary actions and localization actions and always choose the actions with the highest utility with respect to the belief state of the robot. While this technique is optimal with respect to the modeled quantities and the planning horizon, its applicability is limited due to the computational complexity of the approach. To be practical, our approach only considers the most likely position for planning. Whenever the certainty of the position estimation falls below a specified threshold, active relocalization is performed. Although this might not lead to the optimal behavior, extended experiments showed the reliability of this approach. We furthermore expirienced, that possible additional costs are negligible during long-term operation, because our localization technique failed very rarely.

The remainder of this article is organized as follows. In the next section we sketch our approach to geometric Markov localization and its extension to active localization. Then we describe the implementation of primary activities using RPL. The integration of the active localization method into our high-level control system has been tested in an office delivery application using RHINO [BBC+95,TBB+98] (see Figure 1), an RWI B21 mobile robot.

2 The Markov Localization System

Localization is one of the fundamental problems in mobile robot navigation. So far research on localization has mainly focused on methods for keeping track of the robot's position (see [BEF96] for a comprehensive overview). Most of these methods, however, are not able to globally localize the robot and do not provide a means for detecting situations in which the robot loses track of its position. To overcome these deficiencies, recently different variants of Markov localization methods have been proposed for globally estimating the position of a robot [NPB95,SK95,KCK96,BFHS96,HK96].

Markov localization techniques combine several advantages in an elegant way. They are able to deal with imperfect information (sensor noise and approximate world models) and they are able to represent ambiguous situations, which occur frequently during *global* position estimation [BFHS96]. In addition to this, Markov localization allows for an *active* extension in a mathematically consistent way (see Section 2.2). Markov localization techniques have been shown to be extremely robust and reliable for determining and keeping track of the robot's position even in populated environments. One successful application example is the "Museum tour-guide project", where our mobile Robot *RHINO* was deployed over six days as a tour-guide in the "*Deutsches Museum Bonn*" [BCF+97].

The key idea of Markov localization techniques is to maintain a density P over the set L of all possible positions of the robot in its environment. This density P is updated whenever the robot moves or new sensory input is obtained. Whenever the robot moves and a movement a is measured, the following general formula coming from the field of Markov chains is applied:

$$P(l) \longleftarrow \sum_{\tilde{l}} p(l \mid a \wedge \tilde{l}) \cdot P(\tilde{l}) \tag{1}$$

Here, $p(l \mid a \wedge \tilde{l})$ models the probability that the robot is at location l after having executed action a at location $\tilde{l}$. On the other hand, if sensory input s is received, the well-known Bayesian update formula is used:

$$P(l) \longleftarrow \frac{P(s \mid l)\ P(l)}{P(s)} \tag{2}$$

where $P(s)$ is a normalizer ensuring that the probabilities over all l sum up to 1.

In the remainder of this section we present an extended approach to Markov localization which is able (1) to efficiently keep track of the robot's position, (2) to determine when the tracking has failed, and (3) to actively relocalize the robot in case of such a failure. All these features are basic preconditions for mobile robots with a high degree of autonomy.

2.1 Grid-based Markov Localization

To achieve the necessary level of accuracy in position estimation, which is essential for performing various tasks, we use a fine-grained grid-based representation of the position probability density P [BFHS96]. While the resolution of robot orientation is typically of the order of $1°$ to $2°$, longitudinal resolution is often as small as 10 to 15cm. In contrast with other variants of Markov localization techniques [NPB95,SK95,KCK96,HK96] using topological discretizations and a resolution of $90°$ in orientation, our grid-based approach provides position estimates with an accuracy of a few centimeters or degrees. Whereas the coarse discretization of the topological approaches requires the definition of abstract features such as openings, doorways or other types of landmarks, our high resolution grids allow to integrate raw (proximity) sensor readings.

For a mid-size environment of size $30 \times 30\text{m}^2$ and an angular resolution of $2°$ the state space L consists of $7,200,000$ states. To deal with such huge state spaces in real-time, which is essential for fast position position estimation, we apply different optimization techniques. First, we use a fast sensor model allowing to compute the sensing probability $P(s \mid l)$ by simple look-up operations [BFH97]. The second optimization approach is a technique for a *selective* update of the grid. The key idea of this approach is to exclude unlikely cells in P from being updated. For this purpose, we introduce a threshold[1] θ and approximate $P(s \mid l)$ for cells with $P(l) < \theta$ by $\tilde{P}(s)$ which is the average or a priori probability of measuring the feature s given a uniform distribution over all possible locations. This leads us to the following update rule:

$$P(l) \longleftarrow \begin{cases} \frac{P(s|l)}{P(s)} \cdot P(l) \text{ if } P(l) > \theta \\ \frac{\tilde{P}(s)}{P(s)} \cdot P(l) \quad \text{otherwise} \end{cases} \tag{3}$$

Please note that $\tilde{P}(s)$ differs from $P(s)$, which is the probability of measuring s given the current belief state P of the robot. Obviously, this update rule is equivalent to

$$P(l) \longleftarrow \begin{cases} \alpha \cdot \frac{P(s|l)}{\tilde{P}(s)} \cdot P(l) \text{ if } P(l) > \theta \\ \alpha \cdot P(l) \qquad\quad \text{otherwise} \end{cases} \tag{4}$$

where α is a normalizer, ensuring that all $P(l)$ sum up to one.

[1] In our current implementation θ is set to 1% of the a priori position probability.

According to this formula, the unlikely cells don't have to be updated on incoming sensor data. They only have to be changed whenever P has to be normalized or the robot moves. To speed up even these operations we partition L and proceed as follows. Each part of this partition is excluded from updates, as soon as all probabilities in it are less than θ. For each deactivated part π, we store the accumulated normalization factor α_π which is the product of all normalization factors α computed since π was deactivated. In addition to this we store the accumulated movement a_π since deactivation. Whenever α_π exceeds 1, then α_π and a_π are applied to π. After that, π is updated again. In our current implementation we partition L such that each part π_α consists of all locations with equal orientation α. In extensive experimental tests we did not observe evidence that these modifications impact the robot's behavior in any noticeable way. During global localization, the certainty of the position estimation increases and the density typically concentrates on the real position of the robot. If the position of the robot is determined, each sensor scan is processed in less than .1 seconds using a 200MHz Intel PentiumPro.

While our description of Markov localization is brief, it is important that the reader grasps the essentials of the approach. The robot maintains a belief distribution P which is updated upon robot motion and upon the arrival of sensor data. This probabilistic representation is well-suited for mobile robot localization due to its ability to handle ambiguities and to represent degree-of-belief. For typical navigation tasks the probability of the most likely position is generally the most useful measure for the certainty of the position estimation during position tracking. Using our selective updating mechanism, even large state spaces can be efficiently updated without losing the desired features of the overall approach. Thus it combines the benefits of approaches to both: position tracking (low computational complexity) and global (re-)localization of the robot.

2.2 Active Localization

While our technique to localization has proved to be very robust, it is passive in the sense that it does not include a means of controling a mobile robot in order to globally localize it if its position is not known. Due to inherent symmetries in typical environments, different places often appear the same to the robot's sensors. This generally results in several local maxima in the position distribution P. In [BFT97] we demonstrated that actively guiding the robot to places that help it to distinguish these local maxima can significantly increase the efficiency of the localization process. For active localization techniques, an appropriate measure for the uncertainty in the robot position is given by the entropy of the belief P:

$$H_P = -\sum_{l \in L} P(l) \cdot \log(P(l)). \tag{5}$$

$H_P = 0$ if the position is known with certainty so that P is Dirac distributed whereas H_P is maximal if the robot is completely uncertain and P is uniformly distributed. The general principle of our approach to active localization is to select those actions which minimize the expected future entropy of P.

Let $P_a(l)$ denote the belief of being at position l after executing motion a. $P_a(l)$ can easily be computed from $P(l)$ using the Markov positioning update rule (1). The expected entropy, conditioned on the action, can then be expressed by the following term:

$$E_a[H_P] = -\sum_{l \in L} \sum_{s \in S} P(s \mid l) P_a(l) \cdot \log[P(s \mid l) P_a(l) P(s)^{-1}] \tag{6}$$

where S is the set of features detectable by the sensors. This equation is obtained from the definition of the entropy by integrating over all possible sensor values $s \in S$, weighted by their likelihood, and by applying the update rule (2).

At first glance, one might use simple motor control actions (such as "move 1m forward") as basic actions when determining where to move so as to best position the robot. However, only considering the next single motor command is often insufficient. For example, a robot might have to move to a remote room in order to uniquely determine its location, which might involve a long sequence of individual motor commands. For this reason, we have chosen to consider arbitrary target points as atomic actions in our approach. Target points are specified relative to the current robot location, not in absolute coordinates. For example, an action $a = move(12\text{m}, 2\text{m})$ will make the robot move to a location 12 meter ahead and 2 meters to the left, relative to its current location and heading direction. Since the costs of reaching a target point can differ substantially depending on the time-of-travel, they also have to be taken into account.

The remainder of this section specifies the computation of the costs, the cost-optimal path, and demonstrates how to incorporate costs into action selection.

Occupancy probabilities: Our approach rests on the assumption that a map of the environment is available, which specifies which point l is occupied and which one is not. Let $P_{occ}(l)$ denote the probability that location l is blocked by an obstacle. The robot has to compute the probability that a target point a is occupied. Recall that the robot does not know its precise location. Thus, it must estimate the probability $P_{occ}(a)$ that a target point a is occupied. Simple geometric considerations permit the "translation" from $P_{occ}(l)$ (in real-world coordinates) to $P_{occ}(a)$ (in robot coordinates):

$$P_{occ}(a) = \sum_l P(l)\, P_{occ}(f_a(l)) \tag{7}$$

Here $f_a(l)$ is a simple coordinate transformation, which expresses the real-world coordinates of the point a, assuming that the robot is at l. In essence, Eq. (7) translates, for any l, the point a into real-world coordinates $f_a(l)$, then considers the occupancy $P_{occ}(f_a(l))$ of this point. The expected occupancy is then obtained by averaging over all locations l, weighted by the robot's subjective belief of actually being there. The result is the expected occupancy of a point a relative to the robot.

Costs and cost-optimal paths: Based on P_{occ}, the expected path length and the cost-optimal policy can be obtained through *value iteration*, a popular version of dynamic programming (see e.g., [LDK95] for details). Value iteration assigns to each location a a *value* $v(a)$ that represents its distance to the robot. Initially, $v(a)$ is set to 0 for the location $a = (0, 0)$ (which is the robot's location, recall that a is specified in relative coordinates), and ∞ for all other locations a. The value function $v(a)$ is then updated recursively according to the following rule:

$$v(a) \longleftarrow P_{occ}(a) + \min_b [v(b)] \tag{8}$$

Here b is minimized over all *neighbors* of a, i.e., all locations that can be reached from a with a single, atomic motor command. Eq. (8) assumes that the cost for traversing a point a is proportional to the probability that a is occupied ($P_{occ}(a)$). Iteratively applying Eq. (8) leads to the cost function for reaching any point a relative to the robot, and hill climbing in v (starting at a) gives the cost-optimal path from the robot's current position to any location a.

Action selection: Armed with the definition of the expected entropy and the expected costs, we are ready to set the policy for selecting actions in active localization. At every point in time, the robot chooses the action

$$a^* = \operatorname*{argmin}_a (E_a[H_P] + \alpha v(a)) \tag{9}$$

Here $\alpha \geq 0$ determines the relative importance of certainty versus costs. The choice of α depends on the application. In our experiments, α was set to 1.

While position probability grids can be efficiently updated using the techniques described in the previous section, they are still computationally too expensive for active localization. The complexity of computing the expected future entropy $E_a[H_P]$ for a single action a is in $O(|L| \cdot |S|)$, where $|L|$ is the number of grid-cells, and S the set of distinguishable sensings, so that computing the expected entropy for all possible actions and sensings is too time-consuming.

To approximate all necessary quantities in short time, we consider only a small subset of L, assuming that P can be approximated by a set P_m of m Gaussian densities with means $\mu_i \in L$. The center of the Gaussians μ_i are computed at runtime by scanning for locations in L whose probability exceeds a certain threshold. Our simplification is somewhat justified by the observation that in practice P is usually quickly centered on a small number of hypotheses and approximately zero everywhere else. Without this modification, action selection could not be performed efficiently enough.

To summarize our approach to active localization, actions, which represent target points relative to the robot's current position, are selected by (1) minimizing the tradeoff between the expected future uncertainty (entropy) of the belief state and (2) the costs of executing the action. In order to efficiently perform active localization, we approximate P by a small set of Gaussian distributions.

2.3 Application Example

This section describes an application example of our active localization technique in our department. The size of this test environment is approximately 20×27 m^2. To demonstrate the capabilities of our approach, we placed the robot in the corridor of the environment (see Figure 2). Notice that this corridor is basically symmetric and possesses various places that look alike, making it difficult for the robot to determine where it is. In this particular case, the robot must move into one of the offices, since only there it can find distinguishing features.

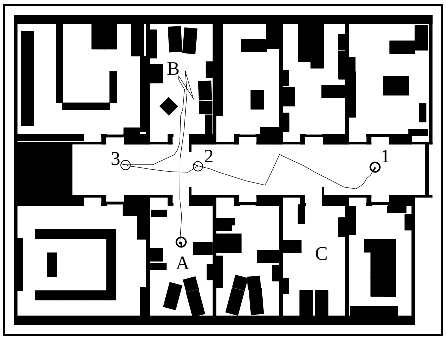

Fig. 2. Environment and path of the robot

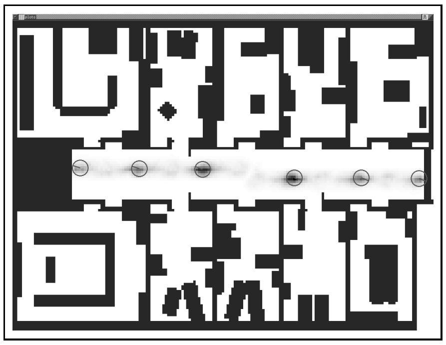

Fig. 3. Belief $P(l)$ at pos. 2

In a total of 10 experiments, random wandering and/or wall following consistently failed to localize the robot. This is because our wandering routines are highly unlikely to move the robot through narrow doors, and the symmetry of the corridor made it impossible to determine the location uniquely. In more than 20 experiments using the active navigation approach presented here, the robot always managed to localize itself in a considerably short amount of time making use of only its sonar sensors.

Figure 2 shows a representative example of the path taken during active localization, and also defines the positions and office names (1, 2, 3, A, B, C) used in the text. In this particular run, we started the robot at position 1 in the corridor facing south-west. The task of the robot was to determine its position within the environment, and then to move into room A (so that we could see that localization was successful). After about ten meters of random motion, it reached position 2 shown in Figure 2. Figure 3 depicts the belief P at this point in time (more likely positions are darker). The positions and orientations of the six local maxima are marked by the six circles. Based on the entropy-cost trade-off (see Eq. (8)), the robot first decided to move to the end of the corridor and progressed to position 3.

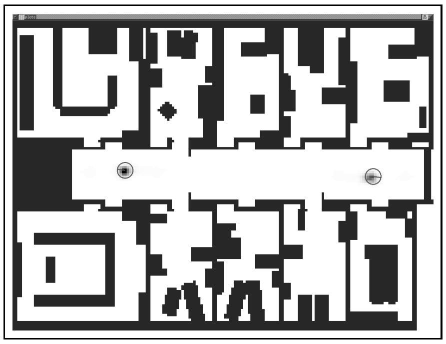

Fig. 4. Belief P at pos. 3

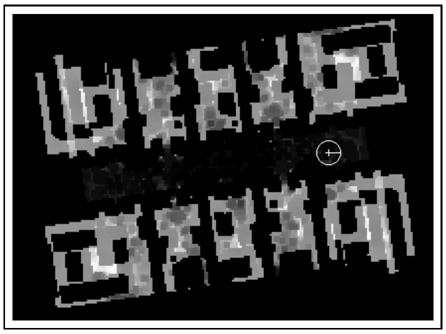

Fig. 5. Expected entropy $E[H_P]$ at pos. 3

After reaching the end of the corridor (position 3) the belief state contained only two local maxima (see Figure 4). Figure 5 shows the expected entropies of the target points, according to Eq. (6). High entropy is shown in dark colors. Figure 5 also contains the origin of the corresponding coordinate system. In this coordinate system a coordinate $\langle x, y \rangle$ represents a target point x meters in front of the robot and y meters to the left. Note that the ambiguity in Figure 4 can no longer be resolved without leaving the corridor. Accordingly, the expected entropy of target points in the corridor is high compared to the expected entropy of actions which guide the robot into the rooms.

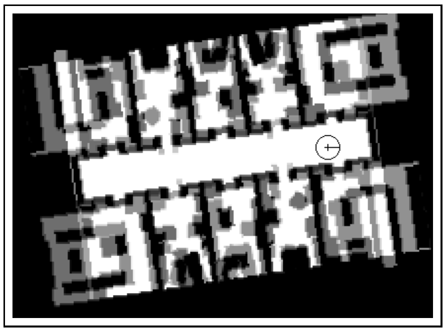

Fig. 6. Occupancy probability P_{occ} at pos. 3

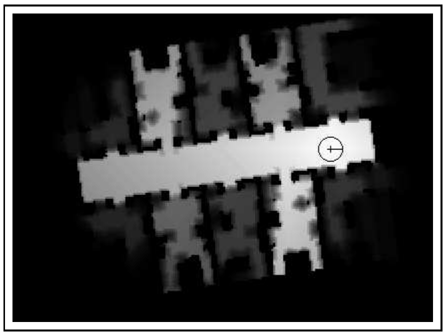

Fig. 7. Expected costs v at pos. 3

The expected occupancy probabilities P_{occ}, obtained by Eq. (7), are depicted in Figure 6. Note that this figure corresponds to a weighted overlay of the environmental map relative to the different local maxima, where the weights are given by the probabilities of the local maxima. Finally, Figure 7 displays the expected costs for reaching the different target points (see Eq. (8)). Because of the state of the doors, which influences the cost of reaching target points, the net-payoff (displayed in Figure 8) is maximal for target points in rooms B and C. This is why the robot decided to move into the room behind it on the right, which in this case turned out to be room B. After resolving the ambiguity between the rooms B and C the position has been determined uniquely and the robot moved straight to the target location in room A. Figure 9 shows the belief state at this final target point. The time for completing this experiment took less than eight minutes including navigation, position estimation, and the computation of the localization actions.

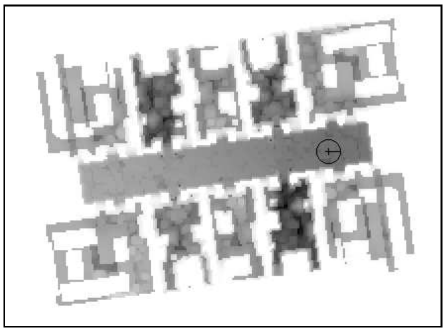

Fig. 8. $-E[H_P] - v$ at pos. 3

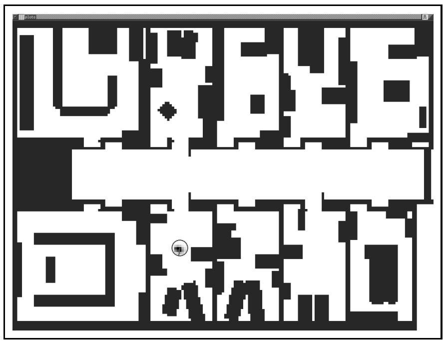

Fig. 9. Final belief P

3 The Control System of the Office Delivery Robot

The previous section described how the Markov localization method operates, how the measure of position uncertainty is computed, and how the active localization method directs the robot to eliminate uncertainty in the position estimate. In this section we will explain how these capabilities are incorporated into the robot control system of an office delivery robot and how the control system mediates between the primary activities of the office delivery robot and the activities issued by the active localization processes.

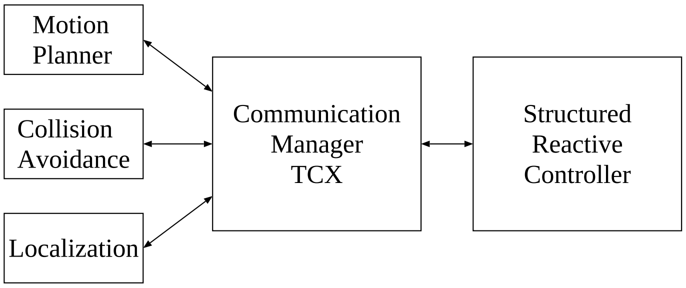

Fig. 10: Control architecture of the office delivery robot

The control system comprises modules, which provide such capabilities as "collision avoidance," "localization," and "motion planning." These modules run distributedly over the computer network and communicate asynchronously via message passing established by the communication management module TCX [Fed93]. Upon receiving a target point, the "motion planning" module computes, for each location in the environment, a feasible and optimal path between the location and the target point [TBB+98]. It then follows this path whilst providing the other modules with information about its progress by means of a sequence of intermediate target points on the path which have not yet been reached. These intermediate target points form the path to the desired location and are sent to the "collision avoidance" module. The "collision avoidance" module [FBT97] causes the robot to move to the next intermediate target point while at the same time avoiding unforeseen obstacles on its path. Finally, the localization module provides the other modules with the most likely position of the robot and a measure of the uncertainty in this estimate. In addition, the other modules can start and stop the active localization processes.

While the modules listed above provide a means for reliable navigation in the form of continuous control processes that can be activated and deactivated, they do not provide an effective means for combining the processes into coherent goal-directed behavior. This is provided by another module: the *structured reactive controller* [Bee96]. This controller is called structured because it makes use of expressive control abstractions to structure complex activities. It is called reactive because it can respond immediately to asynchronous events whilst managing concurrent control processes.

Structured reactive controllers are implemented in RPL [McD91]. RPL programs look very much like LISP programs but make use of control abstractions typical of structured concurrent programming languages. Such abstractions include those for sequencing, concurrent execution, conditionals, loops, assignments of values to program variables, and subroutine calls. Several high-level concepts (such as interrupts and monitors) are provided and used to synchronize parallel actions and to make plans reactive and robust.

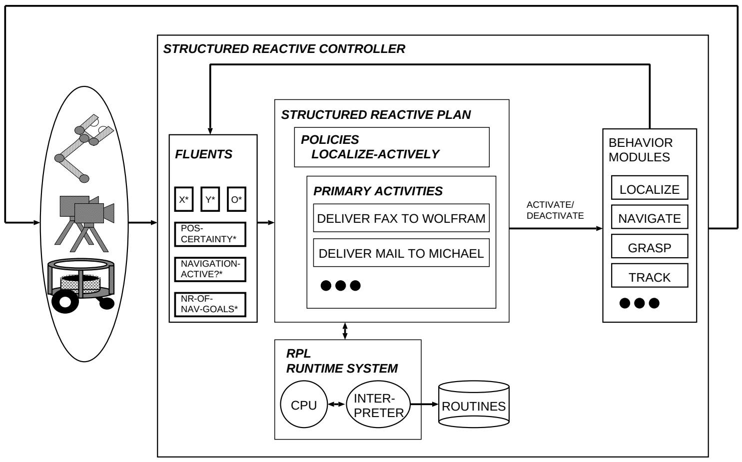

Fig. 11: Architecture of the structured reactive controller.

The main components of a structured reactive controller (see Figure 11) are the *behavior modules*, the *fluents*, the *structured reactive plan*, and the RPL *runtime system*. The elementary program units in the structured reactive controller are the behavior modules. Behavior modules constitute a uniform interface between RPL programs and the continuous control processes, such as collision avoidance and localization, run by other modules. Fluents, which are updated continuously according to the messages received by the structured reactive controller, provide information about the current state of the other modules of the control system. The structured reactive plan specifies how the robot is to respond to messages and feedback (from the navigation modules) to accomplish its jobs. It is a collection of concurrent control routines that specify routine activities and can adapt themselves to non-standard situations by executing planned responses [BM94].

A compact description of the mode of operation of the structured reactive controller is as follows. The RPL interpreter executes the structured reactive plan, causing the behavior modules to be activated and deactivated. Threads of control become blocked when they have to wait for certain conditions to become true. For example, if the robot is asked to go to a location x and pick up an object, the controller activates the behavior of moving towards x. The interpretation of the subsequent steps is blocked until the robot has arrived at x (that is until the move behavior signals its completion).

To control some robot with `RPL` plans, we have to specify the control processes provided by the remaining modules of the control system as behavior modules and store the state information provided by them in the global fluents of the structured reactive controller.

The behavior module `NAVIGATION` can be activated with a destination as its argument. The destination is specified through its coordinates, and by a navigation path, provided as a sequence of target positions. The navigation process performed by the behavior module causes the robot to move to the target points in the specified order whilst updating the register `NR-OF-NAV-GOALS*` containing the number of remaining target points to be reached by the robot. The module detects the successful completion of a navigation task, as given by measuring a suitably small distance between the robot and its target destination, and sends a success signal (cf. [McD92,Fir94]). A failure of the navigation task is signalled if the robot detects that it is unable to reach its destination [Fir92].

Combined with the behaviors `POSITION-TRACKING` and `ACTIVE-LOCALIZATION` described in the previous section, this description of the module `NAVIGATION` completes the list of behavior modules used in the remainder of the paper.

The following table shows the fluents used by the structured reactive controller of the office delivery robot and the information they store:

`X*, Y*, O*`	RHINO's position estimation (x and y coordinate and orientation)
`POS-CERTAINTY*`	RHINO's confidence in the global maximum of the position probability distribution
`NAVIGATION-ACTIVE?*`	True whenever the robot executes a navigation plan
`NR-OF-NAV-GOALS*`	number of remaining target points in the current navigation plan

3.1 Structured Reactive Navigation Plans

Office navigation plans are specified as concurrent reactive `RPL` routines. Figure 12 pictures such a structured reactive navigation plan, that is automatically generated for a given destination by the navigation planner of the structured reactive controller. The plan consists of two components. The first one

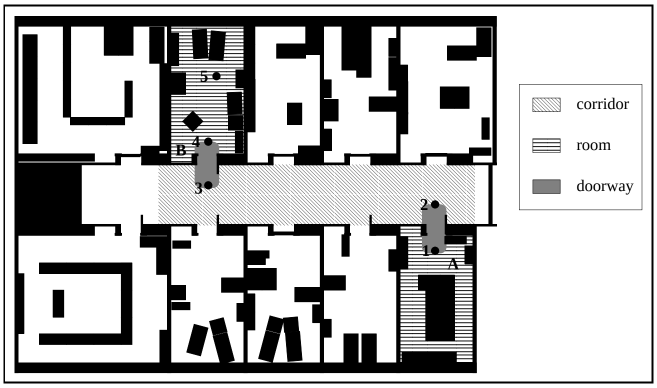

Fig. 12: Topological navigation plan for navigating from room A to B with regions indicating different travel modes

specifies a sequence of target points (the location indexed through the numbers 1 to 5 in Figure 12) to be reached by the robot. The sequence of target points encodes knowledge about the structure of the environment. It ensures that, for instance, the robot goes orthogonally through doorways and does not pass other offices on its way to the destination. The navigation between the target points is accomplished by a standard path planner [TBB+98].

The second component specifies in detail when and how the robot is to adapt its travel modes as it follows the navigation path. In many indoor environments it is advantageous to adapt the driving strategy to the surroundings: to drive carefully (and therefore slowly) within offices because offices are cluttered, to switch off the sonars when driving through doorways (to avoid crosstalk between the sonars), and to drive quickly in the hallways. This part of the plan is depicted through regions with different textures for the different travel modes “office,” “hallway,” and “doorway.” Whenever the robot crosses the boundaries between regions the appropriate travel mode is set in our collision avoidance module [FBT97].

Obviously these structured navigation plans make assumptions about the location of the robot while they execute the navigation plan. The sequence of target points makes sense only if the robot moves as expected. If the navigation plan gets interrupted, say because the robot relocalizes itself, the localization process might cause the robot to move to another region of the environment (see Section 2.2). To handle these situations, we use an `RPL` program that, given a destination such as “in front of Wolfram’s shelf,” computes a structured reactive navigation plan and executes it (see [BM96] for details).

The synchronization between active localization and navigation is specified using the `RPL` construct <u>`WITH-POLICY`</u> P B, which means, "execute the primary activity B such that the execution satisfies the policy P." In our case, "head towards the location *`DEST`* until you are there;" is subject to the policy that, whenever the robot loses track of its position, it must *first* go to a place where it can relocalize itself before it continues to go towards the destination. Policies implement constraints on the execution of the primary activities. They are concurrent processes that run while the primary activity is active and have higher priority than the primary activity. Most of the time these behaviors are wait-blocked, waiting for conditions to arise that require their intervention.

To enable `RPL` control threads to react to asynchronous events, such as the robot losing track of its position, we use fluents and in our particular case the fluent `POS-CERTAINTY*`. The role of fluents is best understood in conjunction with the `RPL` statements that respond to changes of fluent values. For instance, the `RPL` statement <u>`WHENEVER`</u> F B is an endless loop that executes B whenever the fluent F gets the value "true." The statement is implemented such that the control thread interpreting it goes into the state "wait-blocked" as long as F is false; waits for the fluent to signal that it has become true; and then resumes the interpretation. Besides `WHENEVER`, `WAIT-FOR` is another control abstraction that makes use of fluents. The `RPL` expression <u>`WAIT-FOR`</u> F blocks a thread of control until F becomes true.

Using the `RPL` constructs introduced above we can now specify a policy for the primary activities that localizes the robot actively whenever it is necessary and passes the control back to the primary activities after the relocalization has succeeded.

```
WITH-POLICY   WHENEVER (≤ POS-CERTAINTY* 0.8)
                 ACTIVATE-ACTIVE-LOCALIZATION;
                 WAIT-FOR(> POS-CERTAINTY* 0.8)
                 DEACTIVATE-ACTIVE-LOCALIZATION
  navigate to destination
```

Finally, it is necessary that our primary activities, the navigation routines, are robust against interruptions by the active localization process. The pseudo code that achieves this robustness is listed below. In order to prevent other (concurrent) routines directing the robot to other locations we make use of semaphores or "valves," which can be requested and released. Therefore, any process of the structured reactive controller that asks the robot to move around requests the valve `WHEELS`, moves the robot only after it has received `WHEELS`, and releases `WHEELS` after it is done.

```
WITH-POLICY  [relocalize when necessary]
  PROCESS NAVIGATION-PROC
    WITH-VALVE WHEELS
      EVAP-PROTECT
        LOOP
          TRY-ALL
            WAIT-FOR (EXECUTION-INTERRUPTED?)
            GENERATE&EXECUTE-NAVIGATION-PLAN(DEST)
        UNTIL VERIFY-CONDITION(AT DESTINATION?)
       PLAN-REMOVE-ALL-GOALS
```

Of course, in many cases processes with higher priorities must move the robot urgently. In this case, the valves are simply pre-empted. To make our routine robust against such interrupts the routine has to do two things. First, it has to detect when it gets interrupted and, second, it has to handle such interrupts appropriately.

To detect interrupts by other processes the navigation process monitors the target points given to the path planner. In our control system, a process has to delete all target points before it can cause the robot to move somewhere else. Thus, interrupts that are critical for the navigation routine are detected by monitoring the fluent NR-OF-NAV-GOALS* and pulsing a fluent EXECUTION-INTERRUPTED? whenever the number of target points has dropped to zero.

To handle interrupts, we implement the navigation routine as a loop that generates and executes navigation plans for the navigation task until the robot can verify that it is close enough to its destination. Interrupts are handled by terminating the current iteration of the loop and starting the next iteration, in which a new navigation plan starting from RHINO's new position is generated and executed. The TRY-ALL construct succeeds when its first subtask succeeds, that is if the navigation plan has been completely executed or the navigation plan has been interrupted.

There is one more case that our routine has to handle, namely an asynchronous deactivation of the navigation process. Upon deactivation of the behavior modules, the corresponding control processes cannot simply stop. For example, the robot cannot simply start moving around to relocalize itself while it is in the middle of grasping an object and has its arm extended. To handle such deactivations, we use the construct EVAP-PROTECT A B that secures the execution of the code piece B (in our case, to delete the remaining target points) in the case that A is deactivated in the midst of its execution.

3.3 Related Work on High-level Control

RPL is a successor of the RAP system [Fir89,Fir87]. It borrows many of RAP's ideas and adds concepts for structuring complex reactive activities and synchronizing concurrent actions. `ESL` [Gat96] is a language with similar expressive power but, unlike `RPL`, does not provide tools for reasoning, revising robot control programs, or local runtime planning [BM96]. `RS` [LA89], another robot control language that provides similar control abstractions and planning facilities, is tailored to highly repetitive tasks. Other languages such as `Gapps` [Kae88], `Rex` [Kae87] and the `Behavior Language` [Bro90] can be realized as macro languages using `RPL`'s control structures.

Structured reactive controllers are hybrid robot control architectures that integrate deliberation and action, discrete actions, and continuous control processes. The most prominent class of architectures are hybrid layered approches that have three separate layers of control [BFG+97]. Structured reactive controllers can avoid flaws in the behavior of the robot that hybrid layered approaches cannot. This comes at a cost: the action planning systems in structured reactive controllers have to reason about — and possibly revise — full-fledged concurrent robot control programs, whereas planning systems in layered architectures can restrict themselves to partially ordered sets of actions. Another architecture that has been successfully applied to autonomous robot control is the `BDI` architecture [RG92] as implemented for instance in the Procedural Reasoning System (`PRS`) [GI89]. The main difference between structured reactive controllers and `PRS` is that `PRS` does not make a commitment to planning techniques or the interaction between planning and execution. It leaves it to the programmer to implement them as meta-procedures when required. `TCA` (Task Control Architecture) [Sim94] is a specialized real-time operating system for managing the task-specific processes of a robot controller. `TCA` provides a C subroutine library for interleaving planning and execution, monitoring environment changes and reacting to them, recovering from execution failures, and for coordinating multiple processes. Since `TCA` is implemented as a C subroutine library, it cannot provide the powerful control abstractions provided by `RPL`, `ESL`, or `RS` and, as a consequence, complex problem-solving behavior cannot be implemented in a concise and transparent form.

4 Experiments

To demonstrate the capabilities of our approach we performed several complex experiments in our office environment. In all these experiments the task of the robot was to reach a location in front of a shelf in one room of our building. In order to force the control system to start the active localization process we manually rotated the robot by about 70 degrees shortly after starting its mission.

Figures 13-17 show a representative example of these experiments. Figures 14-17 contain an outline of our environment including the corresponding position

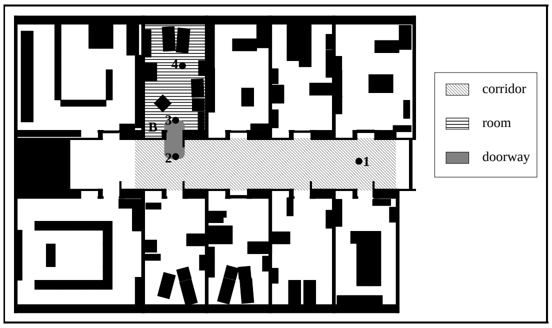

Fig. 13: Initial plan of the delivery robot

probability density (dark regions are more likely). In this example RHINO was started at the east end of the hallway (see Figure 14). The destination was a location in front of Wolfram's shelf in room B. Upon receiving the command to go in front of wolfram's shelf, the structured reactive controller computed the navigation plan shown in Figure 13. Shortly after starting the execution of the navigation plan we manually turned the robot to the right. After this turn the robot was facing west (i.e. to the left of the page) while the position tracking assumed the robot to face south-south-west. Consequently the path planner generated target points that caused the robot to turn right and move ahead. Shortly after that, RHINO's reactive collision avoidance [FBT97] turned the robot to the left because of the northern (top) wall of the corridor. While the solid line in Figure 14 shows the real trajectory of the robot, the dashed line depicts the trajectory estimated by the position estimation module. Note that this module didn't realize the manual change in orientation. To the robot it seemed as if there was som obstruction in the corridor which forced it off course and through the doorway. At the end of this path the accumulated discrepancies between expected and measured sensor readings reduced the confidence in the position of the robot below the given threshold.

At that point the robot control system interrupted the navigation process and started the active localization module. The inconsistencies in the belief state caused the localization module to re-localize the robot, which starts with uniform distribution over all possible positions in the environment (see Figure 15). After a short period of random movement, the number of possible positions had been reduced to 10 local maxima in the belief state where the positions at the west end facing east and the east end facing west were the most likely ones. To disambiguate these two positions, RHINO decided to enter the next room to its left (either room C or D). This target location was chosen because it was expected to maximally reduce the uncertainty of the position estimation (see Section 2.2). In this particular run the corresponding navigation action lead RHINO into room C, where in fact all ambiguities could be resolved (see Figure 16).

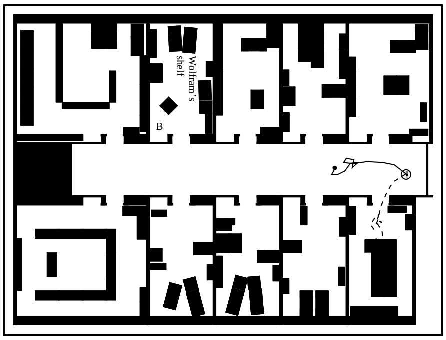

Fig. 14. Path of the robot before active localization

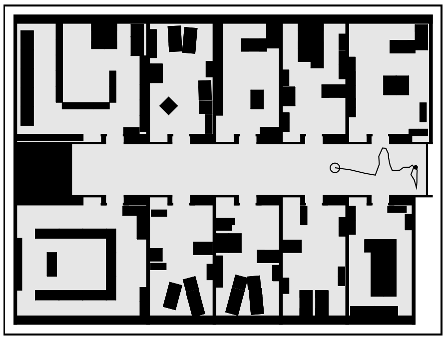

Fig. 15. Initial stage of active localization

After successful relocalization, the control was transferred back to the navigation task which generated a new navigation plan to room B (see Figure 12), which as shown in Figure 17 was executed immediately after re-localization. Figure 17 also shows the complete trajectory taken by the robot.

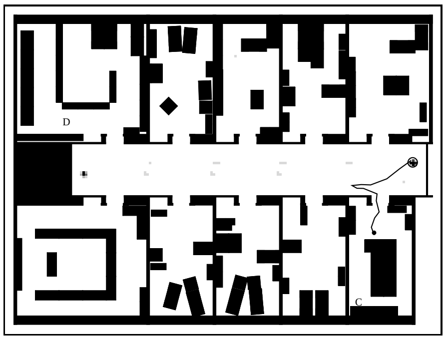

Fig. 16. RHINO moves into room C for disambiguation

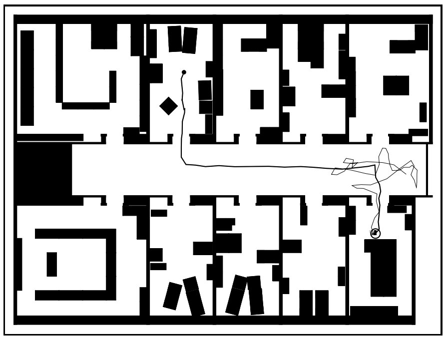

Fig. 17. Navigation after successful re-localization

5 Conclusions

In this paper we presented an integration of global active localization into a high-level robot control system. The principle of this approach is to monitor the certainty of the (passive) position tracking process and to invoke active localization whenever the uncertainty grows too large. To realize this approach, we treat active localization as an *action* which possibly interferes with other actions of the robot. Thus, to ensure that the robot can reliably carry out its mission, the high-level routines are implemented such that they provide a means for handling interrupts and continuing successfully after reactivation.

We proposed a localization system which is able (1) to efficiently and accurately track the robot's position, (2) to detect situations in which the tracking has failed, and (3) to actively relocalize the robot in the case of such a failure. All these features are basic preconditions for establishing mobile robots with a high degree of autonomy.

We showed how to design plans that can be interrupted by active localization processes at any time and still permit the mission to be reliably completed. Such plans are coded concisely using structured reactive plans written in `RPL`. `RPL` provides high-level control abstractions, such as concurrent processes, interrupts, and monitors. Additionally, it provides means to synchronize parallel actions and to make plans reactive and robust.

Our approach has been implemented and tested in a real world office environment. In different experiments this method has been proven to be reliable and effective. It allows the robot to complete its mission even if the robot has been manually rotated producing a rotational error of more than 70 deg. The time resources consumed by our approach to localization are negligible: autonomous localization in a real world office environment of size 20×27 m^2 without any prior information about the robot's position can be done within only a few minutes. In our experience, there have hardly been any cases in which the robot has lost its position when using our passive localization method [BFHS96]. Thus, the expected average time costs for localization are extremely low. Furthermore, the time spent on localization is generally outweighed by the performance gain resulting from accurately knowing the robot's position.

Despite these encouraging results there are several warrants for future research. Obviously, the efficiency of the system can be improved by adapting the position certainty measure according to the current tasks. As an example, consider the situation where the robot has to transport an object and dispose of it anywhere in the hallway. In this case it suffices to know that the robot is in the hallway, but the exact location is not needed. To competently handle such problems, the control system has to predict the effect of position uncertainty on its course of action and react appropriately. Active localization could be improved by taking into account the navigation goals when choosing the best localization action. Another source of improvement is to prefer tasks which direct the robot to locations that are expected to increase the certainty in the position estimate. Such locations could be learned through long-term experience.

Acknowledgement

We would like to thank Thomas Arbuckle and Sebastian Thrun for fruitful discussion and valuable comments on earlier versions of this article. This work has partially been supported by EC Contract No ERBFMRX-CT96-0049 under the TMR Programme.

References

[BBC+95] J. Buhmann, W. Burgard, A.B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(2):31–38, Summer 1995.

[BCF+97] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The *RHINO* museum tour-guide project. `http/www.cs.uni-bonn.de/~rhino/tourguide`, 1997.

[Bee96] M. Beetz. *Anticipating and Forestalling Execution Failures in Structured Reactive Plans.* Technical report, yale/dcs/rr1097, Yale University, 1996.

[BEF96] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd., Wellesley, MA, 1996.

[BFG+97] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(1), 1997.

[BFH97] W. Burgard, D. Fox, and D. Hennig. Fast grid-based position tracking for mobile robots. In *Proc. of the 21th German Conference on Artificial Intelligence, Germany.* Springer Verlag, 1997.

[BFHS96] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the Fourteenth National Conference on Artificial Intelligence*, pages 896–901, 1996.

[BFT97] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[BM94] M. Beetz and D. McDermott. Improving robot plans during their execution. In Kris Hammond, editor, *Second International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1994.

[BM96] M. Beetz and D. McDermott. Local planning of ongoing behavior. In Brian Drabble, editor, *Third International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1996.

[Bro90] R. Brooks. The behavior language; user's guide. A.I. Memo 1227, MIT AI Lab, Cambridge, MA, 1990.

[FBT97] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, March 1997.

[Fed93] C. Fedor. *TCX. An interprocess communication system for building robotic architectures. Programmer's guide to version 10.xx.* Carnegie Mellon University, Pittsurgh, PA 15213, 12 1993.

[Fir87] J. Firby. An investigation into reactive planning in complex domains. In *Proc. of AAAI-87*, pages 202–206, Seattle, WA, 1987.

[Fir89] J. Firby. *Adaptive Execution in Complex Dynamic Worlds.* Technical report 672, Yale University, Department of Computer Science, January 1989.

[Fir92] J. Firby. Building symbolic primitives with continuous control routines. In J. Hendler, editor, *AIPS-92*, pages 62–69, Morgan Kaufmann, 1992.

[Fir94] J. Firby. Task networks for controlling continuous processes. In Kris Hammond, editor, *AIPS-94*, pages 49–54, Morgan Kaufmann, 1994.

[Gat96] E. Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. In *AAAI Fall Symposium: Issues in Plan Execution*, Cambridge, MA, 1996.

[GCJK97] D. Guzzoni, A. Cheyer, L. Julia, and K. Konolige. Many robots make short work. *AI magazine*, 18(1), 1997.

[GI89] M. Georgeff and F. Ingrand. Decision making in an embedded reasing system. In *Proc. of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978, Detroit, MI, 1989.

[HK96] J. Hertzberg and F. Kirchner. Landmark-based autonomous navigation in sewerage pipes. In *Proc. of the First Euromicro Workshop on Advanced Mobile Robots*, pages 68–73, 1996.

[Kae87] L. Kaelbling. REX: A symbolic language for the design and parallel implementation of embedded systems. In *Proceedings of AIAA Conference on Computers in Aerospace*, Wakefield, MA, 1987.

[Kae88] L. Kaelbling. Goals as parallel program specifications. In *Proc. of AAAI-88*, pages 60–65, St. Paul, MN, 1988.

[KCK96] L.P. Kaelbling, A.R. Cassandra, and J.A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[KMRS97] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 1997.

[LA89] D. Lyons and M. Arbib. A formal model of computation for sensory-based robotics. *IEEE Journal of Robotics and Automation*, 5(3):280–293, 1989.

[LDK95] M.L. Littman, T.L. Dean, and L.P. Kaelbling. On the complexity of solving markov decision problems. In *Proc. of the Eleventh International Conference on Uncertainty in Artificial Intelligence*, 1995.

[McD91] D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[McD92] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[NPB95] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine*, 16(2), Summer 1995.

[RG92] A. Rao and M. Georgeff. An abstract architecture for rational agents. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Third International Conference (KR'92)*, pages 439–449, San Mateo, CA, 1992. Kaufmann.

[SGH+] R.G. Simmons, R. Goodwin, K.Z. Haigh, S. Koenig, J. O'Sullivan, and M.M. Veloso. Xavier: Experience with a layered robot architecture. *ACM magazine* Intelligence. to appear.

[Sim94] R. Simmons. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 34–43, 1994.

[SK95] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. International Joint Conference on Artificial Intelligence*, 1995.

[TBB+98] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998. to appear.