An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments*

Cyrill Stachniss Wolfram Burgard

University of Freiburg, Department of Computer Science, D-79110 Germany

Abstract

Whenever robots are installed in populated environments, they need appropriate techniques to avoid collisions with unexpected obstacles. Over the past years several reactive techniques have been developed that use heuristic evaluation functions to choose appropriate actions whenever a robot encounters an unforeseen obstacle. Whereas the majority of these approaches determines only the next steering command, some additionally consider sequences of possible poses. However, they generally do not consider sequences of actions in the velocity space. Accordingly, these methods are not able to slow down the robot early enough before it has to enter a narrow passage. In this paper we present a new approach that integrates path planning with sensor-based collision avoidance. Our algorithm simultaneously considers the robot's pose and velocities during the planning process. We employ different strategies to deal with the huge state space that has to be explored. Our method has been implemented and tested on real robots and in simulation runs. Extensive experiments demonstrate that our technique can reliably control mobile robots moving at high speeds.

1 Introduction

Path planning is one of the fundamental problems in mobile robotics. As mentioned by Latombe [11], the capability of effectively planning its motions is "eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world." Over the past years, a huge variety of techniques for motion planning has been developed. They can roughly be divided into map-based approaches such as road-map or cell-decomposition techniques (see [11] for an extensive overview), and reactive, sensor-based approaches [1, 4, 6, 7, 9, 10, 15].

Goal-directed path planning techniques compute paths based on a given map of the environment. Thereby they assume that the environment does not change while the robot is moving. However, when robots are designed to operate in dynamic or populated environments, this assumption is no longer justified. Instead the robots have to be equipped with sensors to be able to react to unforeseen obstacles. Therefore, many researchers have developed reactive, sensor-based approaches that control the movements of the robot based on its sensory input. Typical members of this class are the curvature-velocity method [15], the dynamic window approach [4], the vector-field histogram techniques [1] and the potential field techniques [8]. The key idea of these approaches is to use an evaluation function to determine the best next action based on the current sensory input and eventually given the current state of the system. The advantage of these techniques lies in their efficiency, so that new commands can be generated at a high frequency. One disadvantage of the purely sensor-based approach is the sub-optimality which comes from the fact that potentially available global information is ignored. Moreover, techniques of this type are also incomplete, since the robots can get stuck in U-shaped obstacles.

Therefore, several researchers have worked on extensions to remedy these problems. For example, Kathib and Chatila [7] modify the potential function to make the motion of the robot more efficient and to achieve certain desirable behaviors such as wall following and tracking. Schlegel [13] presents an approach to modify the evaluation function according to the shape of the robot, which is especially useful for transportation or manipulation tasks. The techniques described by Fox et al. as well as Schmidt and Azarm [5, 14] combine the sensory information with a given map of the environment, for example to deal with objects that cannot be detected with the robot's sensors. Ko and Simmons [9] introduced the Lane-Curvature method. This approach extracts lanes out of a given map of the environment and modifies the evaluation function so that the robot stays on the lanes. This leads to smoother trajectories, especially in long corridors.

Additionally, there are approaches to integrate path-planning techniques with sensor-based approaches. For example, the systems described by Burgard et al. and Thrun et al. [3, 17] uses a path planning system to compute intermediate points lying on the optimal path. These intermediate points are transmitted to the reactive collision avoidance system, so that the robot can no longer

*This work has partly been supported by the EC under contract number IST-2000-29456.

get stuck in local minima of the evaluation function. Recently, Brock and Kathib [2] presented an elegant integration of path planning and reactive collision avoidance. They modify the evaluation function according to the previously planned path and this way can integrate the knowledge about globally optimal actions into a sensor-based approach.

Whereas all these approaches have been proven to allow mobile robots to reliably navigate in dynamic environments, the resulting paths sometimes are sub-optimal. As we will point out in more detail in this paper, one of the main reasons for this lies in the fact that the next control action for the robot is chosen mainly based on its current state (position, orientation and velocities), eventually taking into account a globally planned path. This path, however, is only planned in the $\langle x, y\rangle$ space without considering the orientation and velocities of the robot. As we will argue below, in certain situations the kinematics of a robot also has to be considered during the map-based path planning, especially when the robot moves at a high speed.

In this paper we present integrated path-planning and collision avoidance technique that takes into account the kinematics of the robot as well as the dynamic of the environment. Our method plans the control actions of the robot in the space of positions ($\langle x, y, \theta\rangle$) and velocities ($\langle v, \omega\rangle$). We apply several techniques to cope with the enormous complexity of the state space that has to be explored. As a result our system is able to efficiently determine the next motion command. Our technique has been implemented and tested in extensive experiments on different robot systems as well as in simulation. All experiments show that our technique is able to reliably control a robot in dynamic environments even with narrow passages. They furthermore illustrate that our approach yields a better behavior than previous approaches.

This paper is organized as follows. In the next section we will describe the motion equati8ons for synchro-drive and differential-drive robots, which build the basis for our collision avoidance system. Section 3 contains a discussion of the popular dynamic window approach. In Section 4 we will present our planning technique. Finally, in Section 5 we describe different experiments illustrating the capabilities of our approach. We also perform a comparison to an alternative technique and to the optimal solution.

2 Motion Equations for Synchro-Drive and Differential-Drive Robots

To plan the trajectories of a robot our system is guided by the well-known motion equations for synchro-drive or differential-drive robots [4]. Thereby it assumed that the trajectory of a robot can be approximated by a sequence of circular arcs. Accordingly, the x-coordinates of the

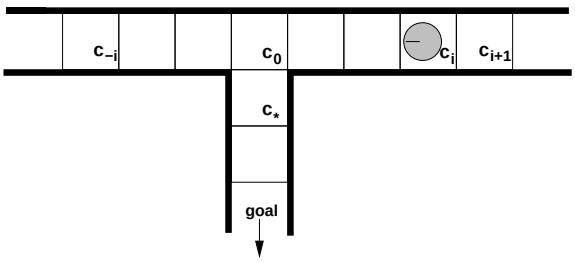

Figure 1: *Situation in which a robot using the dynamic window approach cannot reach the goal.*

robot given it starts at position $x(t_0)$ and given that it has the velocities $\langle v_i, \omega_i\rangle$ at time t_i can be computed as:

$$x(t_n) \quad = \quad x(t_0) + \sum_{i=0}^{n-1} (F_x^i(t_{i+1}))$$

where

$$F_x^i(t) \quad = \quad \begin{cases} -\frac{v_i}{\omega_i}(\sin\theta(t_i) - \sin(\theta(t_i)+ \\ \quad \omega_i \cdot (t - t_i))), \, \omega_i \neq 0 \\ \\ v_i \cos(\theta(t_i)) \cdot t, \, \omega_i = 0. \end{cases}$$

The corresponding equations for the y-coordinate are obtained by replacing $\cos$ by $\sin$ and multiplying the first line by -1.

3 The Dynamic Window Approaches

The dynamic window approach (DWA) is a popular technique for reactive collision avoidance. Their key idea is to use an evaluation function that takes as input the current state of the system (pose and velocities) and to determine a new steering command (translational and rotational velocities) such that a given evaluation function is maximized. The search for appropriate steering commands is carried out within the set of admissible velocities. Among them are all velocities that can be reached within a certain time frame given the maximum possible accelerations or decelerations of the system. Furthermore, velocities that would inevitably lead to a collision with an obstacle are not considered admissible. The evaluation functions typically measure the progress towards the goal, the current velocities and the distance to obstacles [2, 4, 9, 13, 15].

Although dynamic window techniques have successfully been applied on a variety of robot systems, there are situations in which they lead to a sub-optimal behavior. As we figured out in practice, this mainly comes from the fact that the existing approaches only consider the immediately next action or, if not, only plan in the $\langle x, y\rangle$ configuration space. However, when the robot travels along a corridor and has to enter a doorway, it has to decelerate early enough to be able to actually turn into the doorway. Approaches not performing a look-ahead or techniques not considering the velocities during planning are

not able to detect this and therefore will fail to produce the correct motion commands.

For example, consider the situation depicted in Figure 1. The Global Dynamic Window Approach [2] tries to maximize the following evaluation function:

$$\begin{aligned}\Omega_g(p,v,a) &= \alpha \cdot \textit{nf1}(p,v) + \beta \cdot \textit{vel}(v) \\ &\quad +\gamma \cdot \textit{goal}(p,v,a) + \delta \cdot \Delta\textit{nf1}(p,v,a).\end{aligned}$$

Here, $\textit{nf1}(p,v)$ is the so-called navigation function, which is based on A^*-search in the $\langle x,y\rangle$ space. The term $\textit{vel}(v)$ is a function that assesses the velocity of the robot. Far away from the goal, a high velocity means a high assessment. Furthermore, $\textit{goal}(p,v,a)$ is a binary function which indicates that the robot is near the goal. $\Delta\textit{nf1}(p,v,a)$ is the grid-based gradient of the navigation function. Now suppose that the goal is at the very end of the long corridor starting in the field marked c_0 and facing south. Furthermore suppose the robot evaluates Ω_g when it is in cell c_i with $i > 1$. Obviously, the values of $\textit{nf1}(p,v)$ force the robot to stay on its horizontal path. The same holds for the gradient $\Delta\textit{nf1}(p,v)$. Furthermore, $\textit{goal}(p,v,a)$ is zero here, because the target location is far away from c_0. Please note that only in c_0 the values of $\textit{nf1}(p,v)$ and $\Delta\textit{nf1}(p,v)$ force the robot to turn south. Thus, the steering command in the cells c_i ($i > 1$) of the robot is mainly governed by the term $\textit{vel}(v)$. Since the system seeks to maximize Ω_g, the robot will accelerate as much as possible. As a result, it will be too fast to make the necessary turn into the corridor when reaching c_0. Thus, without considering the velocities the robot will fail to enter the corridor. This illustrates that it is not sufficient to plan in the $\langle x,y\rangle$-space only. Instead one has to perform a look-ahead in the space of velocities to choose appropriate steering commands.

4 Velocity-based Motion Planning

Our approach to integrated path planning and collision avoidance considers the five-dimensional $\langle x,y,\theta,v,\omega\rangle$ configuration space and tries to optimize a trade-off between time and collision risk. Unfortunately, planning in the whole state space is too time-consuming and cannot cope with the real-time constraints imposed by a robot moving quickly in dynamic environments. Our system therefore employs different strategies to deal with the huge size of the space that has to be explored as well as with the dynamics of the environment. In particular, we proceed in the following four steps explained in more detail below:

1. Update the given (static) map according to the recent sensory input.
2. Compute a path in the $\langle x,y\rangle$ space given the updated map.
3. Use the path generated in Step 2 to determine the search space to be explored in the next step. Furthermore compute a heuristic to guide the search during this exploration.
4. Search for a partial path in the fraction of the $\langle x,y,\theta,v,\omega\rangle$ configuration space computed in Step 3.

Step 1: To represent the environment, our system uses occupancy grid maps [12]. This representation separates the environment into a grid of equally spaced cells and stores in each cell $\langle x,y\rangle$ the probability $P(occ_{x,y})$ that it is occupied by an object. We assume that a map representing the static aspects of the environment is a priori given. To integrate sensory input into this map we apply a conservative strategy. We simply set the occupancy probability of a cell in which the beam ends to 1.0 which prevents the system from planning a path through that cell. As soon as this cell is detected to be free again or a certain period of time has elapsed (120 secs), we reset its occupancy value to the original value.

Step 2: The goal of the second step is to compute a path from the current location to the target position in the $\langle x,y\rangle$-space. Our system applies the popular A^* procedure using the grid-graph induced by the occupancy grid map to find the shortest path. Thereby, the cost for traversing a cell $\langle x,y\rangle$ is proportional to its occupancy probability $P(occ_{x,y})$. Cells for which $P(occ_{x,y})$ exceeds a threshold of 0.15 are assumed to have infinite costs. To speed up the search, the heuristic is based on a value-iteration in the $\langle x,y\rangle$-space computed using the static map. Please note that this heuristic has to be computed only once for each global target location.

Step 3: The goal of this step is the computation of the search space to be explored in the next step. To this end we create a channel around the path computed in the previous step. In our current system we adopt the width and length of this channel dynamically. We start with a channel width of 70 cm at the current location of the robot and expand the channel along the path computed in Step 2. We stop the process as soon as the channel has reached a certain size that depends on the performance of the underlying computer. The goal is to determine a channel that can be explored within the next 0.25 seconds so that the system can issue a new motion command within this time frame. The final location on the optimal two-dimensional path reached during the channel expansion will be the next subgoal $\langle x_g,y_g\rangle$ for the robot. We also perform a deterministic value-iteration in this channel. The resulting value function is identical to the navigation function *nf1* described above and builds an admissible heuristic for the A^* search carried out in the following step.

Step 4: In the final step we compute a path from the robot's current state $\langle x,y,\theta,v,\omega\rangle$ to the location $\langle x_g,y_g\rangle$. Thereby we explore the five-dimensional con-

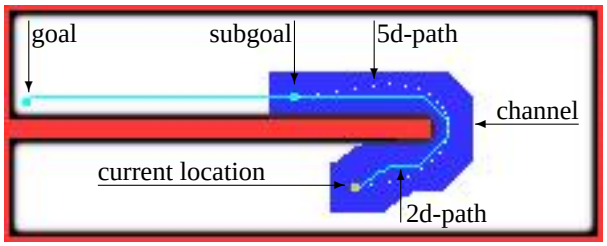

Figure 2: *Trajectories and corresponding channel generated in one cycle.*

figuration space $\langle x, y, \theta, v, \omega \rangle$ bounded by the channel computed in step 3. To determine appropriate values for θ_g, v_g and ω_g we distinguish two situations. If $\langle x_g, y_g \rangle$ corresponds to the global target location we allow an arbitrary orientation for θ_g. In this case the velocities v_g and ω_g, however, have to be zero. If $\langle x_g, y_g \rangle$ differs from the global target, we allow arbitrary values for v_g but set ω_g to zero and θ_g to the direction between $\langle x_g, y_g \rangle$ and the next position on the path computed in step 2. Again we apply A^* to find the optimal sequence of motion commands. As heuristic we use the value function computed in the previous step. The discretization of the state space typically is 10 cm for the position, $\pi/16$ degrees for the heading, $\pi/16$ degrees per second for the rotational velocity, and 10 cm/sec for the translational velocity. To compute the successor state of a given state, we use the motion equations described in Section 2:

$$\langle x_1, y_1, \theta_1, v_1, \omega_1 \rangle \stackrel{\langle v_2, \omega_2 \rangle}{\longrightarrow} \langle x_2, y_2, \theta_2, v_2, \omega_2 \rangle.$$

Thereby, the maximum changes in the velocities are limited by the accelerations of the robot's motors. As a result, we obtain a sequence of velocity commands $\langle v, \omega \rangle$ the robot must execute to reach $\langle x_g, y_g, \theta_g, v_g, \omega_g \rangle$.

Figure 2 shows some of the data structures generated during Steps 1 to 4. The solid line corresponds to the optimal path planned in the two-dimensional configuration space. In this situation the robot starts in the lower corridor and has to travel to the left end of the upper corridor. Also shown is the channel computed in Step 3 and the resulting intermediate goal location $\langle x_g, y_g \rangle$. The dotted line corresponds to the trajectory resulting from the full planning in the five-dimensional configuration space in Step 4.

Our current implementation is highly efficient [16]. It uses lookup-tables to perform most of the geometric operations. This includes the computation of cells covered by the robot and the computation of successor states. Our current system is able to explore channels of 2 m length and 70 cm width within 0.25 seconds on a standard 800 MHz Pentium III computer. Whenever the channel is too large to be explored, we use the dynamic window technique to quickly generate admissible velocities. We then reduce the size of the channel for the next time frame. If, however, the channel turns out to be too small since sufficient time remains in Step 4, we increase the size of the channel appropriately. This way, our approach dynamically adapts itself to the performance of the underlying processor and to the complexity of the planning problems. Please note, that our approach in principle is able to compute the optimal path from the robot's current location to the target position given the map and the current sensory input. If enough computational resources are available, the channel can cover the whole configuration space so that the system can compute the optimal sequence of steering commands.

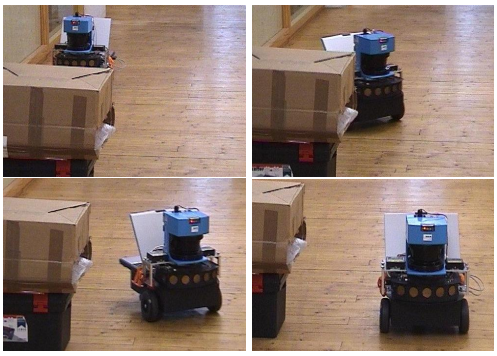

Figure 3: *Pioneer 1 robot Ludwig driving around an unexpected obstacle.*

5 Experimental Results

Our path planning method has been implemented and extensively tested on our mobile robots Albert and Ludwig. Whereas Albert is an RWI B21r robot, Ludwig is a Pioneer 1 system. Both robots are equipped with SICK laser range finders that are used to detect dynamic obstacles. Additionally, we carried out a series of simulation runs to compare our system to the dynamic window technique described in [4].

5.1 Collision avoidance in Dynamic Environments

The first experiment was carried out using both robots in our office environment at the University of Freiburg. To test the capabilities of our system to deal with unexpected obstacles we installed several objects in the corridor and changed their positions frequently. Additionally, people were walking in the environment. In both experiments, during which the robots traveled over 300 m with average speeds of over 30 cm/s, we did not observe a single collision. Figure 3 shows a typical situation during these experiments. Here Ludwig is moving around an unexpected obstacle in the corridor. We also performed extensive simulation experiments with an overall path length of 20 km. The simulator we used realizes the full functionality and behavior of the a mobile platform including the ability to set different accelerations, velocities etc. During all experiments we found that the generated paths were very smooth and that the overall behavior was quite efficient.

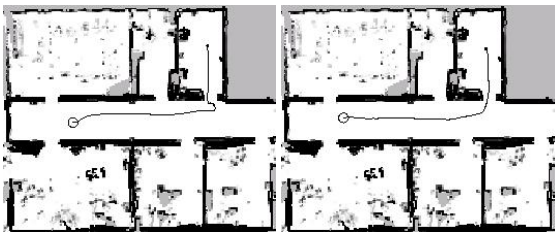

Figure 4: *Typical trajectories taken by Albert when entering a room using the DWA (left) and our approach (right).*

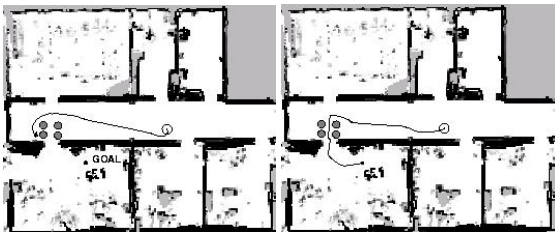

Figure 5: *Albert tries to enter a doorway blocked by several obstacles using the DWA (left) and our technique (right).*

5.2 Comparison to the Dynamic Window Approach

We also performed several experiments to compare our technique to the collision avoidance system that we successfully employed over the past years. This system uses the dynamic window approach technique combined with a 2d path planner [17]. Figure 4 shows the outcome of one such experiment. Here the robot had to travel along the corridor of our office environment and had to enter the rightmost office in the north. The left image shows the trajectory generated by the dynamic window approach. The right image contains the trajectory generated by our algorithm. Since the DWA chooses too high speeds in the corridor, it is not able to directly enter the room when it reaches the doorway. Rather it first stops, turns back and then enters the room. Our module, in contrast, slows the robot down before it reaches the doorway area so that Albert is able to directly enter the room. With our system the robot completed the whole run in 34.5 seconds driving 11.86 m, which corresponds to an average speed of 34.3 cm/sec. The dynamic window technique, however, required 48.5 seconds and traveled 12.31 m, which results in an average speed of 25.4 cm/s. The maximum speed of the system was set to 40 cm/sec in both runs.

Another experiment carried out with Albert is depicted in Figure 5. Here we installed several objects in front of a doorway to increase the difficulty of entering the corresponding room. As can be seen in the left image, the dynamic window technique again is too fast to make the necessary sharp turn into the narrow passage. Our technique, however, slows the robot down early enough so that it can enter the passage immediately. Figure 6 depicts a sequence of images showing Albert executing the

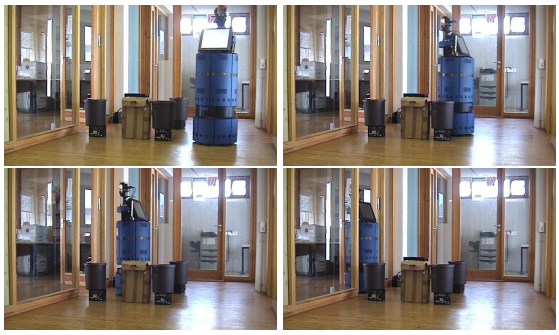

Figure 6: *Albert traveling along the path shown in the right image of Figure 5.*

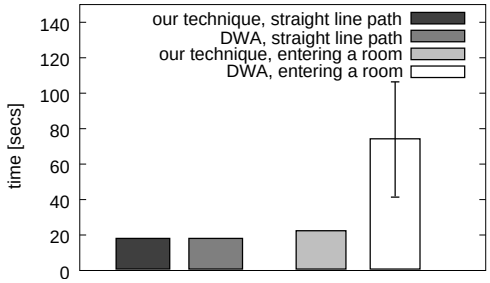

Figure 7: *Average time needed to reach the goal location using our planning module and using the DWA.*

steering commands generated by our system.[1]

At this time we would like to mention that the evaluation function used by the dynamic window techniques in fact can be tuned to optimize the behavior of a robot in specific situations like the ones given in the experiments described above. However, this generally reduces the performance of the system in other situations. In the experiments described here we therefore used parameters for the DWA that we generally used on our system since they have shown to yield the best overall performance in a wide range of situations.

5.3 Simulation Experiments

To get a quantitative assessment of the performance of our approach we performed a series of simulation experiments. In the first task the robot had to travel along a straight corridor without any unexpected obstacles. The second scenario was similar to that shown in Figure 4. The robot had to move along a corridor and had to enter a room. Figure 7 shows for 50 different runs the average time the robot needed to reach the target location. The error bars, which are not visible in the first three columns, indicate the $\alpha = 0.05$ significance level. As can be seen from the figure, both collision avoidance strategies show the same performance if the robot stays in the corridor. However, our approach requires significantly less time to complete the second task. Furthermore, our method

[1] More pictures and complete videos are available at http://www.informatik.uni-freiburg.de/~burgard/publications.html

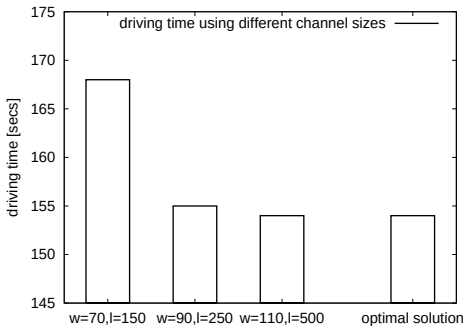

Figure 8: *Average time needed to reach a goal location for different sizes (w=width and l=length).*

yields only a small variance of the overall completion time. The DWA, in contrast, shows a quite high variance, which comes from the necessary retries to enter the room.

Figure 8 shows the time needed to complete a series of navigation tasks using channels of different size. The last column is the time needed by the optimal solution, i.e., if the whole state space is explored. As can be seen, the results obtained with channels of 500 cm length and 110 cm width are as good as the optimal solution.

In a further simulation experiment we investigated the performance of our method in situations like that shown in Figure 1. In contrast to the DWA, our method decelerates the robot early enough so that the system is able to turn into the corridor immediately [16].

6 Conclusion

In this paper we presented an integrated approach to sensor-based collision avoidance and path planning for mobile robots in dynamic environments. Our system plans in the full $\langle x, y, \theta, v, \omega \rangle$ configuration space and therefore takes into account the kinematics of the robot. The key advantage of our approach is that it also performs a lookahead in the velocity space. Accordingly the robot can decelerate early enough which is highly important especially in narrow environments.

Our algorithm includes several techniques to deal with the complexity of the induced search problem during replanning. The overall system is highly efficient and can be run on a standard PC. It automatically adapts itself to the performance of the underlying processor and to the complexity of the search problem.

The approach has been implemented and tested on different robotic systems. In all experiments our technique was able to generate safe trajectories. We compared our approach to the popular DWA technique. The experiments demonstrate that our algorithm yields more efficient trajectories which are often close to the optimal ones.

References

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

[2] O. Brock and O. Khatib. High-speed naviagation using the global dynamic window approach. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1999.

[3] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1-2), 2000.

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

[5] D. Fox, W. Burgard, and S. Thrun. A hybrid collision avoidance method for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.

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

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

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

[9] N. Y. Ko and R. Simmons. The lane-curvature method for local obstacle avoidanc. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1998.

[10] K. Konolige. A gradient method for realtime robot control. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2000.

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

[12] H.P. Moravec and A.E. Elfes. High resolution maps from wide angle sonar. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 116–121, 1985.

[13] C. Schlegel. Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1998.

[14] K. Schmidt and K. Azarm. Mobile robot navigation in a dynamic world using an unsteady diffusion equation strategy. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1992.

[15] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1996.

[16] C. Stachniss. Zielgerichtete Kollisionsvermeidung für mobile Roboter in dynamischen Umgebungen. Master's thesis, University of Freiburg, Department of Computer Science, 2002. In German, to appear.

[17] 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.