Local Action Planning for Mobile Robot Collision Avoidance

Thorsten Belker Dirk Schulz

Department of Computer Science III, University of Bonn, Germany

Abstract

This paper presents an approach to collision avoidance for mobile robots that uses local path planning within a map built from the robot's latest range measurements followed by a search for control commands to steer the robot towards the goal safely and efficiently. While the search for control commands allows to take the robot's dynamics into account and therefore allows for very smooth navigation behavior, planning is able to determine the optimal path towards the goal and minimizes the likelihood that the robot gets trapped in dead end situations. We introduce and compare several approaches, to utilize the results of path planning in the search for good control commands. In extensive experiments we show that our combined approach outperforms a collision avoidance that is purely based on local planning and we empirically investigate the pros-and-cons of the different search strategies for control commands and find one to significantly outperform the others.

1 Introduction

Collision avoidance is a crucial part for all autonomous mobile robots operating in populated and dynamic environments. In order to navigate savely, they must be able to react to unforeseen obstacles like humans crossing their paths. For this reason, virtually all navigation systems comprise some kind of collision avoidance component that controls the robot's motion.

In general, it is also desired that the robot reaches its destination fast. It should therefore take the robot's dynamics into account and move the robot towards the target location most effectively. The optimal approach would be to do an exhaustive search for the sequence of control commands that achieves the shortest time to target. However, this approach is computationally expensive and not suited to issue safe control commands at high frequency. An exhaustive search in the space of control commands is therefore not feasible in order to be able to navigate at high speeds.

Many collision avoidance systems use purely reactive approaches which allow for high update rates. They decide on the next control command solely based on different heuristics. However, as no planning is involved, these approaches risk that the robot gets trapped in dead end situations like for example u-shaped obstacle configurations.

In this paper we present an collision avoidance approach that uses local path planning followed by a search for the best next motor control command given the current plan. Local path planning finds the shortest path within a map built from the robot's latest range measurements, and can be carried out much faster then a search for the optimal sequence of control commands. That way, we reduce the risk of getting stuck in local minima and still achieve the high update rates required for fast navigation.

The crucial aspect of this hybrid reactive approach of course is, how to decide on the control commands based on a plan. A simple approach would be to stick to the planned path as close as possible. However, as path planning does not respect the robot's dynamics, this can result in poor performance. In this work, we consider path planning systems that assign a utility value to each state in the state space. For this purpose, we formalize the navigation problem as a Markov Decision Process (MDP) which allows us to use dynamic programming algorithms to efficiently compute optimal utility functions for it.

We introduce different evaluation functions that employ the utility function computed in the planning step to evaluate control commands. We compare the performance of these functions in extensive experiments carried out with a differential drive robot within our office environment. In these experiments we find one evaluation function to significantly outperform the other evaluation functions, especially the one that models a path following algorithm.

The remainder of this paper is organized as follows. After discussing related work in Section 2, we describe how to compute optimal utility functions that can be used to evaluate control commands in Section 3. Section 4 describes in detail, how the evaluation of the performance of control commands is based on the utility function computed in the planning step and we introduce the different evaluation functions we have developed. In Section 5, we give results on the performance of our collision avoidance method using the different evaluation functions.

2 Related Work

Most existing collision avoidance methods are purely reactive in the sense that they search for safe robot control commands based on the robot's current proximity sensor data without any projection of the robot's future state. These methods differ in the way this search is carried out. The Vector Field Histogram method [1] for example decides on the next movement direction and the speed of the robot based on an angular histogram, which describes the density of obstacles in the surrounding of the robot. Potential field methods [7] achieve collision avoidance behavior by simulating repulsive forces exerted from obstacles and an attractive force towards the target. Both approaches do not explicitly take the constraints imposed by the dynamics of the robot into account. Koren and Borenstein [6] identify two major problems of potential field approaches: They often fail to find trajectories between closely spaced obstacles and can produce oscillatory behavior in narrow passages. Behavior-based navigation system are composed of at least two behaviors. An approach target behavior and a collision avoidance behavior. While arbitration shemes based on voting, are often not able to take dynamic constraints into account, a decision-theortic arbitration scheme recently proposed by Rosenblatt [8] is.

Another popular method, which is more closely related to our approach is the dynamic window approach to collision avoidance [3, 10, 9]. This method searches for the trajectory the robot should take within the next time step based on a local map of the robot's surrounding built from the latest sensor measurements. A trajectory is specified by the translational and rotational velocity of the robot and the search is carried out in this velocity space. In order to reduce the search space, the robot's dynamic constraints are taken into account by considering only velocities which can be reached within the next time interval. The approach decides on the best next velocities based on a linear evaluation function which weighs clearance, heading towards the target and speed. Our method adopts this search method, but uses new evaluation functions based on the computation of optimal utility functions from a local map.

The major disadvantage of all purely reactive approaches is that they might get stuck in local maxima of the evaluation function. Therefore, more recently, methods have been developed, which use local planning to overcome this problem. Ulrich and Borenstein [11] describe an extension of the Vector Field Histogram method which carries out A-star search on a local map in order to obtain the best sequence of movement directions towards the target location. Konolidge [5] uses dynamic programming on the local map to compute the gradient towards the target. To make this approach computationally feasible only the two dimensional space of possible robot positions is considered for planning. This results in optimal paths with respect to some cost function but does not allow to model the robot's dynamics correctly. In this paper we suggest a method that simultaneously plans optimal collision free paths and takes the dynamics of the robot into account. Like Konolige we use a path planner based on dynamic programming to compute a utility value for each state in the state space, but rather than computing an optimal path from the utility function using gradient ascent, we directly use the utility function to evaluate control commands. As our experiments show, our method is able to produce a smoother navigation behavior than a path following algorithm, because it makes a better use of the computed utility function.

Brock and Khatib [2] developed a similar method for holonomic mobile robots. They compute a navigation function which labels each cell in the local grid map with the L^1 distance to the goal. They replace the term for the target heading and the clearance term in the evaluation function of the dynamic window approach by two features derived from that navigation function. Though the computed navigation function is guaranteed not to have local optima, the combined evaluation function is not. In our approach, however, the evaluation function is directly derived from the utility function and thus guaranteed not to have any local optima.

3 Computing Optimal Utility Functions

In this section we discuss how to compute optimal utility functions for navigation problems. For this purpose, we introduce Markov Decision Processes (MDPs) which can model path planning problems very elegantly.

Markov Decision Processes (MDPs) provide a general framework for the specification of simple control problems where an agent acts in a stochastic environment and receives rewards from its environment. A solution to an MDP is an optimal utility function V^* which assigns a utility value to each state in the state space. An MDP is given by

- a set of states S,
- a set of actions A,
- a probabilistic action model $P(S|S, A)$ and
- a reward function $R : S \times A \to \mathbb{R}$.

$P(s'|s, a)$ specifies the probability that action a taken in state s leads to the state s'. $R(s, a)$ denotes the immediate reward gained by taking action a in state s. The property that action effects only depend on the last action and the state in which they are executed is called the Markov property.

The utility of being in state s is given by the following formula:

$$V^*(s) = \max_{a \in A}(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a)V^*(s')).$$

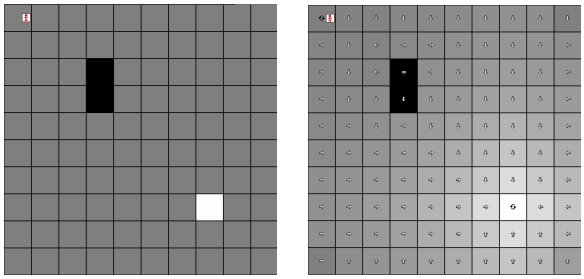

Figure 1: *Left: The navigation problem. Right: the optimal navigation policy computed by value iteration.*

The utility of being in state s (when always performing the best possible action) is thus given by the immediate reward for executing the optimal action a in state s plus the expected discounted future reward. The constant γ is called discounting factor and weighs the expected future rewards with respect to how far in the future they will occur. This formula can be exploited to compute the optimal value function V^* using dynamic programming techniques. Kaelbling et al. [4] give a detailed discussion of MDPs and POMDPs, a generalization of MDPs where the state is not fully observable.

The utility function V^* can be used to select the optimal action $\pi(s)$ in each state s according to the Bellman equation:

$$\pi(s) = \text{argmax}_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')].$$

The function π is called policy and assigns to each state $s \in S$ the action $a \in A$ with the highest expected utility.

For illustration, Figure 1 shows a simple grid world navigation problem (the obstacles are marked black while the goal is marked white) together with the optimal navigation policy for this problem. The gray scale of each cell visualizes the utility of this state.

Given a navigation policy it is straight forward to compute a path to the goal using gradient ascent. However, this leaves us with the problem of how to compute a sequence of motor control commands that achieve the intended navigation behavior. In the following we suggest to use the utility function V^* directly to evaluate control commands.

4 Evaluation Functions for Control Commands

In this section we describe how we search for admissible control commands which in the case of synchro-drive robots can be approximated by simple trajectories. In the second part of this section we examine how these trajectories can be evaluated using utility functions computed from MDPs.

4.1 Searching for Control Commands

When using synchro-drive robots each control command is given by a pair (v_o, ω_o), the target translational and rotational velocity of the robot. When we assume that the robot immediately reaches the new velocities and therefore ignore boundaries on the maximal positive and negative accelerations, we can model these control commands as simple trajectories: (v_o, ω_o) corresponds either to a circular trajectory (if $v_o > 0 \wedge \omega_o \neq 0$), to a straight line trajectory (if $v_o > 0 \wedge \omega_o = 0$) or to a rotation (if $v_o = 0 \wedge \omega_o \neq 0$). As Fox et al. show in [3] the error we make under this assumption when predicting the position (x, y) of the robot after Δt seconds is bounded by $\sqrt{2(\Delta v)^2(\Delta t)^2} = \sqrt{2}\Delta v \Delta t$ where $\Delta v = |v_t - v_{t+\Delta t}|$. We can account for this error by thickening the robot and by keeping the time between two successive control commands small.

If we model control commands and their effects using these simple trajectories, cutting algorithms can be used to check whether a trajectory is admissible, that is whether the robot can stop safely before the next obstacle on the trajectory. It is also straight forward to project the state $(x', y', \theta', v', \omega')$ after $\Delta t'$ seconds given the initial state $(x, y, \theta, v, \omega)$ and the trajectory (v_o, ω_o). Assuming that (v_o, ω_o) can be reached in Δt seconds, the projection error is bounded as before. The projected state of the robot after $\Delta t'$ seconds can - as we will discuss in the following - be used to evaluate control commands.

To select a suitable control command we have to search for admissible control commands in the space of all possible velocity combinations and then select the one with the highest evaluation with respect to some given evaluation function. To make this search feasible, only the small window of this space is considered that is given by the dynamic constraints of the robot, namely its maximal velocities and its maximal (positive and negative) translational and rotational accelerations together with its current velocity. Given a small constant time Δt, the time the robot needs for one iteration of the algorithm, these constraints limit the possible velocity combinations (v, ω) the robot can reach within this time window.

4.2 Evaluating Control Commands

Under all the admissible trajectories in the window of reachable velocities one has to be selected according to an evaluation function G. Fox et al. [3] propose to use the class of functions

$$G(v,\omega) = \sigma\left(\alpha \text{head}(v,\omega) + \beta\, \text{dist}(v,\omega) + \gamma\, \text{vel}(v,\omega)\right)$$

Here $\text{head}(v,\omega)$ is $180 - \angle(v,\omega)$ where $\angle(v,\omega)$ is the angle to the target after executing action (v,ω) for Δt seconds, $\text{dist}(v,\omega)$ is the distance to the closest obstacle on the trajectory and $\text{vel}(v,\omega)$ is a projection on the translational velocity v. The function σ is a smoothing function

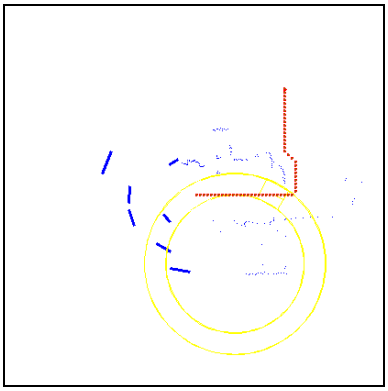

Figure 2: *A local map of the robot's surroundings.*

intended to increase the side-clearance of the robot. In the evaluation function head(v,ω) and vel(v,ω) take the role of progress estimators, while dist(v,ω) accounts for the future utility of a trajectory.

However, this class of evaluation functions in general have local maxima which means that the robot might get stuck in situations like u-shaped obstacle configurations. In the following we will discuss how an optimal utility function V^* can be used to evaluate trajectories. We start by considering a quite simple MDP with a two dimensional state space where the robot's state is only determined by its position, a deterministic action model and a distance-based reward function. After that we proceed by considering more complex reward functions which can take the robot's clearance into account as well and we consider three dimensional state spaces where besides the robot's position its orientation is part of its state.

We compute the state space of the robot by discretizing the local map (obstacle field) of the robot which is build from its last sensor readings only into quadratic cells. In our implementation we used a map of $10\,\text{m} \times 10\,\text{m}$ and a resolution of $10\,\text{cm}^2$. Figure 2 shows such a map. Each cell that contains a sensor reading is marked as a wall cell. The walls are then thickened which allows to treat the robot as a point while doing path planning. To compute the optimal utility of each state s we assume a deterministic action model where the robot can translate to each neighboring grid cell. The reward for an action a executed in state s is $r(s,a) = -\text{cost}(a) = -\text{dist}(m(s), m(\text{succ}(s,a))$ where $s(s,a)$ is the successor state of s when executing action a, $m(s)$ is the midpoint of the grid cell s and dist(p,p') denotes the euclidian distance between points p and p'. (With a resolution of 10cm^2 $r(s,a)$ is thus either -10 or $-\sqrt{200}$).

As we use a deterministic action model we can use Dijkstra's algorithm to compute $V^*(s)$ for each cell s. To further speed up the algorithm we do not consider wall cells for expansion. To evaluate trajectories we project the state $(x', y', \theta', v', \omega')$ after $\Delta t'$ seconds given the initial state $(x, y, \theta, v, \omega)$ and the trajectory (v_o, ω_o), map (x, y) to a state s in the MDP's state space and assign the utility $v(s)$ to (v_o, ω_o) where $v(s)$ is computed from V^* by distance-weighted linear interpolation using all eight neighbors of s_i including s. We call this evaluation function *distance-based evaluation function* (DEF2D).

An obvious extension of the MDP considered so far is to make the reward for an action a in state s not only dependent on the cost for executing action a, cost(a), but on the reward for being in state succ$(s,a) = s'$ which results from executing a in s: $r(s,a) = -\text{cost}(a) + \text{reward}(succ(s,a))$ It is reasonable to make reward(s') dependent on the clearance of the robot in state s'. In our implementation we have used the following formula: reward$(s') = -\alpha \max(0, \theta_{cl} - \text{clearance}(s'))$ where θ_{cl} is a threshold value specifing when the robot has enough clearance. In the experiments we have used the parameters $\alpha = 10$ and $\theta_{\text{cl}} = 60$. Because of the deterministic action model, we can again compute an optimal value function for the MDP using Dijkstra's algorithm. We interpolate the values for this evaluation function like for DEF2D.

In this case, however, the evaluation of a trajectory should not only depend on the projected state after $\Delta t'$ seconds, but on the projected states after $\frac{1}{k}\Delta t', \frac{2}{k}\Delta t', ..., \frac{k}{k}\Delta t'$. We obtain the evaluation as the average evaluation of all the projected states at these times. We call this evaluation function the *distance- and clearance-based evaluation function* (DCEF).

One problem with the evaluation functions developed so far is that they do not support turns in place. As the evaluation of a trajectory only depends on the projected position (x, y) after $\Delta t'$ seconds and this position does not change for a pure rotation, we cannot model that the robot should turn in place in some situations. To be able to do so, we have to introduce an additional dimension to the state space: the robot's orientation. To keep things feasible the orientation has to be discretized. In the third evaluation function we have considered four discrete orientations and the three actions: forward translation, left turn and right turn. By assigning different costs to forward translations (in the experiments: 10) and turns (in the experiments: 30) we can bias the robot to prefer to turn in place rather than make some translation. We will call this evaluation function *distance evaluation function 3D* (DEF3D).

For our experiments we have developed a fourth evaluation function: the *path following evaluation function* (PFEF). It takes the projected position (x', y') of the robot after $\Delta t'$ seconds given trajectory (v_o, ω_o) and assigns $-\infty$ to the trajectory if the closest point on the shortest path to the goal is more than max_{dev} cm (in the experiments $\text{max}_{\text{dev}} = 50.0$) away from (x', y') and DEF2D (v_o, ω_o) in any other case. This evaluation function simulates the computation of robot control commands from a planned path to the goal.

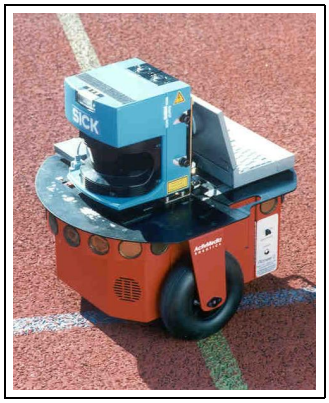

Figure 3: *The Pioneer II platform*

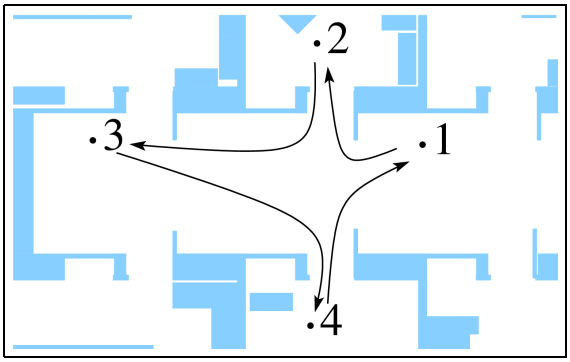

Figure 4: *The experimental setup.*

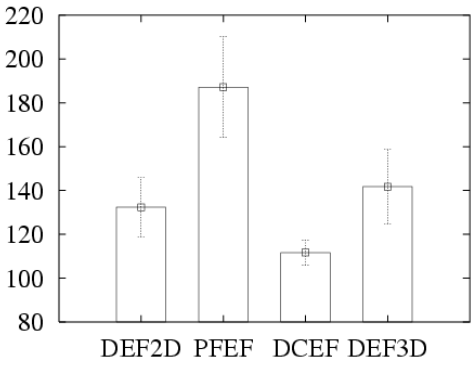

Figure 5: *The time it took the collision avoidance module to complete a sequence of four navigation tasks using different evaluation functions. the mean.*

5 Experimental Results

In this section we describe the experiments we have carried out in order to compare the performance of the four evaluation functions introduced in the previous section and discuss the results.

The experiments have been carried out using a Pioneer II platform equipped with a laser range finder as shown in Figure 3. In the first experiment the robot repeatedly executed planned paths between four goal positions which are depicted in Figure 4. Here, the path planner randomly generates intermediate goal points for the collision avoidance which are between 1m and 5m ahead on the planned path. Note, that this is a quite challenging setup to test a collision avoidance algorithm as the goal points tend to be quite far away. Using the evaluation function described in [3] in this setup, we experienced a lot of situations where the robot could not reach the target point at all. We draw the distance to the next target point on the path randomly to ensure that the robot faces a lot of different situations. Please also note that we performed our experiments in a populated university office where humans cross the robot's way occasionally. The sampling of target points introduces a lot of variance in the robot's behavior. Therefore each course of the experiment consisting of the four navigation tasks shown in Figure 4 was repeated 12 times. Figure 5 shows the average time it took the collision avoidance system to complete the course using the four evaluation functions together with the 95% confidence interval of the mean.

As we can see from Figure 5, DCEF is significantly better than all other evaluation functions. For example it is 15% better than DEF2D, the second best evaluation function. DEF2D and DEF3D do not differ significantly in the experiment. PFEF is by far the worst evaluation function and significantly worse than the other three functions.

It is not surprising that the robot performs better using DCEF than using DEF2D alone as DCEF takes the side clearance of the robot as well as the progress towards the target into account. In a second experiment, we drove the robot up and down the corridor of our department building with a speed of up to 100 cm/s where the corridor is blocked by two unknown obstacles. In Figure 6 you can see typical traces produced by these two evaluation functions during this experiment. As illustrated in the figure, DEF2D tends to come closer to obstacles and then has to reduce its velocity. In the figure the robot's current velocity is indicated by the width of the light grey background of the robot's trajectory. In additional simulator experiments using the same setting, DCEF on average required 11.4% less time for completing a sequence of four navigation tasks than DEF.

Surprisingly, it turned out that DEF3D is not significantly better and in fact even slightly worse than DEF2D. However, this can be explained by the very coarse model used, and in addition, the evaluation of this function takes more than 1 second per iteration using our current implementation rather than about 0.25 seconds with DEF. As the robot adapts to these long update times, it is in general still possible to navigate reliably, but the robot drives more carefully. However, for high speed navigation in the hallway the update interval for DEF3D is much too large and results in unreliable navigation behavior. Here DEF2D clearly outperforms DEF3D, though not for prin-

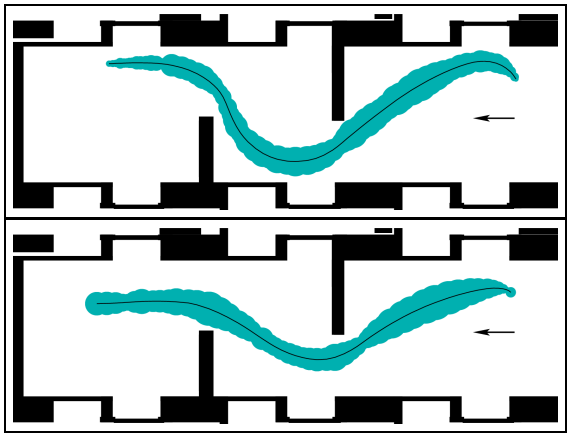

Figure 6: *Paths produced by DEF2D and DCEF.*

cipal reasons but due to the speed limitations of our computers.

PFEF performs significantly worse than the other evaluation functions. This is caused by the type of map used for planning. We use a local map of the robot's environment that is built from one sensor reading only. Due to occlusion, the map can change considerably during a short time period leading to a new shortest path which differs drastically from the previous one. This can lead to unstable navigation behavior when using PFEF. Figure 7 demonstrates this effect. After a small rotation of the robot the shortest path to the target point has changed completely.

Conclusion

In this paper we have presented an approach to collision avoidance for mobile robots that uses local path planning within a map built from the robot's latest range measurements followed by a search for control commands to steer the robot towards the goal safely and efficiently. While the search for control commands allows to take the robot's dynamics into account and therefore allows for very smooth navigation behavior, planning is able to determine the optimal path towards the goal and minimizes the likelihood that the robot gets trapped in dead end situations. In extensive experiments we have compared four approaches to utilize the results of path planning in the search for good control commands and show that the *distance- and clearance based evaluation function* is significantly better than the other three evaluation functions, especially as the one that models a path following behavior.

Acknowledgments

The research reported in this paper is partly funded by the Deutsche Forschungsgemeinschaft (DFG) under contract number BE 2200/3-1.

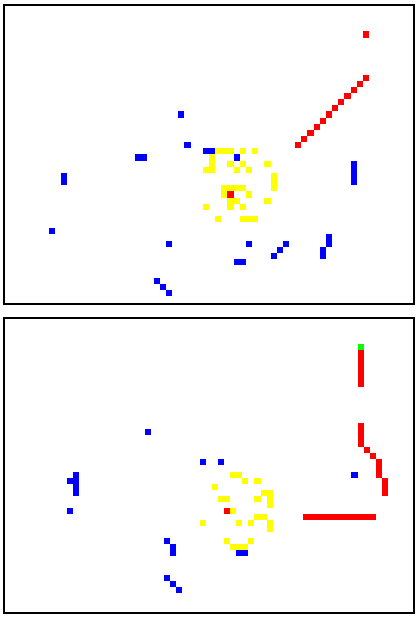

Figure 7: *The instability of paths.*

References

[1] J. Borenstein and Y. Koren. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 6 1991.

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

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

[4] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

[5] K. Konolige. A gradient method for real-time robot control. In *Proc. of IEEE/RSJ Conf. on Intelligent Robots and Systems*, 2000.

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

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

[8] J. Rosenblatt. *Maximizing Expected Utility for Optimal Action Selection under Uncertainty*. Autonomous Robots 9(1), pp. 17-25, 2000.

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

[10] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1996.

[11] Iwan Ulrich and Johann Borenstein. VFH*: Local obstacle avoidance with look-ahead verification. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2000.