Proceedings of the IEEE/RSJ International Conference on Intelligent
Robots and Systems (IROS 2002). Lausanne, Switzerland, Sept. 2002

Motion Planning through Policy Search

Nicholas Roy[†] and Sebastian Thrun[‡]

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

Abstract

We propose a motion planning algorithm for performing policy search in the full pose and velocity space of a mobile robot. By comparison, existing techniques optimize high-level plans, but fail to optimize the low-level motion controls. We use policy search in a high dimensional control space to find plans that lead to measurably better motion planning. Our experimental results suggest that our approach leads to superior robot motion than many existing techniques.

1 Introduction

Most mobile robot navigation systems to date have separated robot motion into two levels of control: global path planning and local motion control [3]. The global path planner is usually implemented in some low-dimensional space, such as the discretized x, y manifold of the robot's configuration space. The path in this space is then optimized using a number of techniques: dynamic programming is perhaps the most popular technique [17, 12] at present, in that it guarantees optimality with respect to the model of the robot and the environment, converges very quickly for most environments, and also provides a universal plan for the entire space. Other examples of these planners are potential-field based planners [9], and probabilistic roadmaps [8].

However, a plan in x, y space must still be converted into controls for the mobile robot. Note that most global path planning methods cannot be used to plan directly in the full state space of the robot, because the state space of a mobile robot is at least 5 dimensional (x, y, θ, v_t, v_r, where v_t is the translational velocity and v_r is the rotational velocity). Many different local controllers have been suggested in recent years [2, 18, 6, 10]. However, the local controllers have all abandoned optimality in the local space, in exchange for adapting to dynamic obstacles and speed of convergence. Existing local controllers make approximations such as modelling the robot's shape with a bounding circle [13], (eliminating the θ dimension), deterministic motion models [16], etc. The controllers then use a variety of heuristics for finding some path that approximates the trajectory suggested by the global planner[1]. The disadvantage to these approaches is that the high-level structure of the path may be close to optimal, but controls may be suboptimal on a detailed level. This sub-optimality is usually manifested as overly-conservative motions, such as passing through doors.

By comparison, our approach is to generate a complete plan in the full state space of the robot. We first generate an intermediate plan in a low dimensional, discrete space by using dynamic programming over a value function. We then project this plan to the full state space that includes orientation and velocity as state features. The plan is represented as a sequence of waypoints and a controller that drives through the waypoints in sequence. Using the high-dimensional projection of the value function plan as a seed, we then perform gradient ascent in the plan space, modifying the waypoints to increase the expected reward of the plan. Under a number of simple and physically motivated assumptions, we will demonstrate that this approach finds paths where conventional means fail, and does so in a realistic amount of time.

The central assumption of our approach is that the value function generates an initial plan that is "close to" the optimal solution, and provides a reasonable starting point. For our mobile robot navigation problem, we assume that the value function provides a topologically correct solution, which is refined by the search. We do not provide any guarantees on the quality of the final plan if the initial value function policy is allowed to be arbitrarily bad. However, we provide empirical evidence that this approach is successful for real world domains.

2 Related work

One of the earliest local controllers that mediates between global plans and local constraints was the Vector Field Histogram approach [2] and its successor VFH+ [18]. This approach uses a form of potential based histograms of range measurements to determine appropriate headings. Fox et al. [6] use the dynamic window method for converting high-level plans into local controls, while accounting for dynamic obstacles. Ko and Simmons [10] used a lane-based method for local planning around dynamic and unmodelled obstacles. Konolige [12] demonstrates an approach that is closer to our approach in spirit,

[1]Only recently have global path planners been able to replan fast enough to allow for dynamic obstacles in the environment[12].

in that he uses dynamic programming for generating a path that obeyed more of the kinematic constraints of the mobile robot, replanning as necessary after every control. However, this approach still contains approximations of the mobile robot as a point, and also does not respect the dynamic constrains of the robot.

Davies et al. [5] propose an algorithm that is very similar in spirit to our mobile robot application; however, while they project a low-dimensional approximate value function solution to a higher space, they do not explicitly search for policies that minimize expected reward. Instead, they use a heuristic search to find an admissible trajectory for a deterministic agent. In contrast, our work allows a noisy action model, and will not only find an admissible path, but will attempt to find the best path according to some objective function.

Path planning has been posed as a form of traditional control in some preceding work. Koditschek described navigation control in terms of navigation functions for navigating a mobile robot [11], however, this approach is difficult to scale to more dimensions, and partially observable settings. Laumond et al. also described a control-theoretic approach for local/global control [7], however, this approach suffers from the same limitations as non-control-theoretic mixed-level approaches. In general, control-theoretic approaches are best applied to generating steering controls in the absence of obstacles. Extending the existing approaches to handling obstacles in more than two or three dimensions remains an open problem for the control community.

3 Markov Decision Processes

A Markov decision process (MDP) describes a problem where an agent, such as a mobile robot, must take actions in a world to maximize the expected reward. The model is Markovian if the current state is sufficient to predict the next state, conditioned on the action. We model the outcome of each action as non-deterministic, with some probability distribution. However, we make an assumption that the true next state is fully observable after the action, and also that the next state is independent of all but the most recent state and action. An MDP is described by the tuple $(\mathcal{S}, s_o, \mathcal{A}, \mathcal{T}, \mathcal{R})$ where $\mathcal{S}$ is the state space, s_o is the initial state and $\mathcal{A}$ is the set of actions available to the agent. The transition matrix $\mathcal{T}$ describes how the state changes with actions: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0,1]$ such that $T(s,a,s') = p(s'|s,a)$, the probability of being in state s' starting from state s and taking action a. $\mathcal{R}$ is the reward given to the agent. In general, the reward function can be a mapping from states and actions to numerical rewards, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto [0, R_{max}]$, but for simplicity (and without loss of generality) we will assume the reward depends only on the current state.

The optimal solution to an MDP is a policy that describes which action to take as a function of the current state to maximize the reward over the lifetime of the agent. For the case of a navigating mobile robot, the optimal policy will correspond to the optimal path, given the current goal, from any point in the environment. Notice that a policy generates an expected trajectory for all possible start locations, and therefore is a more general problem than the standard motion planning problem.

3.1 The Value Function Approach

The goal of the agent in an MDP setting is to maximize its cumulative long-term expected reward. In the value function setting, a value is assigned to each state, where the value is a function of the policy (or path) and represents the long-term cumulative expected reward. The optimal policy is one that maximizes this value, and hence the expected reward at each state. The optimal value function is given by Bellman's equation:

$$V(s_i) = R(s_i) + \gamma \sum_{j=1}^{|\mathcal{S}|} V(s_j) \sum_{k=1}^{|\mathcal{A}|} p(s_j|\pi(a_k|s_i), s_i) \tag{1}$$

where $\pi(a_k|s_i)$ is the current policy and γ is the discount factor that determines the contribution of future reward to the current state value. For the rest of this work, we assume (again, without loss of generality) an undiscounted ($\gamma = 1$) finite-horizon problem with some termination state. Furthermore, we will assume a deterministic policy, $\pi(a_k|s_i) = \{0, 1\}$.

Value iteration finds the optimal policy by computing the value function for every state based on some policy, and then iteratively updating the policy until the value is maximized over every state. The nature of value iteration highlights its computational cost – a mapping for every state to some value and action is preserved, and iteratively updated. In order to find a policy for a continuous, high-dimensional problem, we need a different, more compact representation of a policy, and a way to evaluate policies in this representation.

3.2 Policy search

Policy search operates by evaluating the long term cumulative reward of the current policy. Gradient-ascent search estimates the gradient of the reward and adjusts the policy parameters to increase the expected reward until a maximum is reached. In order to search the policy space, we need a parameterized, continuous policy representation. In addition, the need to map the low-dimensional policy into a high-dimensional continuous space has implications for our choice of policy representation. There are a number of different choices of policies, e.g., Bayes nets [14], neural nets [1], etc. that take continuous-valued state features and return some control action. However, a mapping from a discrete grid world to these complex representations is not easy.

Instead, we represent the policy as a series (that is, an ordered set) of waypoints w_j, and an associated local controller. The controller emits actions, where each action moves the agent towards the next (closest) waypoint. The series of waypoints can be viewed as a forming an approximately piece-wise linear path from the start state to to the goal. At execution time, the next action a at time t is a function of the current state $s(t)$ and the next waypoint $\mathbf{w}$,

$$a = (\Delta v_t, \Delta v_r) = f(s(t), \mathbf{w}_j). \qquad (2)$$

The policy has an additional free parameter $d_{approach}$ that determines when the controller switches to the next waypoint, as in

$$\mathbf{w}_j = \mathbf{w}_{j+1} \text{ iff } ||s(t) - \mathbf{w}_j|| < d_{approach}. \qquad (3)$$

It should be noted that the principle sacrifice made by this type of policy is that the final policy is no longer an optimally universal plan. Policies computed from the optimal value function have the desirable property that the policy describes the optimal action for every possible state. In comparison, the policy that found using our approach exhibits this property close to the expected trajectory, but the quality of the policy can become arbitrarily bad as the actual path deviates from the expected path. However, if the model of the local controller accurately reflects the true next state distribution, then result of the policy search will be a policy for those states that truly matter.

3.3 Initial Policy Estimate

In order to search for the best policy (or expected path) efficiently and avoid problems such as local minima, we want an estimate of a good path to begin the search. The value function gives a reasonable path in the low-dimensional state space, so it remains to convert the low-dimensional policy into a continuous, high-dimensional path. We do this by first extracting the set of maximum likelihood states that describe the expected trajectory in the lower space. We project these states into the higher-dimensional space to generate a large set of waypoints that describe the expected path. This set of projected waypoints is unnecessarily dense, so we prune by merging path neighbors that share the same value function policy, as shown in figure 1. The endpoints of these line segments represent the policy in the high dimensional space, and this policy is used as the initial estimate for the search.

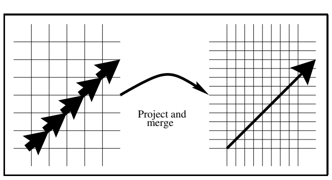


Figure 1: *Projecting the value function policy on the left to the higher dimension space on the right, we merge neighboring states with the same policy into a single line segment. The endpoints of the line segments constitute the initial estimate for policy search.*

3.4 Computing the policy value

The optimal policy π^* is one that maximizes the expected reward over the execution of the policy from the start state s_0 to the goal state,

$$\begin{aligned} V_{\pi^*}(s) &= \max_{\pi \in \pi} V_\pi(s_0)) \\ &= \max_{\pi \in \pi} R(s_0) + \int_{|S|} p(s(dt)|s_0, \pi(s_0)) V_\pi(s(dt)) ds. \end{aligned} \qquad (4)$$

where $\pi(s_0)$ is the action dictated by the policy at state s_0, $s(dt)$ is the next state after time dt, and the integral is simply the expectation reward of the next state distribution after time dt (cf. equation (1) in continuous form).

We make a final simplifying assumption that the local controller is unbiased. That is, the local controller keeps the next state distribution centered on the expected trajectory, with some distribution $G(s(t), \theta)$ where θ represents the parameters of the local controller. This dictates that the result of each action will have maximum likelihood on a line between w_j and w_{j+1}, which makes computing the expected distribution easier.

We can therefore modify the integral in equation (4) by transforming the next state distribution $p(s(t)|s_0, \pi(s_0))$ into a function centered on the maximum likelihood next state $G(s; s(t), \theta) : \mathcal{S} \mapsto [0, 1]$. If we write $G(s)$ and $V(s)$ in matrix form, we can simplify equation (4) to

$$V(\pi) = R(s_0) + G(s(dt), \theta) \cdot V(s(t)) \qquad (5)$$

where $s(dt)$ is the maximum likelihood next state after time dt under policy π. From here onwards, $G(s(dt))$ is assumed to be the agent's distribution over the state space, the parameters θ being assumed to be constant.

If we further constrain the policy $\pi(s_i)$ to be a described by a series of n waypoints $\mathbf{w}_1 ... \mathbf{w}_n$ in some Euclidean space, we can expand recurrence equation 4 to

$$\begin{aligned} V(\pi_n) &= \int_{t=0}^{T} G(s(t)) \cdot R(s(t)) dt && (6) \\ &= \sum_{j=0}^{n-1} \int_{s(\mathbf{w}_j)}^{s(\mathbf{w}_{j+1})} G(s) \cdot R(s) ds. && (7) \end{aligned}$$

We maximize equation (7) by differentiating and travelling along the positive gradient. Differentiating gives

$$\begin{aligned} \nabla V(\pi_n) &= \frac{\partial}{\partial \mathbf{w}_{1...n}} \sum_{j=0}^{n-1} \int_{s(\mathbf{w}_j)}^{s(\mathbf{w}_{j+1})} G(s) \cdot R(s) ds && (8) \\ &= \sum_{j=1}^{n-1} \frac{\partial}{\partial \mathbf{w}_j} \left(\int_{s(\mathbf{w}_j)}^{s(\mathbf{w}_{j+1})} G(s) \cdot R(s) ds \right) && (9) \end{aligned}$$

Note that in the case of purely deterministic action models (i.e., a perfect controller), we could drop the $G(\cdot)$ term and simply integrate the reward along the expected path. By integrating the expected reward with respect to $G(\cdot)$, we capture the noise inherent in the local controller, subject to the assumption of no bias. The larger the noise in the local controller, the more conservative the eventual policy.

3.5 Determining policy size

The initial policy estimate provided by value function will consist of some number n of waypoints. However, the optimal policy may require an arbitrary number of waypoints. Consequently, some procedure is required to introduce new waypoints as needed. The number of waypoints needed to represent the policy is similar in concept to the problem of estimating the number of clusters used during Expectation-Maximization, and consequently we borrow a popular split-and-merge technique [15]. We perform policy search to convergence, and then consider inserting a new waypoint where appropriate. The algorithm terminates when the policy value does not increase with any inserted waypoint. The heuristic we use for inserting waypoints is to do so when the immediate reward between waypoints drops below the immediate reward at the waypoints:

$$\begin{aligned} \text{Insert } \mathbf{w}'_{j+1} \quad & \text{if} \quad R(\mathbf{w}'_{j+1}) < R(\mathbf{w}_j) \\ & \text{and} \quad R(\mathbf{w}'_{j+1}) < R(\mathbf{w}_{j+1}). \end{aligned} \tag{10}$$

Figure 1 summarizes our path planning algorithm.

Table 1: *Algorithm summary*

1. Run the dynamic program and extract a policy from 2-dimensional value function (Section 3.1)
2. Determine the maximum likelihood trajectory and convert to a set of 5-dimensional waypoints to form the expected path (Section 3.3)
3. *Policy search*: for each waypoint w_j
   (a) Determine value contribution of trajectory from previous waypoint w_{j-1} to next waypoint w_{j+1} (Section 3.4)
   (b) Measure gradient at endpoints
   (c) Move waypoint w_j along gradient until path segment value increases
   (d) Repeat for all waypoints until convergence
4. *Add new waypoints* (Section 3.5)
   (a) Find lowest immediate reward along the trajectory
   (b) Insert a new waypoint
   (c) Repeat step 3.
5. Repeat waypoint insertion until convergence

4 Mobile Robot Navigation

We demonstrate our approach on a mobile robot navigating in an indoor environment. Figure 2 shows the mobile robot Pearl interacting with elderly people in their assisted-living residence. The nature of this particular environment emphasizes the need for optimal policies; in confined quarters, with slower and less agile humans in the surroundings, control policies that make approximations about the shape and orientation of the robot become increasingly brittle. Most importantly, as has been stated already, value function planning in a more exact state space is actually computationally intractable.

We represent the robot's state at time t by a 5-tuple $s(t) = (x(t), y(t), \theta(t), v^t(t), v^r(t))$ where x, y gives the position of the robot, θ the orientation. and v^t, v^r are the translational and rotational velocities. We assume that the state space is continuous in all dimensions. The kinematic model of the robot coupled with the local controller gives the next state distribution $G(s)$. The goal of the

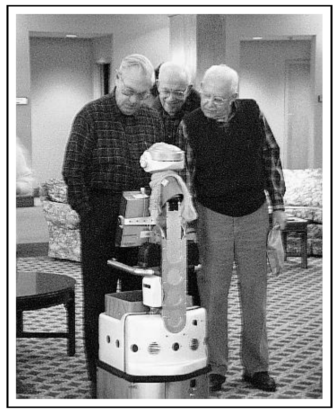
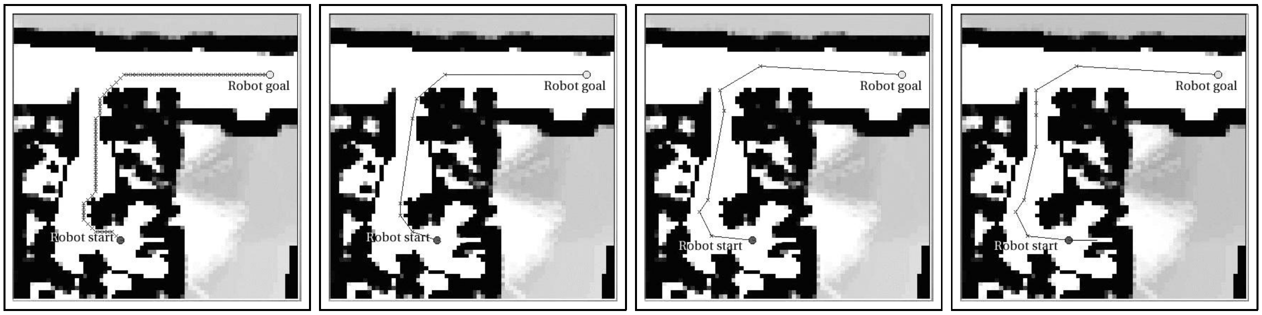


Figure 2: *Pearl, the nursebot, interacting with residents of a health care facility.*

robot is to maximize its expected reward; the navigation problem is designed such that the robot incurs some small negative reward $R = (-c \cdot d)$ for traveling a distance d, and receives some large positive reward g when it reaches the goal. Furthermore, the robot receives an large negative reward $R = (-h \cdot d)$ for attempting to travel a distance d when an obstacle is closer than a certain safety range.

By allowing the reward function to depend on the orientation and velocity of the robot, as well as the current location, we can capture the reward's dependence on the shape of the robot and the dynamics. The reward function can penalize the robot for some orientations and not others for the same position; similarly, the reward function can encode the dynamics of the robot by penalizing attempted velocities which have a high expectation of hitting nearby obstacles.

We can use knowledge about the state space to compute the gradient terms $\frac{\partial}{\partial w_j}$ approximately. The reward function seeks to minimize travel time, while also minimizing the likelihood that the robot will hit an obstacle. The expected travel time at any point in the trajectory is a function of the direction to the previous waypoint $(\mathbf{w}_{j-1} - \mathbf{w}_j)$ and next waypoint $(\mathbf{w}_{j+1} - \mathbf{w}_j)$. The like-



(a) (b) (c) (d)

Figure 3: *(a) Dynamic program solution; (b) Projected DP solution; (c) Policy search of DP solution; (d) With additional waypoints. The black areas are obstacle, and the white area is free space.*

lihood of hitting an obstacle is a function of the direction and distance to the nearest obstacle, $\zeta(\mathbf{w}_j)$. We compute the gradient as

$$\nabla V(\mathbf{w}_j) = ((\mathbf{w}_{j-1} - \mathbf{w}_j) + (\mathbf{w}_{j+1} - \mathbf{w}_j)) - (\zeta(\mathbf{w}_j) - \mathbf{w}_j)) \quad (11)$$

where the first term moves the waypoint $\mathbf{w}_j$ closer to its immediate neighbors $\mathbf{w}_{j-1}$ and $\mathbf{w}_{j+1}$, and the second term moves the waypoint away from the obstacle at $\zeta\mathbf{w}_j$.

5 Experimental Results

We compared our integrated value function/policy search method to an existing approximate value function planner [6] which uses a local collision avoidance module to refine the approximate value function to a realizable trajectory. The expected trajectory of a sample problem is shown in figure 3 in its 4 phases: the dynamic programming path, the projection of the DP solution to the 5-dimension continuous space, the initial phase of search, and the final optimized path. The image depicts a map of an indoor environment (our lab), and part of the corridor outside the lab. The robot starts inside the lab, and attempts to drive out to the corridor. The black areas are obstacles (i.e., walls, etc.) and the white areas are free space. The opening at the top of the map is the doorway to the corridor, and is 60 cm wide, whereas the robot's width and safety margin is 54cm, leaving only 2cm clearance on either side.

Figures 3a and 3b demonstrate that although the value function does show a trajectory that is in some sense topologically correct, the policy is clearly suboptimal with respect to the real world. The reduced state space means that the model has no notion of the size of the robot or of the consequences of different orientations. As a result, the policy is free to move the robot arbitrarily close to obstacles. Figures 3c and 3d show the results of policy iteration, after searching on the initial set of waypoints and after the waypoint set size has converged. The policy found with the initial set of waypoints is a clear improvement, however some odd motions are generated to compensate for an impoverished policy representation. When more waypoints are added, the policy converges to a straight-line path through the narrow doorway. The effect of the additional dimensions of velocity can be seen in the middle of the doorway – the robot's velocity is low entering the doorway, to reduce the likelihood of hitting the door. Once the robot is part-way through the door, the additional waypoints do not change the robot's heading, but allow velocity increase since the kinematics of the robot dictate that the danger of impact has passed.

Table 2 shows a quantitative comparison of an existing planner based on an approximate value function algorithm and a planner with value function seeding policy search. The results shown here indicate that although the planning time increases substantially when adding policy search, the total time taken to execute the trajectory drops significantly compared to the value function approach. The trajectory is also much more acceptable in an "intuitive" sense – the controller from the previous planner spent a significant amount of time driving back and forth across the narrow doorway before it was aligned well enough to enter the lab. It is worth noting that it should be relatively easy to improve the planning time of the policy search even further as the current implementation has not been optimized in any manner.

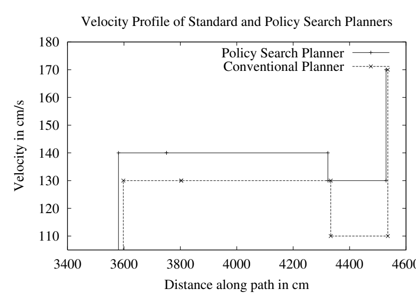


Figure 4: *Velocity profiles of the policy search and conventional planner.*

Figure 4 also shows the velocity profiles of the policy search planner and the conventional planner, for a different trajectory (not shown due to space limitations). The velocity profiles shows that the policy search finds a plan that achieves a consistently higher velocity (between 10-

Table 2: Experimental results

Algorithm	Planning time	Execution time	Distance travelled
Prior (value function) planner	.44 +/- .03 sec	133.15	3272.5 cm
Policy search	7.0 +/- 4.8 sec	107.6	1432.1 cm

20 cm/s faster).

6 Conclusions

In this paper, we have shown how seeding of the policy search with a solution from an approximate value function can be used to directly perform motion planning in the full state space of the robot. This allows us to avoid many of the assumptions of existing motion planners, such as point-shaped robots and deterministic motion. We are also able to generate plans that obey the physical, kinematic and dynamic constraints of the robot, and can express a richer set of plans. The key assumption is that the low-dimensional value function, while suboptimal with respect to reality, biases the policy search correctly, and does not lead to an arbitrarily bad controller.

We demonstrated this work on a real robot navigating in our lab, in particular focusing on a goal that requires a highly constrained trajectory. Preliminary results indicate we are able to handle these real world problems appropriately, and are able to outperform an existing approximate value function style planner in constrained situations.

Although we currently assume a fully observable Markov decision process model, in the future we would like to draw upon much of the recent work on policy search in POMDPs to better model noisy robot sensors. Certainly, approximate value function methods such as Cassandra et al. [4] and Roy & Thrun [16] could be directly applied to this method.

Acknowledgments

The authors would like to thank Drew Bagnell for useful discussions of this work, as well as improvements he made to the execution control loop. Joelle Pineau and Jonathan Baxter provided some valuable suggestions on earlier versions of this paper. Michael Montemerlo's assistance was invaluable in developing software for controlling the robot hardware.

References

[1] J. Andrew Bagnell and Jeff Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proc. International Conference on Robotics and Automation*, 2001.

[2] Johann Borenstein and Ulrich Raschke. Real-time obstacle avoidance for non-point mobile robots. In *Proceedings of the Fourth World Conference on Robotics Research*, pages 2.1–2.9, Pittsburgh, PA, Sept 1991.

[3] 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(1), 1995.

[4] Anthony R. Cassandra, Leslie Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proc. of the IEEE/RSJ Interational Conference on Intelligent Robotic Systems (IROS)*, 1996.

[5] Scott Davies, Andrew Ng, and Andrew Moore. Applying online search techniques to continuous-state reinforcement learning. In *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[6] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 1997.

[7] jean Paul Laumond, M. Taix M., and P. Jacobs. A motion planner for car-like robots based on a global/local approach. In *In Proc. IEEE Internat. Workshop Intell. Robot Syst*, page pages 765773, 1990.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), Spring 1986.

[10] Nak Yong Ko and Reid Simmons. The lane-curvature method for local obstacle avoidance. In *Proc. of Conference on Intelligent Robotics and Systems*, Vancouver, Canada, OPTmonth 1998.

[11] Daniel Koditschek. Control of natural motion in mechanical systems. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 113(4):547–551, 1991.

[12] Kurt Konolige. A gradient method for realtime robot control. In *Proc. of the IEEE/RSJ Interational Conference on Intelligent Robotic Systems (IROS)*, 2000.

[13] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[14] Andrew Ng, Ronald Parr, and Daphne Koller. Policy search via density estimation. In *Advances in Neural International Proc. Systems 12*, volume 12, 2000.

[15] Dan Pelleg and Andrew Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proc. ICML*, 2000.

[16] Nicholas Roy and Sebastian Thrun. Coastal navigation with mobile robots. In *Advances in Neural Processing Systems 11*, volume 11, pages 1043–1049, 1999.

[17] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31:29–53, 1998.

[18] Iwan Ulrich and Johann Borenstein. Vfh+: Reliable obstacle avoidance for fast mobile robots. In *Proc. of the 1998 IEEE International Conference on Robotics and Automation*, pages 1572–1577, Leuven, Belgium, May 1998.