

# Extracting Optimal Paths from Roadmaps for Motion Planning

Jinsuck Kim   Roger A. Pearce   Nancy M. Amato  
Department of Computer Science  
Texas A&M University  
College Station, TX 77843  
{jinsuckk, rap2317, amato}@cs.tamu.edu

## Abstract

*We present methods for extracting optimal paths from motion planning roadmaps. Our system enables any combination of optimization criteria, such as collision detection, kinematic/dynamic constraints, or minimum clearance, and relaxed definitions of the goal state, to be used when selecting paths from roadmaps. Our algorithm is an augmented version of Dijkstra’s shortest path algorithm which allows edge weights to be defined relative to the current path. We present simulation results maximizing minimum path clearance, minimizing localization effort, and enforcing kinematic/dynamic constraints.*

## I. INTRODUCTION

Automatic motion planning has been used in many areas such as robotics and computer-aided design (CAD) to find paths in the presence of obstacles. Though originating in robotics, motion planning techniques have been adapted to areas such as autonomous transportation systems for automobiles or aircraft, military unmanned vehicles that operate in the air or underwater, and computer animations in the entertainment industry.

In these applications, paths must be found quickly in large search spaces. Roadmap-based planners are ideal for such scenarios [4]. A roadmap containing representative paths is computed during a preprocessing step, and paths can be quickly extracted from the roadmap during query processing. In particular, the roadmap is a graph representing the connectivity of the free configuration space, where nodes are robot configurations and edges are paths computed by a simple and deterministic local planner.

The strength of roadmap-based planners is that the roadmap is a compact approximation of the connectivity of the planning space. While roadmap-based planners are extremely effective in providing feasible solution paths for arbitrary queries, generally no guarantees can be provided regarding the quality of the paths. In particular, paths extracted from roadmaps seldom provide optimal solutions because they are restricted to the nodes and edges in the roadmaps. In many cases, this is not a concern because the problem of interest is simply finding a feasible path. For this reason, optimizing paths has received little attention for roadmap-based planners.

In this paper, we consider the problem of extracting an optimal path from among all paths contained in the

roadmap. There are two main issues that are of concern. First, roadmaps contain many possible routes connecting two different nodes. Depending on the graph search algorithm and the criteria applied, different paths connecting the same start and goal nodes can be found. Second, a path extracted from a roadmap is composed of many short line segments and its quality is likely lower than a “smoothed” path obtained by exhaustive numerical optimization. These two properties are inherent in roadmap-based methods. We call the first a *macroscopic property* because the chosen search method can result in large-scale changes in the path. We refer to the second as a *microscopic property* because typically there are no topological differences between the extracted path and the optimal path.

A number of techniques have been proposed to improve the solution paths extracted from roadmaps (the microscopic property). Common approaches are to post-process the path by converting the path to a curve [12], moving existing nodes, or adding additional nodes to the suboptimal path [10].

In this paper, we focus on the macroscopic property and provide a method to quickly compute an optimal path from among all paths contained in the roadmap. Our method is based on an augmented version of Dijkstra’s shortest path algorithm which enables one to consider more general optimization criteria and relaxed definitions of the goal state.

## II. RELATED WORK

Previous research shows that applying common optimization techniques to robotics motion planning is not straightforward because the collision-free requirement renders it difficult to solve analytically or numerically [2], [11]. Figure 1(a) shows a path extracted from a roadmap ( $p_2$ ) and paths numerically generated by general optimization techniques ( $p_1, p_3, p_4$ ). Figure 1(b) shows two regions separated by an obstacle. To solve two-point boundary-value optimization problems, an initial guess of the solution must be given [7]. If the initial guess is  $p_4$ , then the solution cannot be improved beyond  $p_3$  without understanding the discontinuity of the search space. However, the suboptimal path  $p_2$  can be transformed to the optimal  $p_1$ . This explains the importance of the search of an optimal path in the macroscopic way.

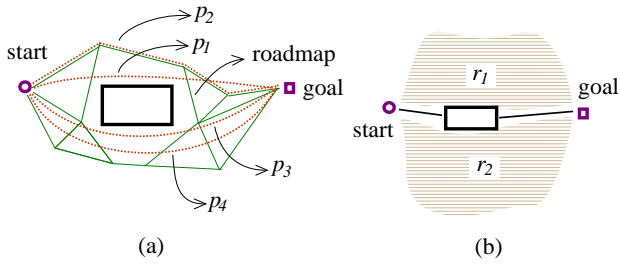


Fig. 1. Optimizing path with initial guesses

Recently, two different approaches have been developed to obtain optimal paths in robotics applications.

**Improving robot paths.** Many recent methods for motion planning are explicitly/implicitly based on roadmaps. Several methods consider the problem of optimizing or improving an existing path, for example, grids [9], visibility graphs [1], [12], and growing control points in barycentric coordinates [8].

All of the approaches above use deterministic roadmaps. Probabilistic roadmaps encoding physical constraints have been studied in [10] where the roadmap is customized for various applications, and paths are improved in the query step.

**Finding optimal paths.** Optimal paths can be obtained by modifying general optimization or optimal control techniques for motion planning. Because the methods are not based on roadmaps, collision checking needs to be geometrically and/or mathematically formulated, and is relatively complex and inefficient.

In [11], the constraints of the optimization problem are extended to AND and OR logic, which are referred to as generalized constraints and deal with polygonal obstacles. Modification of genetic algorithms was attempted in [3] to improve the path using Gram–Schmidt orthogonalization.

**Dijkstra’s algorithm.** Our optimization method is based on Dijkstra’s shortest path algorithm. Dijkstra’s algorithm searches for a shortest path in a weighted directed graph  $(V, E)$  where all edge weights are non-negative. Dijkstra’s algorithm is widely used in many areas where the path cost needs to be minimized, for example in wireless network applications [13] where the edge cost is an estimation of the required transmission power and the propagation delay.

### III. ISSUES IN PATH OPTIMIZATION

In this section, we discuss useful properties and requirements for computing optimal paths in robotics that have not been addressed in previous work.

**Standard cost function.** The optimization of certain values for a physical system that moves from an initial

state at time 0 to a final state at time  $T_f$ , while subject to constraints, is described by the problem of minimizing a cost function. The standard cost function in optimal control theory [7] is described by

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) \quad (1)$$

where  $x(t)$  is the state at time  $t$  and  $u(t)$  is the control input at time  $t$ . The necessary condition at the final time  $T_f$  is described by  $h(x(T_f))$ . This is Markov in the sense that  $x(t+1)$  is determined by the value of  $J$  which is evaluated for time  $t$  only.

**Non-Markov optimization criteria.** Compared to our work, previous work with roadmap-based methods lacks two important properties needed for real applications. The first is the need for non-Markovian states, i.e., states which depend on information from a range of previous states.

For example, to maximize clearance, it is clear that a cost function  $g$  will contain the reciprocal of clearance if the optimizer minimizes  $J$ . We denote the reciprocal of the clearance as  $\frac{1}{cl(x(t))}$ . If we let  $g(x(t), u(t)) = \frac{1}{cl(x(t))}$  in Equation 1, then the resulting path will maximize the accumulated clearance from the start to goal. In most cases, the objective is to optimize the path for maximum safety and the proper criterion is maximizing the minimum path clearance, not maximizing the accumulated clearance. This requires a modified cost function

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) + \frac{1}{cl(x(t_m))} \quad (2)$$

where  $t_m \in [0, T_f]$  such that  $cl(x(t_m))$  is minimum.

**Goal sets – flexible final states.** The second issue that has not been addressed in previous work is a flexible definition of the final necessary condition. If the final condition at  $T_f$  is equivalently described using an equality condition  $h(x(T_f)) = 0$ , then the  $h(x(T_f))$  term is removed from Equation 1.

Unfortunately, in a graph-search based path planner such as Dijkstra’s algorithm, it is difficult to find a node that satisfies  $h(x(T_f)) = 0$  unless some of the nodes are generated exactly on the surface with the condition satisfied. So, we expand the surface using an inequality condition.

$$J = \int_0^{T_f} g(x(t), u(t))dt + \frac{1}{cl(x(t_m))} \quad (3)$$

$$h(x(T_f)) \leq c_f$$

The inequality condition is used to terminate the graph search if any node satisfying  $h(x(T_f)) \leq c_f$  is reached. We call this set of nodes a *goal set*.

Given environment,  $start$  and  $c_f$ ,  
find a path  $p = \{e_1, e_2, \dots, e_{n_f}\}$  such that  
minimize  $cost(p)$   
where  $cost(p) = \sum cost_g(e_i)$  subject to  
 $start(e_1) = start$   
 $clearance(e_i) > 0, \quad i = 1 \dots n_f$   
others (e.g., time, energy, ...)  
and by the final condition  
 $end(e_{n_f}) \in goal_{set}$   
where  $goal_{set} = \{end(e_i) | cost_h(end(e_i)) \leq c_f\}$

Fig. 2. Path optimization problem

#### IV. SYSTEM DESCRIPTION

Our path optimization system is based on the roadmap method and Dijkstra's shortest path algorithm. To address the issues mentioned in the previous section, we designed an augmented version of Dijkstra's algorithm and cost computation.

##### A. Problem Formulation

Before explaining the details of our framework, we reformat the mathematical description (in Equation 3) to a pseudo-code friendly version. Figure 2 describes our path optimization problem of minimizing the cost of a given path  $p$ . Operators  $start(e_i)$  and  $end(e_i)$  denote the start and end vertex of edge  $e_i$ , respectively, and the cost functions  $cost_g$  and  $cost_h$  denote the functions  $g$  and  $h$  in Equation 3, respectively.  $Start$  is a node in the roadmap, and the final condition specified by a constant  $c_f$  is internally transformed to a goal set,  $goal_{set}$ , that will terminate the search when reached. In Section IV-D, pseudo code is used to describe this in detail.

##### B. Markov-like Optimization

**Ideal Markov Function.** The issue of maximizing minimum clearance was introduced in Section III, and the cost function including a non-Markovian state is

$$J = \int_0^{T_f} g(x(t), u(t)) dt + m(x(t_m)) \quad (4)$$

where  $m(x(t_m))$  is a general non-Markovian cost function and  $t_m \in [0, T_f]$ . In Equation 2,  $m(x(t_m))$  was  $\frac{1}{cl(x(t_m))}$ .

**Discretization.** Since we are using a graph search algorithm which is similar to dynamic programming in classic optimization theory, Equation 4 can be represented by a discretized version

$$J = \sum_{i=0}^{i \leq N_f} g(x_i, u_i) + m(x_{i_m}), \quad (5)$$

$$x_i = a(x_{i-1}, u_{i-1}), \quad i_m \in \{0, 1, \dots, N_f\}$$

where  $N_f$  is the total number of time steps,  $i_m$  is the time step corresponding to  $t_m$ , and  $a$  is a discrete time state update equation of the system dynamics.

**Markov-like Cost Function.** Now, we replace  $g(x_i, u_i)$  with  $g(x_i, x_{i-1}, u_i)$  so that both previous and current states are used for computing the cost. The previous state is obtained using the *parent* data structure in the search tree of Dijkstra's algorithm (see Figure 3). We call this approach Markov-like because using  $x_{i-1}$  is not Markov in a strict sense but  $x_i$  and  $x_{i-1}$  can be denoted by a compound state  $\mathbf{x}_i$ . The general cost function is

$$J = \sum_{i=1}^{i \leq N_f} \mathbf{g}(\mathbf{x}_i, u_i) + m(x_{i_m}), \quad \mathbf{x}_i = \begin{pmatrix} x_i \\ x_{i-1} \end{pmatrix} \quad (6)$$

**New State Update Equation.** We added  $x_{i-1}$  to the cost function with the intention of eliminating  $m(x_{i_m})$ , and the state equation  $a(x_{i-1}, u_{i-1})$  needs to be changed accordingly. The idea is that  $\mathbf{x}_i$  should contain the entire history of the non-Markovian property. For example, to maximize the minimum path clearance, an element in  $\mathbf{x}_i$  will indicate the minimum clearance from start to time step  $i$ . Now, we denote the minimum clearance state by  $x_i^{cl}$  and add it to  $\mathbf{x}_i$ .

$$J = \sum_{i=1}^{i \leq N_f} \mathbf{g}(\mathbf{x}_i, u_i), \quad \mathbf{x}_i = [x_i \ x_i^{cl} \ x_{i-1}^{cl}]^T$$

$$\begin{pmatrix} x_i \\ x_i^{cl} \end{pmatrix} = \begin{pmatrix} a(x_{i-1}, u_{i-1}) \\ a^{cl}(x_i, x_{i-1}^{cl}) \end{pmatrix} \quad (7)$$

The state equation  $a^{cl}$  returns  $x_i^{cl}$  that is lower than  $x_{i-1}^{cl}$  only if  $cl(x_i)$ , the clearance of  $x_i$ , is smaller than  $x_{i-1}^{cl}$ . Otherwise,  $x_i^{cl}$  must equal  $x_{i-1}^{cl}$  because the clearance of the current state is not smaller than the minimum clearance discovered so far (see Figure 7).

$$a^{cl}(x_i, x_{i-1}^{cl}) = \begin{cases} cl(x_i) & \text{if } cl(x_i) < x_{i-1}^{cl} \\ x_{i-1}^{cl} & \text{otherwise} \end{cases} \quad (8)$$

**New Cost Function.** Next, we focus on  $g^{cl}(x_i^{cl}, x_{i-1}^{cl})$  which is a part of  $\mathbf{g}(\mathbf{x}_i, u_i)$  and corresponds to the state  $x^{cl}$ . It compares the difference between  $x_i^{cl}$  and  $x_{i-1}^{cl}$ , and should return a nonzero positive value if  $x_i^{cl} < x_{i-1}^{cl}$ . Otherwise, it returns zero so that  $J$  does not increase. So, we have

$$g^{cl}(x_i^{cl}, x_{i-1}^{cl}) = \begin{cases} c \cdot (x_{i-1}^{cl} - x_i^{cl}) & \text{if } x_i^{cl} < x_{i-1}^{cl} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where  $c$  is a constant. This technique for minimum clearance can be applied to other non-Markovian optimization values with the superscript  $cl$  changed in Equations 7, 8 and 9.

### C. Flexible Final Condition

We apply the modified final condition shown in Equation 3 to our new cost function in Equation 7, which is the final form of the cost function that we seek.

$$J = \sum_{i=1}^{i \leq N_f} \mathbf{g}(\mathbf{x}_i, u_i), \quad \mathbf{x}_i = \mathbf{a}(\mathbf{x}_i, u_{i-1}) \quad (10)$$

$$\mathbf{h}(\mathbf{x}_{N_f}, u_{N_f}) \leq c_f$$

### D. Augmented Dijkstra's Algorithm

Dijkstra's algorithm is augmented to reflect the changes, and its pseudo code is shown in Figure 3. Since  $x_{i-1}$  was introduced in Equation 6,  $parent[u]$  is added in line 7 and 8. The cost function  $cost_h$  in line 10 checks if a node is in the goal set using  $c_f$ .

#### AUGMENTED DIJKSTRA( $V, E, start, c_f$ )

1. for (each  $v \in V$ )  $dist[v] \leftarrow \infty$
2.  $dist[start] \leftarrow 0$
3.  $PQ \leftarrow$  PriorityQueue of  $V$  ordered by  $dist$
4. while ( $PQ \neq \emptyset$ )
5.    $u \leftarrow PQ.dequeue$
6.   for each  $v \in PQ$  adjacent to  $u$
7.     if ( $dist[v] > (dist[u] + weight(u, v, parent[u]))$ )
8.        $dist[v] \leftarrow dist[u] + weight(u, v, parent[u])$
9.        $parent[v] \leftarrow u$
10.     if ( $cost_h[v] < c_f$ ) return
11.  $PQ.reorder$

Fig. 3. The augmented Dijkstra's algorithm

## V. MOBILE ROBOT APPLICATIONS

In this section we provide some robotic examples that benefit from the path optimization methods described. They utilize our roadmap-based mobile robot system described in [5], [6]. It uses feature based localization and sonar range sensors. A T-shaped environment and roadmap are shown in Figure 4 where five nodes in the goal set are marked.

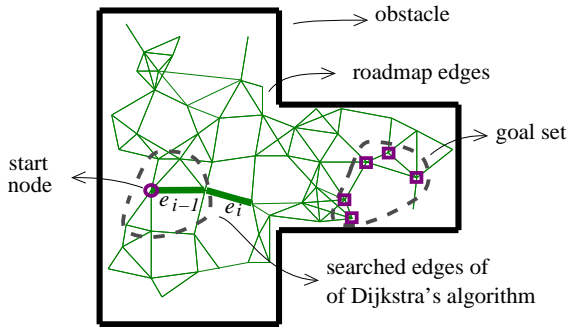


Fig. 4. Environment, roadmap and path searching.

**Avoiding Localization Failure.** In this case, we assume that the robot's sensors have range limits and always fail to localize if no feature exists within the range. The

locations of all features in the environment are assumed to be known. In Figure 6, we use

$$cost(e_i) = c_3 \cdot f_1(\text{visibility of } e_i) \quad (11)$$

where 'visibility of  $e_i$ ' is the expected number of features to be scanned by the robot when it is on edge  $e_i$ . The function  $f_1$  converts the visibility of edge  $e_i$  into a scalar as shown in Figure 5(a). Note that the optimal path can traverse a region with no features if necessary.

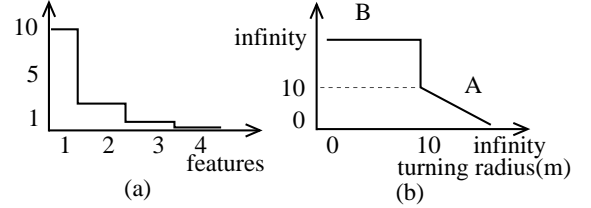


Fig. 5. Cost functions, (a) for features and (b) for turning radius.

**Kinematic Constraints.** If the robot has constraints on its turning radius, two adjacent edges  $e_i$  and  $e_{i-1}$  are needed to compute the required turning radius to obtain the cost of  $e_i$ . The weight function now uses two edges (or three vertices) as shown in the pseudo code in Figure 3. In Figure 6, which reflects the modified weight computation, we use

$$cost(e_i) = c_4 \cdot f_2(\text{turn radius of } e_{i-1} \text{ and } e_i) \quad (12)$$

where  $f_2$  is an appropriate linear or nonlinear function.

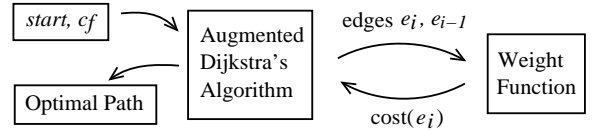


Fig. 6. Weight function with two adjacent edges.

Figure 5(b) shows an example of a nonlinear function that maximizes turning radius (region A) and prohibits  $e_i$  from being used if it violates the kinematic constraint of a turning radius of less than 10 meters (region B).

**Maximizing Minimum Clearance.** As discussed in Section IV-B, the minimum clearance  $x^{cl}$  is a non-increasing variable and is shown as a solid line in Figure 7. To implement this in the augmented Dijkstra's algorithm framework, we add the new variable as auxiliary data as in Figure 8. The data is maintained according to the rule shown in Equation 8. The edge cost computation equivalent to Equation 9 is described by

$$cost(e_i) = \begin{cases} c_5 \cdot (cl_{min} - cl(e_i)) & \text{if } cl(e_i) < cl_{min} \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

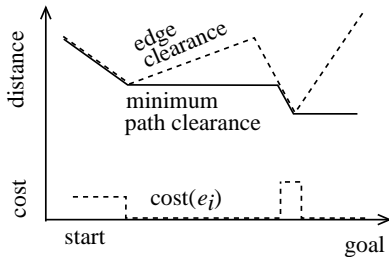


Fig. 7. Cost function for edge clearance

where  $cl_{min}$  is the auxiliary data and  $cl(e_i)$  is the clearance of edge  $e_i$ . Initially,  $cl_{min}$  is set to the clearance of the start node.

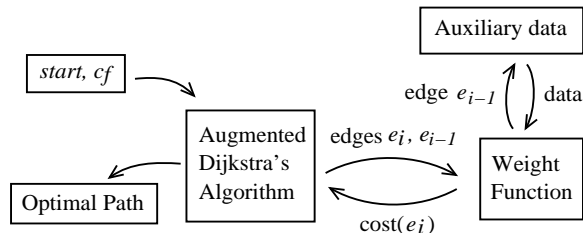


Fig. 8. Weighting with two adjacent edges and related data.

**Combination of Criteria.** Combining various costs into one function results in the simultaneous optimization for multiple values, and is useful in many applications. The combined cost of an edge is the weighted sum of individual costs.

$$cost(e_i) = \sum_j w_j \cdot cost_j \quad (14)$$

## VI. SIMULATION RESULTS

To generate roadmaps for simulations, we used a class of roadmap-based planning methods which are called probabilistic roadmap methods (PRMs) that have proven to be very successful in efficiently solving high-dimensional problems in complex environments [4].

Three different possible routes exist in the environment using the roadmap shown in Figure 9(a) from the start to goal area in Figure 9(c). Paths going through corridor A or C in Figure 9(c) are obtained by maximizing the minimum clearance or minimizing path length, respectively. Figure 9(c) shows the path going through corridor B; this results from the combination of minimizing path length and maximizing minimum path clearance.

$$cost(e_i) = 0.03 \text{ length}(e_i) + \begin{cases} 0.97 (cl_{min} - cl(e_i)) & \text{if } cl(e_i) < cl_{min} \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

Search tree edges of Dijkstra's algorithm are illustrated in Figure 9(b) by arrows representing the direction of the search from the start node.

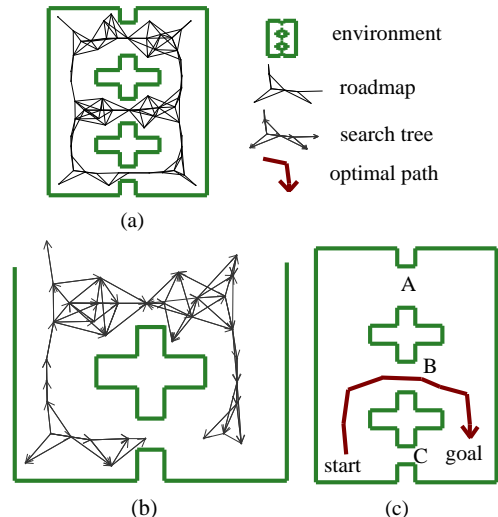


Fig. 9. Maximizing clearance and combination of criteria.

Several simulations in the same environment are presented in Table I using another parameter, turning radius. Then, the  $cost(e_i)$  is computed using three constant weights  $cost_{dist}$ ,  $cost_{cl}$  and  $cost_{tr}$ .  $Cost_{tr}$  is the cost for turning radius and penalizes the edge with a sharp turn. The second row shows that the smoothest path is obtained by going through region B, which is shown in Figure 9(c). The second and third rows, and Equation 15 show that different combinations of weight constants can result in similar paths.

Route	$cost_{dist}$	$cost_{cl}$	$cost_{tr}$
A	0	1	0
B	0	0	1
B	0.03	0.32	0.65
C	1	0	0

TABLE I

SIMULATIONS WITH DIFFERENT PARAMETERS

## VII. CONCLUSIONS

A framework for extracting an optimal path in a motion planning roadmap has been proposed. Our framework combines the mathematical flexibility of general optimization techniques and computational efficiency of roadmap-based methods. We designed an augmented Dijkstra's shortest path algorithm that uses Markov-like states and goal sets. Using PRMs, the path can be efficiently optimized in a large space for several values including kinematic/dynamic constraints and minimum clearance. Simulation results were presented to illustrate the feasibility of our approach.

Future work consists of experimenting with robots with high degrees of freedom, efficient algorithms for path smoothing, and hardware experiments using mobile robots.

#### VIII. REFERENCES

- [1] C. Ahrikencheikh and A. Seireg. *Optimized-Motion Planning*. John Wiley & Sons, Inc., 1994.
- [2] S. Akella and S. Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 624–631, 2002.
- [3] C. Hocaoglu and A. Sanderson. Planning multiple paths with evolutionary speciation. In *IEEE transactions on evolutionary computation*, volume 5, pages 169–191, 2001.
- [4] L. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE Trans. Robot. Automat.*, volume 14, pages 166–171, 1998.
- [5] J. Kim, N.M. Amato, and S. Lee. An integrated mobile robot path (re)planner and localizer for personal robots. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3789–3794, 2001.
- [6] J. Kim, R.A. Pearce, and N.M. Amato. Robust geometric-based localization in indoor environments using sonar range sensors. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2002. To appear.
- [7] Donald E. Kirk. *Optimal Control Theory: An Introduction*. Prentice Hall, 1970.
- [8] P. Konkimalla and S.M. LaValle. Efficient computation of optimal navigation functions for nonholonomic planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 187–192, 1999.
- [9] Z. Shiller, K. Yamane, and Y. Nakamura. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1–8, 2001.
- [10] G. Song, S. L. Miller, and N. M. Amato. Customizing PRM roadmaps at query time. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1500–1505, 2001.
- [11] Y. Wang and D. Lane. Solving a generalized constrained optimization problem with both logic AND and OR relationships by mathematical transformation and its application to robot motion planning. In *IEEE transactions on systems, man and cybernetics, part C*, pages 525–536, 2000.
- [12] M. Yamamoto, M. Iwamura, and A. Mohri. Quasi-time-optimal motion planning of mobile platforms in the presence of obstacles. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2958–2963, 1999.
- [13] M. A. Youssef, M. F. Younis, and K. A. Arisha. A constrained shortest-path energy-aware routing algorithm for wireless sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 794–799, 2002.