OPTIMAL MOTION PLANNING WITH CONSTRAINTS FOR MOBILE ROBOT NAVIGATION

A Senior Honors Thesis

by

ROGER ALLAN PEARCE

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

April 2003

Group: Engineering & Physics 2

OPTIMAL MOTION PLANNING WITH CONSTRAINTS FOR MOBILE ROBOT NAVIGATION

A Senior Honors Thesis

by

ROGER ALLAN PEARCE

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment for the designation of

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOW

Approved as to style and content by:

Nancy M. Amato
(Fellows Advisor)

Edward A. Funkhouser
(Executive Director)

April 2003

Group: Engineering & Physics 2

ABSTRACT

Optimal Motion Planning with Constraints for

Mobile Robot Navigation. (April 2003)

Roger Allan Pearce
Department of Computer Science
Texas A&M University

Fellows Advisor: Dr. Nancy M. Amato
Department of Computer Science

Motion planning is the process of planning a sequence of motions to move an object from one configuration to another. Recently, randomized techniques known as *PRMs* have shown great potential for solving motion planning problems in complicated high-dimensional space. Motion Planning, or path planning for robots, becomes increasing difficult as the dimension of the planning space increases with the robot's degrees of freedom (dof). While the running time of deterministic motion planning algorithms grows exponentially with an increase in dof, *PRMs* can produce solutions in times that do not depend on the dof but only the difficulty of the problem. *PRMs* randomly generate collision free configurations in a robot's Configuration-space (C-space), representing feasible positions and orientations for the robot. Nearby configurations are linked together by so called local planners, and these connections are edges in a roadmap, a graph containing representative discrete paths the robot may travel.

We present methods to extract optimal paths from roadmap–based motion planners. Our system uses Markov–like states and flexible goal states so that general optimization criteria including collision detection, kinematic/dynamic constraints, or minimum clearance can be used in various applications. Our algorithm is an aug-

mented version of Dijkstra's shortest path algorithm. We present simulation results maximizing minimum path clearance, minimizing localization effort, and enforcing kinematic/dynamic constraints.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Nancy M. Amato for giving me the opportunity, tools, and environment to work on this project. Through her dedication toward her research and students, she has provided me a great environment conducive to learning. Dr. Amato has become an invaluable academic mentor in my life.

Next, I would like to thank Jinsuck Kim, a PhD student under Dr. Amato. As my graduate student mentor, Jinsuck has been a great help through our research projects.

Finally, I would like to thank all the members of the Motion Planning group in the Parasol Lab: Burchan Bayazit, Nick Downing, Aimee Estrada, Jinsuck Kim, Jyh-Ming Lien, Marco Morales, Bharatinder Sandhu, Guang Song, Kasthuri Srinivasan, Rick Stover, Xinyu Tang, Shawna Thomas, and Dawen Xie.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS vi

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

I INTRODUCTION 1

II PRELIMINARY AND RELATED WORK 4

A. Roadmap based Motion Planning 4
B. Path Optimization 4
1. Improving Paths 6
2. Finding Optimal Paths 7
C. Dijkstra's Algorithm 7

III ISSUES IN ROBOT PATH OPTIMIZATION 9

A. Standard Cost Function 9
B. Non–Markov optimization criteria 10
C. Goal sets – flexible final states 10

IV SYSTEM DESCRIPTION 12

A. Problem Formulation 12
B. Markov–like Optimization 12
C. Flexible Final Condition 16
D. Augmented Dijkstra's Algorithm 16

V SYSTEM EVALUATION 18

A. Optimization Criteria 19

CHAPTER Page

1. Minimizing Travel Time 19
2. Avoiding Localization Failure 20
3. Kinematic Constraints 20
4. Maximizing Minimum Clearance 21

B. Combination of Criteria 22

C. Simulation Results 22

1. Maximizing Minimum Clearance 22
2. Flexible Final Condition and Dynamic Constraints . . 24

VI APPLICATION: CAMPUS NAVIGATOR 27

A. Components of the Campus Navigator 27

1. Overview of Campus Navigator 28
2. The Campus Graph 29
3. The Roadmap Editor 29
4. Campus Graph Query 30
5. Campus Path Visualization Via Vizmo++ 31
6. User Web Interface 31

VII SUMMARY AND CONCLUSIONS 33

A. Future Work 33

REFERENCES 34

VITA 37

LIST OF FIGURES

FIGURE		Page
1	Overview of roadmap–based motion planning.	5
2	Optimizing path with initial guesses	6
3	Pseudo code for Dijkstra's algorithm	8
4	Pseudo code for shortest path algorithm	8
5	Path optimization problem	13
6	The augmented Dijkstra's algorithm	16
7	Environment, roadmap and path searching.	18
8	Diagram of shortest path computation.	19
9	Cost functions, (a) for features and (b) for turning radius.	20
10	Weight function with two adjacent edges.	21
11	Cost function for edge clearance	22
12	Weighting with two adjacent edges and related data.	23
13	Maximizing clearance and combination of criteria	24
14	Simulations with different parameters	25
15	Mobile platform with robotic arm. (a) Roadmap, (b) start position, and (c) three different configurations in the goal set.	26
16	Overview of Campus Navigator system.	28
17	Prototype of Campus Navigator interface	31

CHAPTER 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 also been adapted to other 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 [1]. A roadmap containing representative paths is computed during a preprocessing step, and paths can be quickly extracted from the roadmap during query processing. Recently, a class of roadmap–based planning methods, called probabilistic roadmap methods (*PRMs*) [1], have proven to be very successful in efficiently solving high–dimensional problems in complex environments. In *PRMs*, the roadmap is a graph representing the connectivity of the free configuration space where the nodes are sampled robot configurations and the edges are paths connecting nodes that are computed by a simple and deterministic local planner.

The strength of roadmap–based planners is that the roadmap approximates 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

---

The journal model is *IEEE Transactions on Automatic Control.*

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 research project, 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, moving existing nodes, or adding additional nodes to the suboptimal path [2, 3].

In this research project, 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.

The results in this thesis will appear in *ICRA 2003*, the IEEE International Conference on Robotics and Automation [4]. This work builds on our previous work

on mobile robot navigation and localization [5, 6].

CHAPTER II

PRELIMINARY AND RELATED WORK

A. Roadmap-based Motion Planning

At the heart of our path optimization techniques is roadmap–based motion planning. These motion planning methods model the robot(s) in Configuration Space (*C-space*), a multi-dimensional space where the dimensions represent the *dof* of the robot. This configuration consists of all the information required to describe the robot's position and orientation in the real world. For example, a cube moving in 3d space will need a six dimensional configuration $< x, y, z, roll, pitch, yaw >$, and a Mobile Robot or car will need a three dimensional configuration $< x, y, orientation >$.

Many roadmap–based motion planners focus on randomly creating roadmaps to solve complicated high-dimensional problems. These roadmaps consist of randomly created vertices representing a single configuration of the robot. These vertices are connected together by edges representing collision free paths between two vertices. Many random and heuristic methods have been developed to create roadmap edges and the end result is a graph consisting of vertices and edges representing a small discrete set of collision-free space in the robot's environment. Figure 1 describes how many roadmap–based motion planning methods work.

B. Path Optimization

Previous research shows that applying common optimization techniques to robotics is not straightforward because the collision–free requirement renders it difficult to solve optimization constraints analytically or numerically [7, 8]. In particular, discontinuity of the search space makes it difficult to find the optimal path. Figure 2(a) shows a

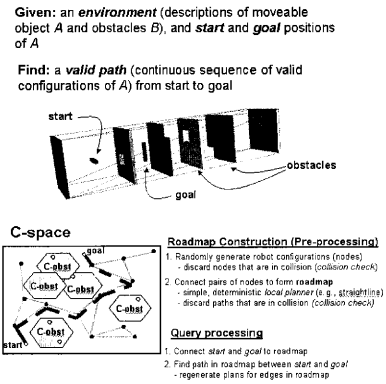

Fig. 1. Overview of roadmap–based motion planning.

path extracted from a roadmap (p_2) and paths generated by general optimization techniques (p_1, p_3, p_4). Figure 2(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 [9]. 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.

Recently, two different approaches have been developed to obtain optimal paths in robotics applications; one is based on improving existing paths, the other applies optimal control techniques to motion planning.

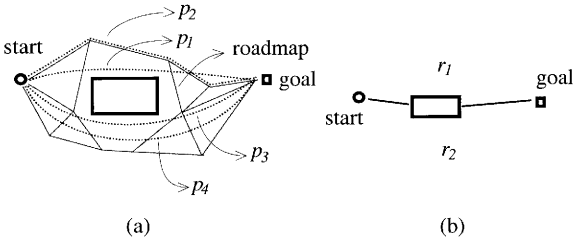

Fig. 2. Optimizing path with initial guesses

1. Improving Paths

Most 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 [10], visibility graphs [11, 3], *PRM*s [12], and growing control points in barycentric coordinates [13].

In [13], the optimal path of a nonholonomic robot is found by iteratively growing the computed region of optimal control points from the goal configuration using a cost–to–go function. To find optimal motions for human figures, [10] uses Dijkstra's shortest path algorithm in grids with edge weights reflecting the clearance and rotation of the body parts. For 2D environments with polygonal obstacles, [3] computes a roadmap from the visibility graph and optimizes a B–spline based path for kinematic constraints and driving torque.

All of the approaches above use deterministic roadmaps. Probabilistic roadmaps encoding physical constraints have been studied in [12] where the roadmap is customized for various applications, and paths are improved by iterative refinement in

the query step.

2. 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 [8], 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 [14] to improve the path using using Gram–Schmidt orthogonalization. To optimally coordinate multiple robots with specified trajectories, [7] used MILP (mixed integer linear programming) where the collision between two robots is formulated by a δ function.

It is difficult to apply these techniques to paths extracted from roadmaps due to the discontinuities in the search space (see Figure 2).

C. 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 nonnegative. Figure 3 shows the pseudo code for Dijkstra's algorithm where $dist[v]$ stores the shortest distance from *start* to v and PQ contains the unexplored vertices sorted by *dist*. The shortest path from *start* to *goal* is computed in the pseudo code in Figure 4 where Dijkstra's algorithm is used as a subroutine. The key to computing the correct solution is the *relaxation* in lines 8–9 of Figure 3 which repeatedly decreases an upper bound on the weight of the vertices

in PQ when a new lower–weighted path is found.

```
DIJKSTRA(V, E, start, goal)
for each v ∈ V
    dist[v] ← ∞
dist[start] ← 0
PQ ← PriorityQueue of V ordered by dist
while (PQ ≠ ∅)
    u ← PQ.dequeue
    for each v ∈ PQ adjacent to u
        if (dist[v] > (dist[v] + weight(u, v))
            dist[v] ← dist[v] + weight(u, v)
            parent[v] ← u
    PQ.reorder
```

Fig. 3. Pseudo code for Dijkstra's algorithm

```
SHORTESTPATH(V, E, start, goal)
parent ← DIJKSTRA(V, E, start, goal)
path ← ∅,   u ← goal
while (suffix of path ≠ start)
    append u to path
    u ← parent[u]
reverse path
```

Fig. 4. Pseudo code for shortest path algorithm

Dijkstra's algorithm is widely used in many areas where the path cost needs to be minimized, for example in wireless network applications [15] where the edge cost is an estimation of the required transmission power and the propagation delay.

CHAPTER III

ISSUES IN ROBOT 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. These issues motivate our augmentation of Dijkstra's algorithm for computing optimal paths in roadmaps. We start with a general cost function which is commonly used in the optimization of physical systems, and then discuss its limitations for robotics applications.

A. 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 J in optimal control theory [9] is described by

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) \tag{3.1}$$

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 form of the cost function has been used in [13] and optimizes the path of a car like robot by subdividing configuration space and linearly interpolating. In general, an optimal path satisfying Equation 3.1 with initial and final boundary conditions can be computed using several numerical methods [9].

B. 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 3.1, then the resulting path will maximize the accumulated $\frac{1}{cl(x(t))}$ 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

$$J = \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) + \frac{1}{cl(x(t_m))} \tag{3.2}$$

where $t_m \in [0, T_f]$ such that $cl(x(t_m))$ is minimum, and maximizing $cl(x(t_m))$ is equivalent to minimizing $\frac{1}{cl(x(t_m))}$. The term $cl(x(t_m))$ is non–Markov, and we force the state to be Markov.

C. 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. Describing the final condition at T_f using an equality condition changes Equation 3.1 to

$$\begin{aligned} J &= \textstyle\int_0^{T_f} g(x(t), u(t))dt \\ h(x(T_f)) &= 0 \end{aligned} \tag{3.3}$$

where the problem is now minimizing J with $h(x(T_f)) = 0$ satisfied. This is identical to one of the boundary conditions of the optimal control formulation where T_f is free and $x(T_f)$ is moving on the surface, $h(x(t)) = 0$. 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 where $h(x(t)) = 0$. So, we modify the surface to be more inclusive by using an inequality condition.

$$\begin{aligned} J &= \int_0^{T_f} g(x(t), u(t))dt \\ h(x(T_f)) &\leq c_f \end{aligned} \tag{3.4}$$

The final necessary condition $h(x(T_f)) \leq c_f$ 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*, and its size is determined by the constant c_f. Note that Dijkstra's algorithm requires two cost functions corresponding to g and h in Equation 3.4.

CHAPTER IV

SYSTEM DESCRIPTION

Our path optimization system is based on the probabilistic roadmap method (*PRMs*) and Dijkstra's shortest path algorithm. To address the issues mentioned in the previous section, we design 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 3.4) to a pseudo–code friendly version. Figure 5 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 3.4, 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 D, pseudo code is used to describe this in detail.

Note that we do not use the approach of iterative improvement of J, such as hill climbing and steepest descent. Like dynamic programming methods in optimal control, we compute the solution in one shot using Dijkstra's algorithm.

B. Markov–like Optimization

Ideal Markov Function. The issue of maximizing minimum clearance was intro-

Given environment, *start* and c_f,

find a path $p = \{e_1, e_2, \ldots\}$ such that

$$\text{minimize } cost(p)$$

where $cost(p) = \sum cost_g(e_i)$ under the constraints

$$start(e_1) = start$$

$$clearance(e_i) > 0, \qquad i = 1 \ldots n$$

others (e.g., time, energy, ...)

and $p = \{e_1, e_2, \ldots, e_f\}$ by the final condition

$$end(e_f) \in goal_{set}$$

where $goal_{set} = \{end(e_i)) \mid cost_h(end(e_i)) \leq c_f\}$

Fig. 5. Path optimization problem

duced in Section III, and the cost function including a non–Markovian state is

$$\begin{aligned} J = \; & \int_0^{T_f} g(x(t), u(t))dt + h(x(T_f)) + \\ & m(x(t_m)), \qquad\qquad t_m \in [0, T_f] \end{aligned} \tag{4.1}$$

where $m(x(t_m))$ is a general non–Markovian cost function. In Equation 3.2, $m(x(t_m))$ was $\frac{1}{cl(x(t_m))}$ with t_m the time when the clearance is lowest. This formulation is not tractable for common optimization solvers. Our approach to this problem is to modify $g(x, u)$ or $cost_g(e_i)$ in Figure 5 so that $m(x(t_m))$ is eliminated in the cost function.

Discretization. Since we are using a graph search algorithm which is similar to dynamic programming in classic optimization theory, Equation 4.1 can be represented

by a discretized version

$$J = \sum_{i=0}^{i \leq N_f} g(x_i, u_i) + h(x_{N_f}) + m(x_{i_m}),$$
$$i_m \in \{0, 1, \ldots, N_f\} \tag{4.2}$$
$$x_i = a(x_{i-1}, u_{i-1})$$

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.

Using Previous State. 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 by using *parent* data structure in the search tree of Dijkstra's algorithm. The vertex corresponding to x_{i-1} can be quickly obtained from the *parent* data structure and the vertex corresponding to x_i. We note that this is similar to converting a continuous time state $\dot{x}$ to a discrete time state composed of x_i, x_{i-1} and Δt using Taylor's series expansion. Many optimization values such as turning angle can be computed from $\dot{x}$ (or x_i, x_{i-1} and Δt if in discrete time). In this case, using $g(x_i, x_{i-1}, u_i)$ in Dijkstra's algorithm can be regarded as applying a standard discrete time optimization to a graph search technique. This does not exhaust the possible applications of our optimizer.

Markov–like Cost Function. An example of an optimization value that cannot be computed from $\dot{x}$ is minimum clearance, which will be computed from x_i and x_{i-1}. There are other optimization values such as localization success ratio that can be formulated using x_i and x_{i-1}. So, our motivation for using $g(x_i, x_{i-1}, u_i)$ is not from discretizing $g(x, \dot{x}, u)$, but to extend the ability of the graph search based path optimizer using current and previous states. We call this approach Markov–like because 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

$$
\begin{aligned}
J &= \textstyle\sum_{i=1}^{i \le N_f} \mathbf{g}(\mathbf{x}_i, u_i) + h(x_{N_f}) + m(x_{i_m}), \\
& \qquad\qquad\qquad\qquad i_m \in \{1, 2, \ldots, N_f\} \\
\mathbf{x}_i &= \begin{pmatrix} x_i \\ x_{i-1} \end{pmatrix}
\end{aligned}
\tag{4.3}
$$

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 $\mathbf{x}_i$.

$$
\begin{aligned}
& J = \textstyle\sum_{i=1}^{i \le N_f} \mathbf{g}(\mathbf{x}_i, u_i) + h(x_{N_f}), \\
& \begin{pmatrix} x_i \\ x_i^{cl} \end{pmatrix} = \begin{pmatrix} a(x_{i-1}, u_{i-1}) \\ a^{cl}(x_i, x_{i-1}^{cl}) \end{pmatrix} \\
& \mathbf{x}_i = [x_i \;\; x_i^{cl} \;\; x_{i-1}^{cl}]^T
\end{aligned}
\tag{4.4}
$$

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 11). It is clear that a^{cl} must contain a Boolean operator.

$$
a^{cl}(x_i,\, x_{i-1}^{cl}) = \begin{cases} cl(x_i) & \text{if } cl(x_i) < x_{i-1}^{cl} \\ x_{i-1}^{cl} & \text{otherwise} \end{cases}
\tag{4.5}
$$

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

$$g^{cl}(x_i^{cl},\ x_{i-1}^{cl}) = \begin{cases} c \cdot (x_{i-1}^{cl} - x_i^{cl}) & \text{if } x_i^{cl} < x_{i-1}^{cl} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

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 4.4, 4.5 and 4.6.

C. Flexible Final Condition

We apply the modified final condition shown in Equation 3.4 to our new cost function in Equation 4.4, which is the final form of the cost function that we seek.

$$\begin{aligned} J &= \textstyle\sum_{i=1}^{i \leq N_f} \mathbf{g}(\mathbf{x}_i, u_i), \\ \mathbf{x}_i &= \mathbf{a}(\mathbf{x}_i, u_{i-1}) \\ \mathbf{h}(\mathbf{x}_{N_f}, u_{N_f}) &\leq c_f \end{aligned} \tag{4.7}$$

D. Augmented Dijkstra's Algorithm

```
AUGMENTED DIJKSTRA(V, E, start, c_f)
for (each v ∈ V)  dist[v] ← ∞
dist[start] ← 0
PQ ← PriorityQueue of V ordered by dist
while (PQ ≠ ∅)
    u ← PQ.dequeue
    for each v ∈ PQ adjacent to u
        if (dist[v] > (dist[v] + weight(u, v, parent[u]))
            dist[v] ← dist[v] + weight(u, v, parent[u])
            parent[v] ← u
        if (cost_h[v] < c_f) return
    PQ.reorder
```

Fig. 6. The augmented Dijkstra's algorithm

Dijkstra's algorithm is augmented to reflect the changes in Equation 4.7, and its pseudo code is shown in Figure 6. To use Markov–like states, the *weight* function that corresponds to $cost_g$ in Figure 5 is changed so that three adjacent vertices are used. The cost function $cost_h$ checks if a node is in the goal set using c_f.

CHAPTER V

SYSTEM EVALUATION

In this chapter we provide some robotic examples that benefit from the path optimization methods described. The following example utilizes our roadmap-based mobile robot system described in [16, 6, 17]. It uses feature based localization and sonar range sensors. A T-shaped environment and roadmap are shown in Figure 7 where five nodes in the goal set are marked.

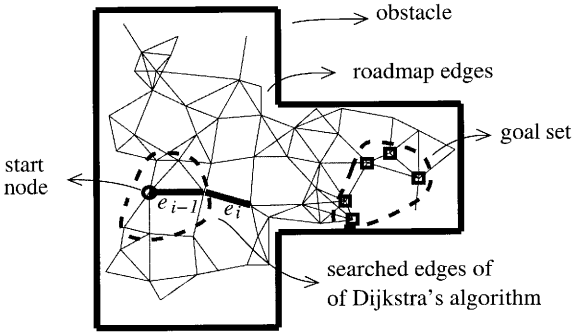

Fig. 7. Environment, roadmap and path searching.

In the following, we first describe various optimization criter and then we present some simulation results.

A. Optimization Criteria

The diagram shown in Figure 8 has two components, Dijkstra's algorithm and weight computation. In this section, we will show that various optimization values are computed by using different weight computations in the common framework. In the diagram shown in Figure 8, we compute the shortest distance path by using

$$cost(e_i) = length(e_i) \tag{5.1}$$

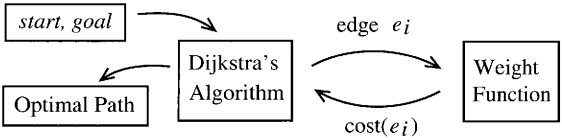

Fig. 8. Diagram of shortest path computation.

1. Minimizing Travel Time

The path extracted from a roadmap consists of a series of translations and rotations (unless converted to a curve). For ease of presentation, we assume that the rotation time can be approximated by a constant value and the translation time is proportional to the length of the edge. In Figure 8, travel time is minimized by using

$$cost(e_i) = c_1 \cdot length(e_i) + c_2 \tag{5.2}$$

where c_1 and c_2 are constants.

2. 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 8, we use

$$cost(e_i) = c_3 \cdot f_1(\text{visibility of } e_i) \tag{5.3}$$

where 'visibility of e_i' determines if the robot can successfully scan one or more feature(s) on the edge e_i. The function $f_1(e_i)$ converts the visibility of edge e_i into a scalar as shown in Figure 9(a). Note that the optimal path can traverse a region with no features if necessary.

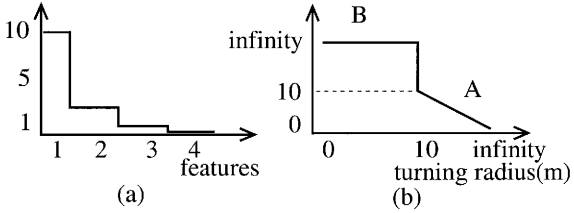

Fig. 9. Cost functions, (a) for features and (b) for turning radius.

3. 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

6. In Figure 10, which reflects the modified weight computation, we use

$$cost(e_i) = c_4 \cdot f_2(\text{turn radius of } e_{i-1} \text{ and } e_i) \tag{5.4}$$

where f_2 is an appropriate linear or nonlinear function.

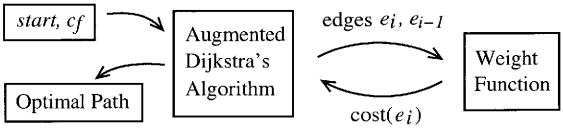

Fig. 10. Weight function with two adjacent edges.

Figure 9(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).

4. Maximizing Minimum Clearance

As discussed in Section B, the minimum clearance x^{cl} is a non–increasing variable and is shown as a solid line in Figure 11. To implement this in the augmented Dijkstra's algorithm framework, we add the new variable as auxiliary data as in Figure 12. The data is maintained according to the rule shown in Equation 4.5. The edge cost computation equivalent to Equation 4.6 is described by

$$cost(e_i) = \begin{cases} c_5 \cdot (cl_{min} - cl(e_i)) & \text{if } cl(e_i) < cl_{min} \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

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.

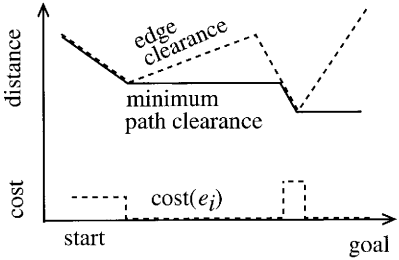

Fig. 11. Cost function for edge clearance

B. Combination of Criteria

Combining various costs into one function results in the optimization for multiple values, and is useful in many applications. The combined edge cost is expressed by

$$cost(e_i) = \sum_j w_j \cdot cost_j \tag{5.6}$$

where w_j is an appropriate weight and $cost_j$ is $cost(e_i)$ in Equations 5.1–5.5.

C. Simulation Results

Simulation results for maximizing minimum clearance and allowing a flexible final condition are presented.

1. Maximizing Minimum Clearance

Three different possible routes exist in the environment using the roadmap shown in Figure 13(a) from the start to goal area in Figure 13(c). Paths going through

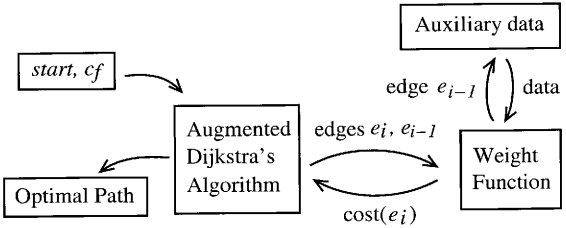

Fig. 12. Weighting with two adjacent edges and related data.

corridor A or C in Figure 13(c) are obtained by maximizing the minimum clearance or minimizing path length, respectively. Figure 13(c) shows the path going through corridor B; this is the result of combining the two conditions depicted in Equations 5.1 and 5.5.

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

Search tree edges of Dijkstra' algorithm are illustrated in Figure 13(b) by arrows representing the direction of the search from the start node.

Several simulations in the same environment are presented in Table 14 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 fourth row shows that the smoothest path is obtained by going though region B, which is shown in Figure 13(c). The fifth and sixth rows show that different combinations of weight constants can result in the similar paths.

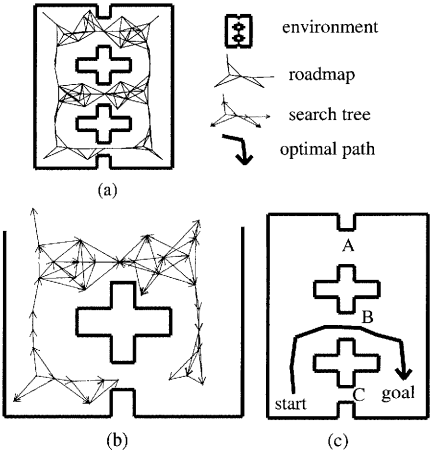

Fig. 13. Maximizing clearance and combination of criteria

2. Flexible Final Condition and Dynamic Constraints

A mobile platform and robotic arm with 3 links in an environment composed of three walls is illustrated in Figure 15. The wall in the middle has a passage, and each node's position in the roadmap (Figure 15(a)) indicates the mobile platform's center of mass. The start configuration is shown in Figure 15(b), and the final condition is that the end effector of the robotic arm should touch the wall opposite to the start position, and the mobile platform must come to a stop. The optimization criterion is time required. We use bang–bang control logic (move at full speed until the end

Route	$cost_{dist}$	$cost_{cl}$	$cost_{tr}$
A	0	1	0
B	0.03	0.97	0
C	1	0	0
B	0	0	1
B	0.08	0.84	0.08
B	0.03	0.32	0.65

Fig. 14. Simulations with different parameters

effector touches the wall, and apply the brake as hard as possible) to compute the minimum time of each path in Dijkstra's algorithm. We assume that the mass of the robot is small enough that it does not exceed the maximum deceleration rate and collide with the goal wall.

Depending on the weight (including payload) of the robot, final configurations will vary. In this case, we do not need to compute the optimal final configuration so long as possible goal configurations (shown in Figure 15(c)) are in the roadmap. Three different simulations show different final configurations in the paths with three different mass values.

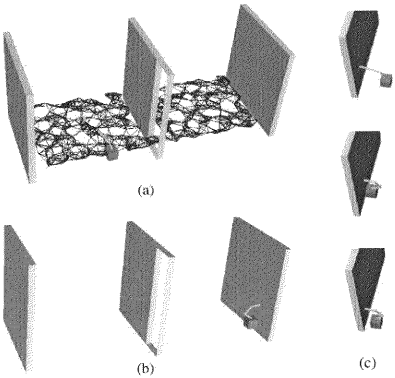

Fig. 15. Mobile platform with robotic arm. (a) Roadmap, (b) start position, and (c) three different configurations in the goal set.

CHAPTER VI

APPLICATION: CAMPUS NAVIGATOR

There are a number of applications of motion planning research. One application currently under development in the Parasol Lab is a campus path planner. This program will allow users to find their way across the Texas A&M University campus. Essentially, the campus navigator is similar to applications such as Yahoo! Map and MapQuest in that it provides users with directions (text and/or an image of the route) to get from one location on campus to another.

However, the campus navigator goes beyond the simple point-to-point route planning of these existing map programs. The campus navigator is designed to allow much more sophisticated queries tailored to the specific needs of the user. For instance, the campus navigator takes transportation mode changes into consideration. The user can specify if she will be walking, riding a bike, driving a car, or willing to take the bus.

As an example, consider a user wishing to find a route to get from a building on main campus to a building on west campus. There are a number of ways to accomplish this. One could simply walk to west campus. Using the campus navigator system, the user can find which bus(es) to take, where and when they stop, saving time and effort. The system will take into account driving conditions (i.e., closed streets due to construction), parking lots based on permit restrictions, and handicapped accessibility to provide the best path for the user.

A. Components of the Campus Navigator

Currently, the campus navigator is under development. A prototype of the system is expected to be ready by the end of the Spring 2003 semester. The next sections

describe the four fundamental components of the campus navigator system.

1. Overview of Campus Navigator

Before delving into each of these components, an overview of the system as a whole is in order. The user will interact with the system through a set of web pages. These pages allow the user to specify the start location and destination. The user's selection is sent to a program, query, that searches a preconstructed graph of campus (created via the roadmap editor). This graph, which is stored in a database, contains all the data needed by query to select a route that meets the user's request.

The path resulting from query's search of the graph is sent to Vizmo++, a visualization tool developed within the Parasol research group [18]. Using a 3D model of campus and the path, Vizmo++ creates a JPEG image that depicts the route through campus. This JPEG image is sent back to the user's browser. In addition to the image, a textual description of the route is provided. Figure 16 shows how all the components of our Campus Navigator system interact.

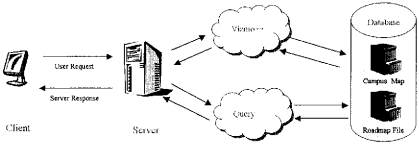

Fig. 16. Overview of Campus Navigator system.

2. The Campus Graph

The fundamental component of the campus navigator is a graph. As with many motion planning problems, this application is built upon the idea of finding a path through a graph. All the various constraints are expressed through properties of the vertices and weights on the edges of this graph. Vertices are used to represent physical places on campus such as buildings and parking lots. The edges of the graph are used to represent streets and walking paths through campus.

For this application, the graph is stored in a database. The current implementation employs the open source MySQL database management system. There are a number of reasons for storing the graph in a database. First, it allows concurrent access to the graph from the various components of the system. Initial designs called for the graph to be stored in a file, which is a customary storage medium for graphs.

Second, the database simplifies the sharing of data between the campus navigator system components. As an example, the roadmap editor, query, and the web interface need to access building names. The roadmap editor uses the names to allow vertices to be associated with buildings. Query employs building names when generating textual directions, and the web interface needs the names to give the user a list of buildings to choose from (for specifying start and/or destination). Each of these components are currently implemented in disparate languages. The database provided the simplest medium through which all three pieces could access the same data.

3. The Roadmap Editor

Currently, the campus graph is constructed manually. This is somewhat ironic as much of the work in motion planning is aimed at automatically creating a roadmap. Automatic construction of the campus graph is not realistic as most autogeneration

techniques rely on randomly generating vertices and connecting those vertices with edges. Random placement of vertices is not appropriate for this application, as vertices need to be tied to specific points on campus. For instance, a vertex needs to be associated with each building and parking lot.

To ease the construction of this large graph, a roadmap editor is currently under development. This program allows the campus graph to be built over an image of the campus. The user of this program can place vertices at each of the buildings, parking lots, intersections, etc. on campus by simply clicking on the building, parking lot, or intersection in the image. Then edges can be added for the streets and walking paths between these vertices. All of this information is stored in the centralized database.

Even if the campus graph could be generated randomly, all the properties of the vertices and edges of the graph would have to be manually specified. For instance, for a vertex representing a parking lot, someone must specify which permits (student, faculty/staff, etc.) are allowed to park in the lot. This is the second role of the roadmap editor. It allows this information to be entered for all the parts of the graph.

4. Campus Graph Query

The query program is responsible for finding routes through the graph that meet the user's request. This application employs Dijkstra's algorithm to find the optimal path through campus. This portion of Campus Navigator builds on the Path Optimization research presented in the previous chapters. Running as a service, the web interface will send requests to query and receive the computed path for Vizmo++ to display.

5. Campus Path Visualization Via Vizmo++

Vizmo++ is responsible for generating a picture of campus with the path overlayed [18]. After the user selects the start and goal points and query generates a path file, Vizmo++ opens up a 3D model of the campus and the path file to begin creating the snapshot. The camera then zooms onto the path and creates the image. This image is sent back to the web server to be displayed on the user's browser.

6. User Web Interface

Users of the campus navigator will interface with it via a set of web pages. Users will select the start and destination via selection boxes populated by data from the database. From the user selections, the web pages determine the vertices that correspond to the selected locations. These so-called start and goal vertices are given to query which finds the route. The web pages receive an image of the route from an image generated by Vizmo++. Figure 17 is a screenshot from a prototype of the Campus Navigator web interface.

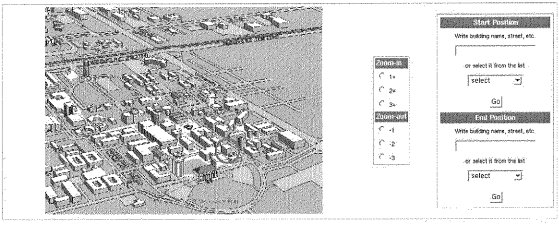

Fig. 17. Prototype of Campus Navigator interface

Initially, the web interface will provide simply a two dimensional map of campus

with the path overlayed. Similar to MapQuest and other programs, the campus navigator web interface allows users to zoom in and out on the returned campus path. Ultimately, it is envisioned that users would be able to generate a movie, allowing the user to "fly-through" a 3D model of campus along the path generated by query.

The campus navigator is an interesting and useful application of motion planning research. Exploiting the techniques developed to plan the paths of robots, the campus navigator aims to guide people around the large campus of Texas A&M University. It is envisioned that this application would potentially be useful for cities. By taking into consideration all the various modes of transportation such as buses and subways, the campus navigator could be extended to a city navigator, allowing residents and visitors to efficiently navigate the city.

CHAPTER VII

SUMMARY AND CONCLUSIONS

A framework for extracting an optimal path in a roadmap for motion planning has been created. 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 state and goal sets. Using *PRM*s, 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.

Application to the Campus Navigator was presented to demonstrate the many areas where path planning and optimization can be used. In this example, we harness the power and flexibility of graph search algorithms to allow customizable queries on a hand-made roadmap. The framework we created in the Campus Navigator can be extended to larger City Navigators.

A. Future Work

Future work consists of experimenting with robots with high degrees of freedom, explicit formulation of dynamic constraints, and hardware experiments using mobile robots. Also, planning paths for multiple robots sharing a roadmap will be considered in the future. Using many ideas presented in this thesis, in particular Flexible Goal Conditions, we may extract optimal paths and meeting or coordination points for multiple robots.

REFERENCES

[1] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *IEEE Trans. Robot. Automat.*, 1998, vol. 14, pp. 166–171.

[2] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2001, pp. 498–501.

[3] 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)*, 1999, pp. 2958—2963.

[4] Nancy M. Amato Jinsuck Kim, Roger A. Pearce, "Extracting optimal paths from roadmaps for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2003, To appear.

[5] Nancy M. Amato Jinsuck Kim, Roger A. Pearce, "Feature-based localization using scannable visibility sectors," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2003, To appear.

[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] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2002, pp. 624–631.

[8] Y. Wang and D. Lane, "Solving a generalized constrainted 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 and cybernetics, part C*, 2000, pp. 525–536.

[9] Donald E. Kirk, *Optimal Control Theory: An Introduction*, Prentice Hall, 1970.

[10] 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)*, 2001, pp. 1–8.

[11] C. Ahrikencheikh and A. Seireg, *Optimized-Motion Planning*, John Wiley & Sons, Inc., 1994.

[12] G. Song, S. L. Miller, and N. M. Amato, "Customizing PRM roadmaps at query time," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2001, pp. 1500–1505.

[13] P. Konkimalla and S.M. LaValle, "Efficient computation of optimal navigation functions for nonholonomic planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 1999, pp. 187–192.

[14] C. Hocaoglu and A. Sanderson, "Planning multiple paths with evolutionary speciation," in *IEEE transactions on evolutionary computation*, 2001, vol. 5, pp. 169–191.

[15] 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)*, 2002, pp. 794–799.

[16] 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), 2001, pp. 3789–3794.

[17] S. Lee, N. M. Amato, and J. P. Fellers, "Fast localization of mobile robots using visibility sectors," Tech. Rep., Department of Computer Science, Texas A&M University, 1999, Appeared in ICRA 2000.

[18] Bharatinder Sandhu, "A visualization tool to study the motion of complex 3d objects in space," *Senior Honors Thesis, University Undergraduate Fellows Program, Texas A&M University*, 2003.

VITA

Roger Allan Pearce began his undergraduate studies in the Fall of 1999. He is pursuing a Computer Engineering Degree from the Department of Electrical Engineering at Texas A&M University.

He has worked as an undergraduate researcher in the Parasol lab under Dr. Nancy Amato for two years. Roger is interested in many aspects of robots including motion planning, mechanical design, and robot-human interaction.

He plans to pursue a Masters of Computer Science after graduating from Texas A&M University.

Permanent address:

219 Wedgewood

Lake Jackson, TX 77566

The typist for this thesis was Roger Allan Pearce.