Numerical Computation of Optimal Navigation Functions on a Simplicial Complex

Steven M. LaValle, *Iowa State University, Ames, IA, USA*

This paper presents a general approach to computing optimal feedback motion strategies for a holonomic or nonholonomic robot in a static workspace. The proposed algorithm synthesizes a numerical navigation function defined by interpolation in a simplicial complex. The computation progresses much in the same way as Dijkstra's algorithm for graphs; however, the proposed approach applies to continuous spaces with geometric and nonholonomic constraints. By choosing a simplicial complex representation instead of a straightforward grid, the number of interpolation operations (which are required for continuous-state, numerical dynamic programming) is reduced from 2^n to $n+1$, in which n is the dimension of the configuration space. Some preliminary findings are discussed for an implementation of the algorithm for the case of a two-dimensional configuration space.

1 Introduction

Many applications in robotics require automatically computing a motion strategy that brings a complex geometric body from an initial configuration to a goal configuration while satisfying geometric constraints. Basic motion planning (or path planning) approaches typically do not apply in cases that are further complicated by imprecise localization and nonholonomic constraints. For example, in mobile robot or vehicle navigation, both the current and future configurations can only be estimated during execution using sensing and predictive models. A pre-planned path might have to be repeatedly adjusted due to such uncertainties; however, a preferable alternative would be to immediately know the optimal motion command given the current configuration (assuming the geometry and goal configuration are fixed). This corresponds to a feedback motion strategy, which is computed using the method presented in this paper. Arguments for directly incorporating feedback in the form of a navigation function were also made in [26]. Furthermore, applications continue to appear that involve the challenge of incorporating nonholonomic constraints on the system velocities (e.g., wheeled-robot systems [6, 17, 29], push planning [1, 23]).

The focus of this paper is on motion planning problems that require either feedback solutions, nonholonomic analysis, or both. Emphasis is placed on developing a technique that can be applied to a broad set of problems, as opposed to requiring particular simplifications that can be made for a particular problem. Although useful and interesting progress is often made by exploiting characteristics of a particular problem, a more general approach is often more likely to find immediate application, even if the approach is inferior for particular problems that are better solved by a specialized approach. In addition, one is often willing to exchange strong guarantees on performance (such as completeness) if the result is to obtain a method that is more broadly applicable. This philosophy shift was once seen in path planning research. Approaches such as the randomized potential field planner [2] or the randomized roadmap planner [11], offer weaker statements regarding the performance (i.e., probabilistic completeness as opposed to completeness); however, their general applicability and practical efficiency has stimulated their use in a variety of applications.

In this paper, a feedback strategy is computed numerically, which implies that the solution quality generally depends on resolution and quantization errors. This implies resolution completeness, and in the limiting case, the numerical solution converges to an optimal solution. This computational approach is also intended as a step toward a more general approach that can additionally handle complications such as stochastic or nondeterministic uncertainty in sensing and predictability, multiple robots, and dynamics. Such generalization can be made using the unified mathematical framework presented in [19].

Relationship to Dijkstra's algorithm The computational approach offered in this paper is built from the principle of optimality as it appears in optimal control theory [13], and is closely related to the level set method in [28]. It also shares similarities with Dijkstra's algorithm for computing single-source shortest paths in a graph; however, in our case a continuous configuration space must be considered (as opposed to a graph). It might seem possible to construct a graph by quantizing the configuration space to obtain vertices and using standard four-neighbors to obtain edges. However, the application of Dijkstra's algorithm (or wavefront propagation) to the resulting graph would not necessarily yield solutions that obey nonholonomic constraints. Furthermore, even results for holonomic problems will generally not be optimal because lengths are measured by forcing motions to occur along the grid (i.e., Manhattan distance is obtained instead of Euclidean distance). The approach proposed in this paper can be viewed as a way to make Dijkstra's algorithm yield the right results for nonholonomic motion planning problems in a continuous configuration space with geometric constraints. Instead of a graph or a grid, a simplicial complex is constructed; this is used to represent a continuous *cost-to-go function* (or navigation function) which serves the same purpose as the costs that are placed on vertices in Dijkstra's algorithm. Another related approach is the "continuous Dijkstra paradigm" [9, 25], which is used to efficiently compute holonomic shortest paths in a 2D polygonal environment by identifying critical events that correspond to a propagating wavefront of level-set curves that measure distance from a goal position.

Relationship to Barraquand-Latombe approach The proposed approach shares similarities with the nonholonomic planning approach used in [3, 24], which involves a branch-and-bound exploration from an initial configuration, using a discretized approximation of time and the set of possible control inputs. Each exploration step is achieved by applying a control input and integrating the system equation over a small time to obtain a new configuration. Within a given neighborhood, only a single configuration is saved. Eventually, the search expands until a configuration near the goal is reached. In our approach, a numerical navigation function is constructed by incremental expansion from a goal region as opposed to the initial configuration. Instead of keeping track of reachable configurations, our samples of the configuration space represent interpolation control points from which approximate values of the navigation function can be computed. Instead of determining a path, the navigation function is used as a representation of a *configuration-feedback* motion strategy that provides the optimal control input from anywhere in the configuration space. This becomes particularly valuable when there is a high degree of unpredictability in future configurations.

2 Problem Formulation

Basic concepts A workspace is defined that completely and precisely characterizes the environment of a robot. An n-dimensional *configuration space* $\mathcal{C}$ captures position, orientation, and/or other information for robot. Let $x \in \mathcal{C}$ denote a configuration. Let $\mathcal{C}_{free}$ denote the collision-free subset of the configuration space. Let $G \subset \mathcal{C}_{free}$ represent the *goal region*. Let u be an m-dimensional vector of control inputs. A control input could, for example, specify a particular steering angle for a car-like robot. Let $U \subseteq \Re^m$ represent the space of possible control inputs. The task is to determine control inputs that depend on a current configuration, "drive" the robot into the goal region, and optimize a specified criterion.

It will be helpful to define time in the upcoming concepts. Suppose that there is some initial time, $t = 0$, at which the robot is at x_{init}. Suppose also that there is some final time, T_f (one would like to at least have the robot at the goal before $t = T_f$). It will be assumed that the system is time-invariant (i.e., no decisions will depend on the particular time).

Incorporating nonholonomic constraints Let $\dot{x}$ represent the velocity of the robot in the configuration space. The nonholonomic constraints can be represented using control-theoretic notation as $\dot{x} = f(x(t), u(t))$ [10]. The case of $\dot{x}(t) = u(t)$ corresponds to a holonomic planning problem. Suppose that any control input is allowed if it satisfies $\|u\| = 1$ or $u = 0$. This implies that one can move the robot in any allowable direction at a local tangent space. A solution to a holonomic planning problem can be represented as a smooth path $h : [0, T_f] \rightarrow \mathcal{C}_{free}$ such that $x(0) = x_{init}$ and $x(T_f) \in G$. By setting the control input as $u(t) = \frac{dh}{dt}$, the original path can be obtained.

For nonholonomic problems, one will only be allowed to move the robot through a function of the form $\dot{x} = f(x(t), u(t))$. For example, as described in [15], p. 432, the equations for the nonholonomic car robot can be expressed as $\dot{x} = v\cos(\theta)$, $\dot{y} = v\sin(\theta)$, and $\dot{\theta} = \frac{v}{L}\tan(\phi)$, in which v is the speed of the rear axle midpoint, L is the distance between the front and rear axles, and ϕ is the steering angle. Using the notation in this paper, $(\dot{x}, \dot{y}, \dot{\theta})$ becomes $\dot{x}$, and (v, ϕ) becomes u. Other nonholonomic models, such as those used for push planning in [23] can be encoded in this way. The function f can be considered as a kind of interface between the user and the robot. Commands are specified through $u(t)$, but the resulting velocities in the configuration space get transformed using f (which can generally prevent the user from directly controlling velocities).

Incorporating feedback As stated in Section 1 the exact path executed by the robot often cannot be predicted in practice. Instead of using a predefined control input, $u(t)$, for all $t \in [0, T_f]$, it becomes necessary to allow the control input to depend on the particular configuration of the robot during execution. In an application, such as mobile robot navigation, a sensing module might be used to provide repeated estimates of the current configuration. The control input should be chosen in light of this estimated configuration. This motivates the definition of a *feedback motion strategy* as a function, $\gamma : \mathcal{C}_{free} \rightarrow U$. Thus, at a configuration $x \in \mathcal{C}_{free}$, $u = \gamma(x)$ specifies a particular control input. Note that the computational task is to compute a representation of a *function of configuration*, as opposed to a *path*, which is standard in motion planning.

Because analytical solutions are avoided, one is obligated at some point to approximately represent γ. This task requires special attention. Suppose that we would like to approximate γ using a discretized set of points and linear interpolation (which are used, for example, in the trapezoid rule for numerical integration). This becomes problematic because interpolations in U can be meaningless. For example, suppose $u \in U$ denotes "go forward" for a car-like robot, and $u' \in U$ denotes "go backwards." If $\gamma(x) = u$ and $\gamma(x') = u'$ for two close configurations (i.e., $\|x - x'\|$ is less than some small $\epsilon > 0$), then what would be the appropriate input for a configuration that lies between x and x'? It would certainly be senseless to compute a control input that denotes "stop" by interpolation. This would cause the car-like robot to be permanently fixed at the same configuration.

It will be seen in Section 3 that the approximation difficulty can be overcome by the use of a real-valued navigation function. The navigation function will be ultimately approximated by determining its values at vertices in a simplicial complex and using linear interpolation. The appropriate control input can be recovered during execution by performing some fast, local computations using the navigation function.

Incorporating optimality A *loss functional* is defined that evaluates any configuration trajectory and control function:

$$L = \int_0^{T_f} l(x(t), u(t))dt + Q(x(T_f)). \qquad (1)$$

The term $u(t)$ yields the control input applied at time t. With a given feedback motion strategy, γ, this would be chosen as $u(t) = \gamma(x(t))$. The integrand $l(x(t), u(t))$ represents an instantaneous cost, which when integrated can be imagined as the total amount of energy that is expended. The term $Q(x(T_f))$ is a final cost that can be used to induce a preference over trajectories that terminate in a goal region of the configuration space.

As an example, the following measures the path length for a trajectory that leads to the goal:

$$L = \begin{cases} \displaystyle\int_0^{T_f} \|\dot{x}(t)\| dt & \text{if } x(T_f) \in G \\ \infty & \text{otherwise} \end{cases} . \qquad (2)$$

The term $\int_0^{T_f} \|\dot{x}(t)\| dt$ measures path length.

Discrete-time approximation Since numerical computations will be performed, the continuous-time formulation will be approximated in discrete time. With the discretization of time, $[0, T_f]$ is partitioned into *stages*, denoted by $k \in \{1, \ldots, K+1\}$. Stage k refers to time $(k-1)\Delta t$. The final stage is given by $K = \lfloor T_f / \Delta t \rfloor$. Let x_k represent the configuration at stage k. At each stage k, an *action* u_k can be chosen from an *action space* U. Because

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}, \qquad (3)$$

the equation $\dot{x} = u$ can be approximated as

$$x_{k+1} = x_k + \Delta t \, u_k, \qquad (4)$$

in which $x_k = x(t)$, $x_{k+1} = x(t + \Delta t)$, and $u_k = u(t)$. The equation $\dot{x} = f(x(t), u(t))$ can be approximated by a transition equation of the form $x_{k+1} = f(x_k, u_k)$.

A discrete-time representation of the loss functional can also be defined:
$L(x_1, \ldots, x_{K+1}, u_1, \ldots, u_K) =$

$$\sum_{k=1}^{K} l_k(x_k, u_k) + l_{K+1}(x_{K+1}), \qquad (5)$$

in which l_k and l_{K+1} serve the same purpose as l and Q in the continuous-time loss functional.

A path-planning problem that does not consider optimality can be represented in discrete time by letting $l_k = 0$ for all $k \in \{1, \ldots, K\}$, and defining the final term as $l_{K+1}(x_{K+1}) = 0$ if $x_k \in G$, and $l_{K+1}(x_{K+1}) = 1$ otherwise. This gives equal preference to all trajectories that reach the goal. To approximate the problem of planning an optimal-length path, $l_k = 1$ for each $k \in \{1, \ldots, K\}$ such that $x_k \notin G$. The final term is then defined as $l_{K+1}(x_{K+1}) = 0$ if $x_{K+1} \in G$, and $l_{K+1}(x_{K+1}) = \infty$ otherwise.

3 Optimal Navigation Functions

Usually in motion planning the solution takes the form of a path; however, in our case the solution involves configuration feedback. One possibility is that the algorithm can return a function that maps configurations into actions. Thus from any location in the configuration space, the robot will have the appropriate action during the execution of the optimal strategy.

This section presents the concept of *cost-to-go* functions, which are used as a navigation function in the sense proposed in [26]. Instead of building a direct representation of the feedback strategy, the proposed approach uses a cost-to-go function as an intermediate representation from which the appropriate action can quickly be obtained. It will be seen that this approach is advantageous for the numerical computations because optimal actions can be selected by performing interpolations on the cost-to-go function. It is generally inappropriate to "interpolate" on the space of possible actions.

The principle of optimality Initially, a cost-to-go function will be defined for each stage k. A cost-to-go function will eventually be obtained that is stage independent. The cost-to-go function $L_k^* : \mathcal{C}_{free} \rightarrow \Re \cup \{\infty\}$ represents the loss that will accumulate if the optimal trajectory is executed from stage k until stage $K+1$, starting at configuration $x_k \in \mathcal{C}_{free}$. Note that $L_{K+1}^* = l_{K+1}$ from (5). If K is sufficiently large, then for reasonably-behaved planning problems there exists an $i < K$ such that $L_k^* = L_i^*$ for all $i < k$. This will hold for problems in which: 1) all optimal trajectories that reach the goal arrive in a finite amount of time; 2) infinite loss is obtained for a trajectory that fails to reach the goal; 3) no loss accumulates while the robot "waits" in the goal region; 4) the environment and motion models are stationary.

Bellman's principle of optimality provides a powerful constraint on the solution structure [4], which directly leads to a numerical computation approach. With a discrete-time model, a difference equation is obtained that relates successive cost-to-go functions. This difference equation will now be derived. The *cost-to-go* function at stage k is formally defined as

$$L_k^*(x_k) = \min_{u_k,\ldots,u_K} \left\{ \sum_{i=k}^{K} l_i(x_i,u_i) + l_{K+1}(x_{K+1}) \right\}. \tag{6}$$

The term $\gamma_i^*(x_i)$ represents u_i, the action chosen at stage i from configuration x_i. Equation (6) represents the loss that will be received under the execution of the optimal strategy, γ^*, from stage k to stage $K+1$.

The cost-to-go can be separated:
$L_k^*(x_k) = \min_{u_k} \min_{u_{k+1},\ldots,u_K}$

$$\left\{ l_k(x_k,u_k) + \sum_{i=k+1}^{K} l_i(x_i,u_i) + l_{K+1}(x_{K+1}) \right\}. \tag{7}$$

The second *min* does not affect the l_k term; thus, it can be removed to obtain
$L_k^*(x_k) =$

$$\min_{u_k} \left[l_k(x_k,u_k) + \min_{u_{k+1},\ldots,u_K} \left\{ \sum_{i=k+1}^{K} l_i(x_i,u_i) + l_{K+1}(x_{K+1}) \right\} \right]. \tag{8}$$

The second portion of the *min* represents the cost-to-go function for stage $k+1$, yielding:

$$L_k^*(x_k) = \min_{u_k} \left\{ l_k(x_k,u_k) + L_{k+1}^*(x_{k+1}) \right\}. \tag{9}$$

This final form represents a powerful constraint on the optimal strategy. The optimal action at stage k and configuration x depends only on the cost-to-go values at stage $k+1$. Furthermore, only the particular cost-to-go values that are reachable from the transition equation $x_{k+1} = f(x_k,u_k)$ need to be considered. The dependencies are local; yet, the globally-optimal strategy is characterized. This property will be exploited by the methods presented in Section 4.

Using the cost-to-go as a navigation function As stated previously, if K is sufficiently large, there exists an i such that for all $k \leq i$ and all $x \in \mathcal{C}_{free}$, $L_k^*(x_k) = L_i^*(x_i)$. In this case, L^* will be used instead of L_i^* because the cost-to-go is stage-invariant.

Suppose that L^* has been numerically computed, and that values of $L^*(x)$ for any $x \in \mathcal{C}_{free}$ can be obtained through a quantization and interpolation scheme (which will be discussed in Section 4). To execute the optimal strategy, an appropriate action must be chosen using the cost-to-go representation from any given configuration. One approach would be to simply store the action that produced the optimal cost-to-go value for each quantized configuration. The appropriate action could then be selected by recalling the stored action at the nearest quantized configuration. This method could cause errors, particularly since it does not utilize any benefits of interpolation. A preferred alternative is to select actions by locally evaluating (9) at the current configuration. Interpolation schemes can also be used in this step. Note that although the approach to select the action is local (and efficient), the global information is still taken into account (it is encoded in the cost-to-go function). Once the optimal action is determined, the next configuration is obtained (i.e., not a quantized configuration) using $x_{k+1} = f(u_k, x_k)$. This form of iteration continues until the goal is reached or a termination condition is met. Note that this scheme does not force the robot to traverse quantized configurations.

During the time between stages, the trajectory can be linearly interpolated between the endpoints given by the discrete-time transition equation, or can be integrated using the original continuous-time transition equation. Once the optimal action is determined, an exact next configuration is obtained (i.e., not a quantized configuration). This form of iteration continues until the goal is reached or a termination condition is met.

4 Computational Approach

This section presents an algorithm for computing the optimal navigation function L^* for a given problem. The optimal motions are executed by selecting actions using L^* as a navigation function, as discussed in Section 3. Two basic methods are described. The first method is derived from numerical dynamic programming techniques presented in optimal control literature

[13, 14]. The method described here and several extensions were implemented and tested on many problems in [18, 20, 21]. The purpose of describing it here is to provide a basis of comparison for the new method, for which a description soon follows. The new method uses many of the same assumptions as the existing method, but computes the result with much better computational performance.

4.1 Previously established approach

For the purpose of discussion, it will be assumed that $\mathcal{C}_{free}$ is a subset of Euclidean space. Topological issues that arise on manifolds such as $\Re^2 \times S^1$ (the case of translation and rotation in the plane) will not be treated here; however, only minor notational variations are required to include such cases.

An optimal strategy can be computed by successively building approximate representations of the cost-to-go functions [18]. The most straightforward way to represent a cost-to-go function is to specify its values at each location in a discretized grid. The first step is to construct a representation of L^*_{K+1}. The final term, $l_{K+1}(x_{K+1})$, of the loss functional is directly used to assign values of $L^*_{K+1}(x_{K+1})$ at discretized locations. Typically, $l_{K+1}(x_{K+1}) = 0$ if x_{K+1} lies in the goal region, and $l_{K+1}(x_{K+1}) = \infty$ otherwise. This only permits trajectories that terminate in the goal region. If the goal is a point, it might be necessary to expand the goal into a region that includes some of the quantized configurations.

The dynamic programming equation (9) is used to compute the next cost-to-go function, L^*_K, and subsequent cost-to-go functions. For each quantized configuration, x_k, a quantized set of actions, $u_k \in U$, are evaluated. For a given action u_k, the next configuration obtained by $x_{k+1} = f(x_k, u_k)$ generally might not lie on a quantized configuration. However, linear interpolation between neighboring quantized configurations can be used to obtain the appropriate loss value without restricting the motions to the grid. Suppose for example, that for a one-dimensional configuration space, $L^*_{k+1}[i]$ and $L^*_{k+1}[i+1]$ represent the loss values for some configurations x_i and x_{i+1}. Suppose that the transition equation, f, yields some x that is between x_i and x_{i+1}. Let

$$\alpha = \frac{x_{i+1} - x}{x_{i+1} - x_i}. \tag{10}$$

Note that $\alpha = 1$ when $x = x_i$ and $\alpha = 0$ when $x = x_{i+1}$. The interpolated loss can be expressed as

$$L^*_{k+1}(x_{k+1}) \approx \alpha L^*_{k+1}[i] + (1-\alpha) L^*_{k+1}[i+1]. \tag{11}$$

In an n-dimensional configuration 1space, interpolation can be performed between 2^n neighbors. For example, if $\mathcal{C} = \Re^2$, the interpolation can be computed as

$$L^*_{k+1}(x_{k+1}) \approx \alpha\beta L^*_{k+1}[i,j] + (1-\alpha)\beta L^*_{k+1}[i+1,j] + \alpha(1-\beta) L^*_{k+1}[i,j+1] + (1-\alpha)(1-\beta) L^*_{k+1}[i+1,j+1] \tag{12}$$

in which $\alpha, \beta \in [0,1]$ are coefficients that express the normalized distance to the neighbors. Convergence properties of the quantization and interpolation are discussed in [4, 5]. Interpolation represents an important step that overcomes the problems of measuring Manhattan distance due to quantization.

The obstacle constraints must also be taken into account. The constraints can be directly evaluated each time to determine whether each x_{k+1} lies in the free space, or a bitmap representation of the configuration space can be used for quick evaluations (an efficient algorithm for building a bitmap representation of $\mathcal{C}_{free}$ is given in [12]).

Note that L^*_K represents the cost of the optimal one-stage strategy from each configuration x_K. More generally, L^*_{K-i} represents the cost of the optimal $(i+1)$-stage strategy from each configuration x_{K-i}. For a motion planning problem, one is typically concerned only with strategies that require a finite number of stages before terminating in the goal region. For a small, positive δ the dynamic programming iterations are terminated when $|L^*_k(x_k) - L^*_{k+1}(x_{k+1})| < \delta$ for all values in the configuration space. This assumes that the robot is capable of selecting actions that halt it in the goal region. The resulting stabilized cost-to-go function can be considered as a representation of the optimal strategy. Note that no choice of K is necessary because termination occurs when the loss values have stabilized. Also, only the representation of L^*_{k+1} is retained while constructing L^*_k; earlier representations can be discarded to save storage space.

4.2 The new approach

The approach outlined in Section 4.1 will numerically determine the optimal motion strategy for a very broad class of problems. It can furthermore be extended in a variety of ways to handle complications such as stochastic predictability uncertainty [18]. In spite of these successes, its computational complexity usually limits its practical applicability to problems that have two or three degrees of freedom.

The following two aspects of the previous method greatly contribute to the computational expense: 1) The number of iterations over the configuration space is equal to the number of stages in the longest (in terms of time) optimal trajectory that reaches the goal; 2) Each evaluation of $L^*(x)$ requires an interpolation between 2^n neighbors if $\mathcal{C}_{free}$ is n-dimensional. The new approach obtains the optimal strategy in a single iteration over the configuration space and uses only $n+1$ neighbors for each interpolation. The first improvement is made by styling the computation in a manner that is similar to the execution of Dijkstra's algorithm for graphs. The second improvement is made by approximating the cost-to-go function by constructing a simplicial complex on the configuration space.

Using a simplicial complex A set of *control points*, $p \in \mathcal{C}_{free}$, will be introduced to replace grid points from the previous approach. For a set of control points, $p_1, p_2, \ldots, p_{n+1}$, let $[p_1, p_2, \ldots, p_{n+1}]$ denote their convex hull, which will be referred to as a *simplex*. In two dimensions, a triangulation is obtained. For an n-dimensional configuration space, a *simplicial complex* is constructed [27]. Every d-dimensional simplex has $d+1$ faces, each of which are $(d-1)$-dimensional simplexes. Furthermore, for every pair, S_1, S_2, of i-dimensional simplexes in the simplicial complex, either $S_1 \cap S_2 = \emptyset$ or S_1 and S_2 share an $(i-1)$-dimensional face.

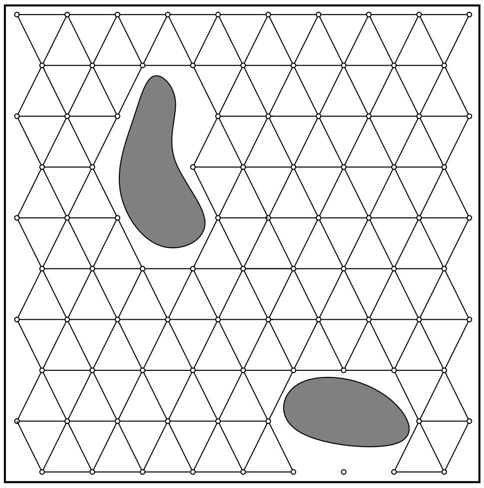


Figure 1: *Simplexes are only constructed in the obstacle-free portion of $\mathcal{C}_{free}$.*

The obstacle regions are avoided in the construction of the simplicial complex, as shown in Figure 1. In practice one might simply generate a simplex if all of its corresponding control points lie in $\mathcal{C}_{free}$. This approach might be efficient; however, to ensure correctness of the results, one must determine whether the entire convex hull of the control points lies in $\mathcal{C}_{free}$. One way this can be accomplished is to use a collision checking algorithm that provides a lower bound on the distance from the point to the obstacle in the configuration space.

Next, consider using the simplicial complex to represent a cost-to-go function. Suppose that the value of L^* is known for each of the control points. For an n-dimensional configuration space, the value of L^* at a point $x \in \mathcal{C}_{free}$ can be estimated through interpolation. Suppose x lies in the interior of an n-dimensional simplex. Note that any point x in a simplex can be uniquely expressed as a linear combination of the control points, $\{p_1, \ldots, p_{n+1}\}$ as

$$x = \sum_{i=0}^{n+1} \beta_i p_i, \tag{13}$$

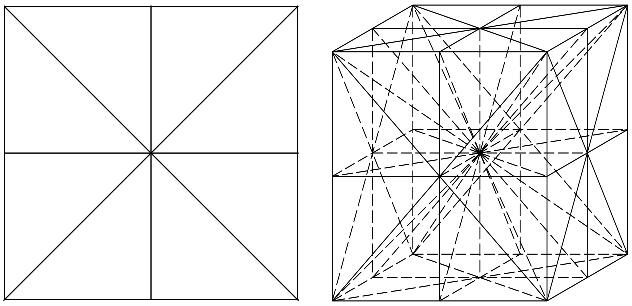
in which each β_i is real-value such that $\beta_i \geq 0$ and $\sum_{i=0}^{n+1} \beta_i = 1$. The coefficients $\{\beta_1, \beta_2, \ldots, \beta_{n+1}\}$ are actually the *barycentric coordinates* of x. These coordinates can be used to provide a estimate of L^* through linear interpolation as

$$L^*(x) \approx \sum_{i=0}^{n+1} \beta_i L^*(p_i). \tag{14}$$

The next question that remains is how to choose a simplicial complex. Generally, three factors should guide the selection of a simplicial complex: 1) to yield a nice representation of the navigation function, each simplex should be as close to spherical in shape as possible, 2) for a given configuration, the problem of determining which simplex it belongs to should be efficient, and 3) the computation of the barycentric coordinates should be efficient. For the 2D case it is straightforward to define a simplicial complex by simply tiling the plane with equilateral triangles; however, in higher dimensions, the computational performance will depend greatly on the choice of simplexes that cover the configuration space. Ideally, one would like to construct a simplicial complex such that all d-dimensional simplexes are regular polytopes; however, as shown in [7], this is impossible, even in $\Re^3$ (e.g., $\Re^3$ cannot be tiled with identical tetrahedra, each of which having all faces be equilateral triangles).

We have recently shown that cubes or hypercubes can be triangulated using barycentric subdivision to yield a simplicial complex with very desirable properties: point location (which simplex does a given point belong to?) and barycentric coordinate computation can be performed in time $O(n \lg n)$, in which n is the dimension of the configuration space. The configuration space is tiled with cubes, and each cube is triangulated as shown in Figure 2 for the 2D and 3D cases. This result will be useful for the implementation in higher-dimensional configuration spaces, but at the present time we have only implemented the algorithm for the 2D case, which is described in Section 5.

Algorithm details An overview of the algorithm is given in Figure 3. The computation proceeds in a manner similar to Dijkstra's algorithm, except in our case, the *region* in which L^* is known increases incrementally, as opposed to the costs on *vertices* in a graph.



(a) (b)

Figure 2: *The cube can be triangulated using barycentric subdivision to obtain a simplicial complex on which efficient point location and barycentric coordinate computation are possible.*

$Q \leftarrow \{\}$; $P_f \leftarrow$ all $p \in G$
For each $p \in Pre(G) \setminus G$
    Compute $lub(p)$; INSERT(p,Q)
Until $Q = \emptyset$ **do**
    $p_{min} \leftarrow$ POP(Q)
    For each p in $Q \cap (Pre(R(P_f \cup p_{min}) \setminus R(P_f)))$
        Recompute $lub(p)$
    $P_f \leftarrow P_f \cup \{p_{min}\}$
    For each $p \in Pre(R(P_f)) \setminus Q$
        Compute $lub(p)$
        INSERT(p,Q)

Figure 3: *This algorithm computes the optimal feedback motion strategy in a single pass over the configuration space.*

For some loss functionals, the computation can be further simplified. For example, in the case of a minimum-time criterion, the computation can be organized as a wavefront expansion. This is analogous to the simplification of Dijkstra's algorithm to wavefront propagation in standard motion planning approaches.

The following notation is used in the algorithm description. For any control point p, let $lub(p)$ denote the lowest upper bound that can be assigned to the optimal cost-to-go at p, given any computation that has been performed. Let P_f denote a set of control points for which $L^*(x)$ is known. In other words, $lub(p) = L^*(p)$ for all $p \in P_f$. For a set of control points, P, let $R(P) \subseteq \mathcal{C}_{free}$ denote the set of all points that lie in a simplex for which all of its control points are contained in P. In other words, R identifies a region over which a cost could be computed through interpolation of control points in P. Let $Pre(C)$ denote the set of all $x_{k+1} \in \mathcal{C}_{free}$ such that there exists some $u_k \in U$ with $x_{k+1} = f(x_k, u_k)$ and $x_k \in C$. In other words, $Pre(C)$ gives the set of configurations from which the set C is reachable in a single stage. This is a familiar construction in motion planning under uncertainty [22, 16], and the computation of $Pre(C)$ might be straightforward or extremely challenging, depending on the particular motion model.

Initially, it is assumed that $lub(p) = \infty$ if $p \notin G$. If $p \in G$, then $lub(p) = L^*(p) = 0$ (i.e., the optimal cost-to-go from the goal is always zero). The objective is to determine $L^*(p)$, for any control point p, that is reachable from G. Once this occurs, P_f will include all control points ($L^*(p) = \infty$ for unreachable control points), and $R(P_f)$ will be the maximal subset, formed from simplexes, on which the optimal cost-to-go can be determined. If there does not exist enough points initially such that 0 is obtained by interpolation over an n-dimensional simplex, then the algorithm will fail. This can occur, for instance, if the goal region is small and the spacing between control points is too large.

Step 1 initializes Q, which is a priority queue of control points that are sorted in ascending order according to $lub(p)$. Also, P_f is assigned to contain all control points that lie in G. Steps 2 and 3 initialize Q by inserting all control points that are reachable from G in a single stage, excluding points already in G. For points in $Pre(G)$, a finite value for $lub(p)$ can be computed by selecting the action $u_k \in U$ that produces the minimum loss. This computation can generally be performed by discretizing the action space; however, better techniques can be used in some particular cases.

Steps 5 to 11 are iterated until Q is empty. After each iteration, L^* is known for a new control point. In Step 3, p_{min} is removed from Q (the control point for which $lub(p)$ is the smallest). It is known that $lub(p_{min}) = L^*(p_{min})$ because a single-stage trajectory exists that brings p_{min} into $R(P_f)$ with less loss than from any other control point in Q. Once L^* is known for a new control point, P_f must be appropriately expanded, which adds new simplexes over which L^* can be obtained through interpolation. Better upper bounds can now be computed for some of the control points in Q. This update is performed by trying the actions $u_k \in U$ and evaluating the loss obtained for configurations that lie in the new simplexes (i.e., if p_{min} was used in the interpolation). After the control points are updated, the next part is to add new elements to Q, which is performed in Steps 9-11.

The priority queue, Q, is empty when there are no new control points that can reach $R(P_f)$ in a single stage. This is a natural halting point for the algorithm. The algorithm only stores the L^* for all control points in P_f. The cost-to-go at any configuration in $R(P_f)$ can be obtained by linear interpolation, which results in a navigation function that can be used to guide the robot into the goal.

5 Implementation

This section presents some results from an implementation of the algorithm in Figure 3 using GNU C++ and LEDA under Linux on a 200Mhz Pentium Pro PC. The current implementation applies only to case of a planar configuration space with a holonomic motion model. Thus, many practical computational issues remain to be explored as the dimension of the configuration space is increased and different nonholonomic systems are considered. For a small set of problems, it has been observed that the new computation method computes optimal navigation functions between 40 and 80 times faster than the method described in [18].

Three example problems are shown in Figures 4.a, 5.a, and 6.a. Figures 4.b, 5.b, and 6.b show level set contours of the optimal navigation function that was obtained using 80×80 control points. Figure 4.c shows 100 paths that were obtained by using the computed navigation function for Problem 1 and starting from random initial configurations. The table shown in Figure 7 shows computation times. A point robot is assumed. The extremal values of each axis are 0 and 100. When 40×40 control points were used, the robot was allowed to move 3.75 units in each stage. The state

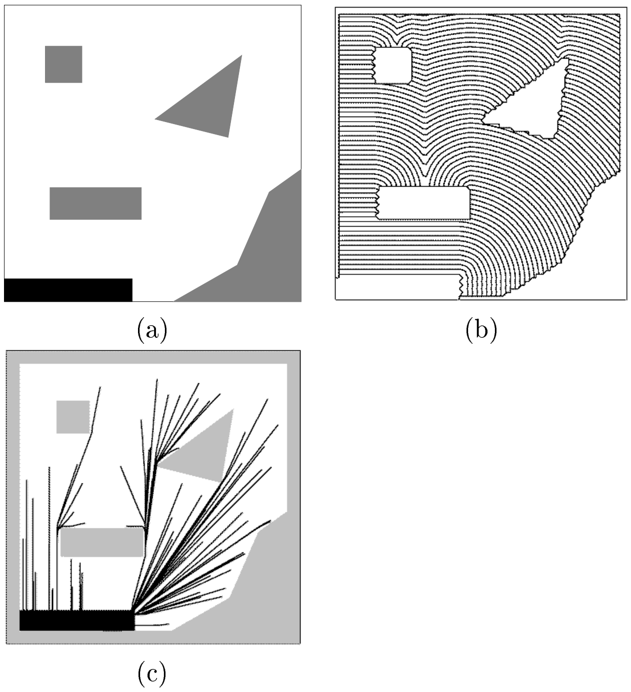


Figure 4: *Problem 1: a) The obstacles are shown in gray, and the goal region is shown in black; b) level set contours of the optimal navigation function for each integer, $\{1, 2, 3, \ldots\}$; c) 100 paths that were obtained by using the computed navigation function from randomly chosen initial positions.*

transition equation, f, is defined such that the robot is allowed to either move a fixed distance in any direction or remain stationary. When 80×80 control points were used, the robot was allowed to move 1.875 units in each stage. The action space was quantized into 32 and 360 orientations; the cost-to-go for each quantized value is computed, to numerically evaluate (9).

For this particular two-dimensional problem, one could evaluate (9) more efficiently, or even use methods similar to those in [9, 25]. The purpose of these experiments, however, is to estimate how the algorithm will perform when it is given more challenging problems in higher-dimensional configuration spaces. Therefore, no attempt was made to exploit properties particular to the 2D holonomic shortest-path problem.

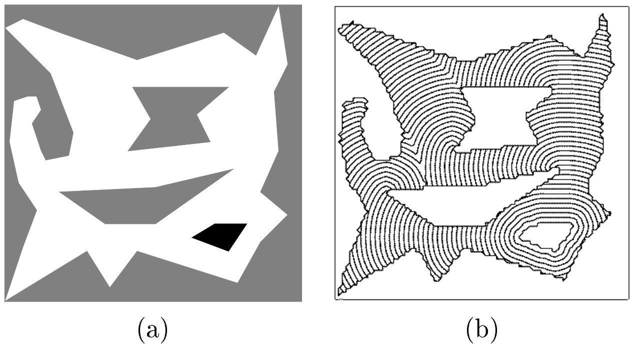


Figure 5: *Problem 2: a) Input; b) level set contours.*

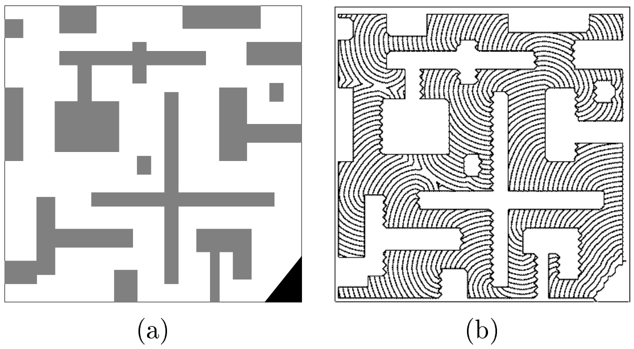


Figure 6: *Problem 3: a) Input; b) level set contours.*

6 Discussion

A new approach has been presented for computing optimal feedback motion strategies for holonomic or nonholonomic planning problems. Instead of precomputing a path, the solution takes the form of a navigation function, as considered in [26]. Two improvements are made over the previous approach to the problem of numerically computing optimal cost-to-go functions: 1) The solution can be obtained in a single iteration over the configuration space; 2) the number of interpolation steps have been reduced from 2^n to $n + 1$. It is important to note that once a navigation function is computed, it can be quickly utilized during execution. Although great performance improvement of the previous technique has been observed for a 2D configuration space, it is important to test these ideas on higher-dimensional spaces. As the dimension is in-

P	Res	Precmp	OldDP	NewDP	Fact
	$\lvert I\rvert x\lvert J\rvert, \lvert U\rvert$	(sec)	(sec)	(sec)	
1	$40 \times 40, 32$	0.45	10.32	0.20	51.6
1	$80 \times 80, 32$	1.77	84.16	0.82	102.6
1	$80 \times 80, 360$	1.77	822.93	8.45	97.4
2	$40 \times 40, 32$	0.61	8.03	0.18	44.6
3	$40 \times 40, 32$	1.37	12.57	0.21	59.8

Figure 7: *Computational performance from a GNU C++ implementation on a 200Mhz Pentium Pro PC running Linux. The columns denote: P = problem number, Res = resolutions, Precmp = precomputation time, OldDP = computation time for the original dynamic programming algorithm, NewDP = computation time for the new algorithm, Fact = factor of improvement in computation time.*

creased, greater benefits of using the simplicial complex as opposed to a grid are expected.

One aspect that deserves careful attention is the sensitivity of the approach to the choices of resolutions. To improve computational performance, it is tempting to increase the spacing between control points. However, this might result in a simplicial complex that does not adequately capture the topological information in $\mathcal{C}_{free}$ (i.e., a path class might be lost). It is also important to appropriately select Δt, which directly affects the distance traveled during each stage. If Δt is too small, the algorithm is likely to terminate prematurely because $R(P_f)$ will not be reachable. If Δt is too large, goal overshoot might occur. Similar issues also exist regarding the quantization of the action space U during the numerical evaluation of (9).

We are currently implementing the algorithm presented in Figure 3 for a three-dimensional configuration space, and intend to perform experiments to determine its performance and robustness. Two applications are worth considering at the outset: 1) optimal planning for car-like robots, and 2) optimal push planning for a robot that pushes rigid bodies on a planar surface. Our approach is expected to be particularly useful for applications that involve unpredictability, such as nonprehensile manipulation [8]. For example, imagine pushing a box with a mobile robot that makes points contact. A vision system can be used to monitor the position of the box, and the motion of the robot should be adjusted during execution to guide the box to the goal. A navigation function would be ideal in this case because a feedback motion strategy is directly obtained, as opposed to forcing the robot to follow a particular trajectory.

In the longer term it will be interesting to attempt generalizations of the algorithm. As argued in [19], one powerful advantage of a unified mathematical framework for motion planning problems is the relative ease of obtaining generalizations. It is expected that this will indeed be the case for the algorithm presented in this paper. One possible generalization is to use heuristic underestimates of the cost-to-go to design an algorithm that is similar to the A^* search generalization of Dijkstra's algorithm. The technique might also apply to problems that involve stochastic uncertainty in predictability, which can dramatically improve computational performance for problems such as computing optimal manipulation strategies under uncertainty [20], or planning in a partially-predictable environment [21].

Acknowledgments

I thank Prashanth Konkimalla for his help with the implementation. I also thank Tammy Verstraete for making corrections to this manuscript.

References

[1] P. K. Agarwal, J.-C. Latombe, R. Motwani, and P. Raghavan. Nonholonomic path planning for pushing a disk among obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 1997.

[2] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. Syst., Man, Cybern.*, 22(2):224–241, 1992.

[3] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[4] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[5] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Trans. Autom. Control*, 20(3):415–419, June 1975.

[6] L. G. Bushnell, D. M. Tilbury, and S. S. Sastry. Steering three-input nonholonomic systems: the fire truck example. *Int. J. Robot. Res.*, 14(4):366–381, 1995.

[7] H. S. M. Coxeter. *Regular Polytopes*. Dover Publications, New York, NY, 1973.

[8] M. Erdmann. An exploration of nonprehensile two-palm manipulation using two zebra robots. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 239–254. A K Peters, Wellesley, MA, 1997.

[9] J. Hershberger and S. Suri. Efficient computation of Euclidean shortest paths in the plane. In *Proc. 34th Annual IEEE Sympos. Found. Comput. Sci.*, pages 508–517, 1995.

[10] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, Berlin, 1989.

[11] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1994.

[12] L. E. Kavraki. Computation of configuration-space obstacles using the Fast Fourier Transform. *IEEE Trans. Robot. & Autom.*, 11(3):408–413, 1995.

[13] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Trans. Autom. Control*, 12(6):767–774, December 1967.

[14] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II*. Dekker, New York, NY, 1982.

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

[16] J.-C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artif. Intell.*, 52:1–47, 1991.

[17] J.-P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Trans. Robot. & Autom.*, 10(5):577–593, October 1994.

[18] S. M. LaValle. *A Game-Theoretic Framework for Robot Motion Planning*. PhD thesis, University of Illinois, Urbana, IL, July 1995.

[19] S. M. LaValle. Robot motion planning: A game-theoretic foundation. In *Proc. 2nd Int'l Workshop on the Algorithmic Foundations of Robotics*, July 1996.

[20] S. M. LaValle and S. A. Hutchinson. An objective-based framework for motion planning under sensing and control uncertainties. *International Journal of Robotics Research*, 17(1):19–42, January 1998.

[21] S. M. LaValle and R. Sharma. On motion planning in changing, partially-predictable environments. *International Journal of Robotics Research*, 16(6):775–805, December 1997.

[22] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic systhesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1):3–24, 1984.

[23] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. In *Algorithmic Foundations of Robotics*. A. K. Peters, Boston, 1995.

[24] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robot. Res.*, 15(6):533–556, 1996.

[25] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Int. J. Comput. Geom. & Appl.*, 6(3):309–332, 1996.

[26] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Trans. Robot. & Autom.*, 8(5):501–518, October 1992.

[27] J. J. Rotman. *Introduction to Algebraic Topology*. Springer-Verlag, Berlin, 1988.

[28] J. A. Sethian. *Level set methods : Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1996.

[29] P. Svestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE Int. Conf. Robot. & Autom.*, pages 1631–1636, 1995.