Differentially Constrained Mobile Robot Motion Planning in State Lattices

Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly
Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
e-mail: mihail@cs.cmu.edu, rak@ri.cmu.edu, alonzo@ri.cmu.edu

Received 6 August 2008; accepted 4 January 2009

We present an approach to the problem of differentially constrained mobile robot motion planning in arbitrary cost fields. The approach is based on deterministic search in a specially discretized state space. We compute a set of elementary motions that connects each discrete state value to a set of its reachable neighbors via feasible motions. Thus, this set of motions induces a connected search graph. The motions are carefully designed to terminate at discrete states, whose dimensions include relevant state variables (e.g., position, heading, curvature, and velocity). The discrete states, and thus the motions, repeat at regular intervals, forming a lattice. We ensure that all paths in the graph encode feasible motions via the imposition of continuity constraints on state variables at graph vertices and compliance of the graph edges with a differential equation comprising the vehicle model. The resulting state lattice permits fast full configuration space cost evaluation and collision detection. Experimental results with research prototype rovers demonstrate that the planner allows us to exploit the entire envelope of vehicle maneuverability in rough terrain, while featuring real-time performance.

1. INTRODUCTION

Capable motion planners are important for enabling field robots to perform reliably, efficiently, and intelligently. Despite decades of significant research effort, today the majority of field robots still exhibit various failure modes due to motion planning deficiencies. These failure modes range from computational inefficiencies to frequent resort to operator involvement when the autonomous system takes unnecessary risks or fails to make adequate progress. On the basis of our extensive field robotics experience, we have developed a motion planning method that addresses the drawbacks of leading approaches. We have demonstrated it here to be superior to state of the art. It is a deterministic, sampling-based method that features a particular sampling of robot state space, which lends itself well to enabling an array of performance capabilities.

Discrete representation of robot state is a well-established method of reducing the computational complexity of motion planning. This reduction comes at the expense of sacrificing *feasibility* and *optimality*, the notions denoting the planner's capacity to compute a motion that satisfies given constraints and to minimize the cost of the motion, respectively. The proposed method is based on a particular discretization of robot state space, the *state lattice*. It is used to

formulate the problem of motion planning as graph search, and so it will be referred to as a *search space*. In computing motions, we seek to satisfy two types of constraints: avoiding the features of the environment that limit the robot's motion (*obstacles*) and the limitation of the robot's mobility due to the constrained dynamics of its motion (*differential constraints*). Motions that satisfy both types of constraints will be referred to as *feasible* motions.

The state lattice is a regular sampling of the state space. It encodes a graph whose vertices are a discretized set of all reachable states of the system and whose edges are feasible motions, *controls*, which connect these states exactly. The motions encoded in the edges of the state lattice form a repeating unit that can be copied to every vertex, while preserving the property that each edge joins neighboring vertices exactly. This property of the search space will be denoted *regularity*. The canonical set of repeating edges will be called the *control set*. The number of edges in the control set is exactly the branching factor, *outdegree*, of each vertex in the reachability graph.

1.1. Motivation

This work is motivated by a number of robotics applications. In the context of terrestrial unmanned ground vehicles, competence in motion planning in dense obstacle fields demands the use of sufficiently high-fidelity, constrained-motion models, at least in the near field. Repeated attempts to get by with less have not been particularly successful (Kelly et al., 2004). A classic case is that of a vehicle with Ackerman (or car-like) steering that drives into a winding corridor only to find it closed off. In this case, the robot must turn around using many velocity reversals while avoiding the obstacles surrounding it. This application motivates high-fidelity representation of robot maneuverability.

Another case of contemporary interest is that of a planetary rover surveying a rocky field. It is often necessary to approach a target rock in a particular configuration and with high precision, in order to deploy onboard instruments. The capacity of the motion planner to enable the accomplishment of this mission depends on generating a path that can be followed reliably with precision. As above, understanding and utilizing robot maneuverability while minimizing computational requirements is critical to accomplishing such missions. Failure of the robot to compute its own motion is likely to lead to system failure that requires operator involvement to resolve. Human participation, even via teleoperation, significantly reduces the utility of the robotic system.

Practical approaches to motion planning in difficult field applications differ from the focus of contemporary theoretical research in several significant ways. Difficult three-dimensional (3D) environments such as rolling vegetated terrain, boulder fields, and forests cannot usually be partitioned into obstacles and nonobstacles because the system must reason about the relative risk of candidate paths, rather than which physical objects will prevent motion. A workspace (i.e., x,y) cost map has been a preferred environmental map representation for us and others since the 1980s. This is a discrete field of cost data whose continuously varying magnitude is represented in a dense spatial array. The cost of each point in robot configuration space, also denoted as *C space*, can be computed as the sum of all cost cells (area integral) occupied by the "footprint" area of the vehicle when it is projected onto the same horizontal plane as the workspace cost map. In the cost map approach, the cost of a path is the line integral of the cost field along the path in configuration space. In such continuous cost fields, *C* space obstacles cannot be defined, because there are no obstacle boundaries.

Difficult environments are usually at least partially unknown, and the topology of navigable regions can be maze-like even on the scale of kilometers. Hence, fielded systems emphasize the use of both a perception system to understand the immediate environment and a deliberative global planning system operating on the kilometer scale. Such planners must replan long paths very regularly in response to incoming perception information. Such perception updates might cause a significant modification of the motion plan. For example, it may be revealed that a predicted ravine leading to the goal is actually a blocked box canyon, or that an unanticipated exit from a dry riverbed has just been discovered.

For the above reasons, fielded systems often use variants of the D* planning algorithm, which was invented to address this core problem in field robotics (Stentz, 1995). In this case, perception causes continuous changes to the cost map that eliminate much of the opportunity to precompute the intersection calculations characteristic of a *C* space obstacle representation. Rather, uniform cost field planners must recompute the cost of each *C* space point every time the underlying workspace cost field changes,

so an explicit computation of C space obstacles is not commonly performed. However, we will see later (Section 3.3) that our approach permits a partial recovery of the advantages of an explicit C space representation.

Difficult environments can also be characterized by dense obstacles through which maneuvering a real vehicle can be a difficult challenge. In cost fields, an obstacle can be identified as a region of high cost. The satisfaction of the differential constraints of a real vehicle adds to the computational complexity of planning. Indoor applications with sparse obstacles may be able to ignore vehicle differential constraints at planning time in favor of smoothing the path at execution time. Another approach is to use a second local planner that satisfies differential constraints but only in the region near the present vehicle location. Our field work has moved into environments for which all of the above smoothing techniques fail utterly because the smoothed path inevitably intersects obstacles, which leads to collision, exceptions, or a basic disagreement in the autonomy system for which we have never found a universal solution. This motivates our premise that effective motion planning in difficult environments requires the satisfaction of differential constraints at planning time.

Our approach in this paper is rooted in the theoretical aspects of the problem, but it also puts a significant emphasis on applications. The approach presented here has been conceived through work with off-road mobile robots, and its development has been driven by failure modes in the field with which we have struggled for some time. The question of efficiency has received considerable attention, even if trade-offs of other motion planning qualities such as completeness and optimality are necessary.

1.2. Related Work

A significant amount of work has been dedicated in recent years to the problem of smooth trajectory generation for differentially constrained vehicles: finding a smooth and feasible path given two end-point configurations. Although in general this is a difficult problem, recent work in this area has produced a variety of fast algorithms. The groundbreaking work in analyzing the paths for differentially constrained vehicles was done by Dubins (1957) and Reeds and Shepp (1990). Their ideas were further developed in algorithms proposed by Scheuer and Laugier (1998), Fraichard and Ahuactzin (2001), and Fraichard and Scheuer (2004), where smoothness of paths was achieved by introducing segments of clothoids (curves whose curvature is a linear function of their length) along with arcs and straight-line segments. Somewhat different approaches by Scheuer and Fraichard (1997) and Lamiraux and Laumond (2001), among others, have also been shown to solve the generation problem successfully and quite efficiently. On the other hand, Frazzoli, Dahleh, and Feron (2001) and Bicchi, Marigo, and Piccoli (2002) suggest that there are many cases in which efficient, obstacle-free paths may be computed analytically. The cases that do not admit closed-form solutions can be approached numerically by solving appropriate optimal control problems (e.g., Anisi, Hamberg, & Hu, 2003; Fernandes, Gurvits, & Li, 1991). A fast differentially constrained trajectory generator by Kelly and Nagy (2002) and Howard and Kelly (2007) generates polynomial spiral trajectories using parametric optimal control.

Some of the methods described above also proposed applications to planning among obstacles. Since the early stages of modern motion planning research (Lozano-Perez, 1983; Lozano-Perez & Wesley, 1979; Reif, 1979), there has been interest in the planning methods that construct boundary representations of configuration space obstacles (Agarwal, Amenta, Aronov, & Sharir, 1996; Agarwal, Aronov, & Sharir, 1999; Canny, Rege, & Reif, 1991, and others). The complexity of motion planning algorithms has also been studied (Alt et al., 1990; Canny, 1988; Jean, 2001; Natarajan, 1988). With the advent of efficient C space sampling methods (Barraquand et al. 1996; Gottschalk, Lin, & Manocha, 1996), there has been interest in algorithms that sample the space in a deterministic fashion (Barraquand & Latombe, 1991; Latombe, 1991). Lacaze, Moscovitz, DeClaris, and Murphy (1998) utilized these ideas to propose a method for planning over rough terrain using generation of motion primitives by integrating the forward model. Cherif (1999) advanced these concepts by basing planning on physical modeling. One novelty of our approach relative to the prior work is the generation of motion primitives that are forced to comply with a convenient state discretization.

Also in the early 1990s, randomized sampling was introduced to motion planning (Barraquand & Latombe, 1990, among others). The probabilistic roadmap (PRM) methods were shown to be well-suited for path planning in C spaces with many degrees of freedom (Hsu, 2000; Kavraki,

1994; Kavraki, Svestka, Latombe, & Overmars, 1996) and with complex constraints, e.g., nonholonomic and kinodynamic (Casal, 2001; Hsu, 2000; Kindel, 2001; Kuffner, 1999). Another type of probabilistic planning was rapidly-exploring random trees (RRT) introduced by LaValle and Kuffner (2001). RRTs were originally developed for handling differential constraints, although they have also been widely applied to the piano mover's problem (Lavalle 2006). Randomized approaches are understood to be incomplete, strictly speaking, but capable of solving many challenging problems quite efficiently (Branicky, LaValle, Olson, & Yang, 2001).

As the randomized planners became increasingly well understood in recent years, it was suggested that their efficiency was not due to randomization itself. LaValle, Branicky, and Lindemann (2004) suggest an intuition that real random number generators always have a degree of determinism. In fact, Branicky et al. (2001) show that quasi-random sampling sequences can accomplish similar or better performance than their random counterparts. The improvements in performance are primarily attributed to the more uniform sampling by quasi-random methods, and hence Lavalle et al. (2004) suggest that a carefully designed low-discrepancy incremental deterministic sequence would be able to do just as well (Lindemann & Lavalle, 2003, 2004). For these reasons, Branicky et al. (2001) introduced quasi-PRM and lattice roadmap (LRM) algorithms that use low-discrepancy Halton/Hammersley sequences and a regular lattice, respectively, for sampling. Both methods were shown to be resolution-complete, although the LRM appeared especially attractive due to its properties of optimal dispersion and near-optimal discrepancy. In this light, our approach of sampling on a regular lattice can be considered to be one of building on the LRM idea and extending it to allow the state lattice to represent the differential constraints of the robot.

Recent works have also discussed "lazy" variants of the roadmap planning methods that avoid collision checking during the roadmap construction phase (e.g., Bohlin, 2001; Bohlin & Kavraki, 2000; Branicky et al., 2001; Sanchez & Latombe, 2001, 2002). In this manner the same roadmap could be used in a variety of settings, at the cost of performing collision checking during the search. An even "lazier" version is suggested, in which "the initial graph is not even explicitly represented" (Branicky et al., 2001). In this regard, the principle of using an implicit lattice and searching it by means of a precomputed control set that captures only local connectivity is similar to the lazy LRM.

In the development of rapidly exploring dense trees for motion planning with differential constraints, the importance of designing offline a family of motion primitives that captures the specifics of the system under consideration is noted (LaValle, 2006). In this light, our proposed control set is precisely the set of primitives that reflects symmetries of wheeled vehicles and encodes differential constraints via offline reachability analysis. Our work is therefore aligned with recent trends in differentially constrained motion planning research, while continuing the study of deterministic sampling methods and the efficiencies of "lazy" exploration of state space.

Initial concepts of this work were validated in a successful field implementation of a differentially constrained motion planner built using an earlier version of the state lattice of limited size represented explicitly (Kelly et al., 2004). In this article, we propose significant improvements in efficiency and generality, while recasting the approach to generate the reachability graph online as it is searched.

1.3. Problem Statement

The basic problem we address here is that of finding a feasible path between two given robot states (e.g., consisting of position, heading, and curvature) for a differentially constrained vehicle in the presence of arbitrary obstacles. If no path exists between the states, the system should indicate failure, and otherwise it must find a path. Our objective is a resolution-complete path planner, so this rule must be satisfied as the sampling resolution approaches infinity. We also desire the solution to be optimal, up to the chosen sampling policy, with respect to an arbitrary but well-behaved notion of a path's cost (e.g., path length, traversal time, and energy expenditure). Further, the planner must be able to handle frequent changes in the environment (e.g., due to noisy and limited perception information). In this setting, the motion plan must be updated frequently enough (e.g., many times per second) to enable efficient operation of the robot. The solution must operate on relevant scales (kilometers of unknown or partially known terrain) while satisfying all of the above constraints.

1.4. Approach

Our approach builds upon a long heritage and many strengths of earlier motion planning techniques. Our aim is to combine the previous results in a novel and unique manner. In particular, our method builds upon the previous introduction of implicit representations of regular search spaces (Branicky et al., 2001; Donald & Xavier, 1995), methods of sampling of relevant state variables of the robot (Barraquand & Latombe, 1991), and recent research on inverse trajectory generation (Frazzoli et al., 2001; Howard & Kelly, 2007; Scheuer and Laugier, 1998). Our approach incorporates efficient replanning by reusing previous computation via the D* search algorithm and its variants (Koenig & Likhachev, 2002; Stentz, 1995).

In general terms, the crux of our proposal is a particular sampling of robot state and control (input) spaces. This sampling subdivides the state space into a set of subregions, *cells*. Each cell is identified with a canonical value of state space that represents the cell. These canonical values are arranged in a regular structure in the state space. A precise trajectory generation algorithm is used that is equipped with a mobility model of the vehicle—including differential constraints and perhaps many other relevant aspects such as steering rate limits, propulsion dynamics, and wheel slip predictions. This trajectory generator is used to compute the controls that precisely connect the states in the above regular arrangement. This setup allows the following:

- generation of a search space (a graph) that satisfies differential constraints by construction (Section 2),
- development of an ideal search heuristic (a precomputed lookup table; Section 3.2),
- introduction of fast and accurate evaluation of cost of motions (replacement for C space expansion with precomputed path swaths; Section 3.3),
- adaptation of existing techniques of efficient replanning (Section 3.4), and
- enhancement of efficiency via a dynamic, multiresolution search space (Section 3.5).

The proposed state lattice reduces the problem of motion planning under differential constraints to unconstrained, replanning heuristic search. It allows offline precomputation of aspects of the problem to enable fast online planning performance.

1.5. Outline

The paper is organized in six sections. In Section 2 we carefully construct the search space upon which our solution is based; we discuss its features and requirements. Section 3 is dedicated to the details of adapting standard search algorithms to this search space. Section 4 provides further details that may help in implementing and evaluating the presented approach. In Section 5, we demonstrate the performance of the planner in simulated and real robot experiments using planetary rover prototype robots. We conclude in Section 6 with closing remarks and future plans for this research.

2. SEARCH SPACE DESIGN

This section presents a progression of design principles that results in the creation of the proposed search space. It satisfies the robot's differential constraints by construction, thereby eliminating the need to consider them explicitly during planning. In this sense, our solution does not consider differential constraints at planning time—a desirable property that was cited earlier. Instead, it considers them offline, during the design of a vehicle-specific search space, which is an even better approach from the perspective of enabling multiple additional efficiencies.

2.1. Regular Lattices

Beneficial state sampling policies include regular lattice sampling, in which a larger volume of the state space is covered with fewer samples, while minimizing the dispersion or discrepancy (LaValle, 2006). It is natural to extend the concept of regular sampling from individual values of state to sequences of states (i.e., paths). As for state space, the function continuum of feasible motions can also be sampled to make computation tractable. The effective lattice state space sampling, developed in this work, induces a related effective sampling of motions.

Suppose that discrete states are arranged in a regular pattern. Besides sampling efficiency benefits, an important advantage of regular sampling of state space is (quantized) translational invariance. Any motion that joins two given states will also join all other pairs of identically arranged states. By extension, the same set of controls emanating from a given state can be applied at every other instance of the repeating unit. Therefore, in this regular lattice

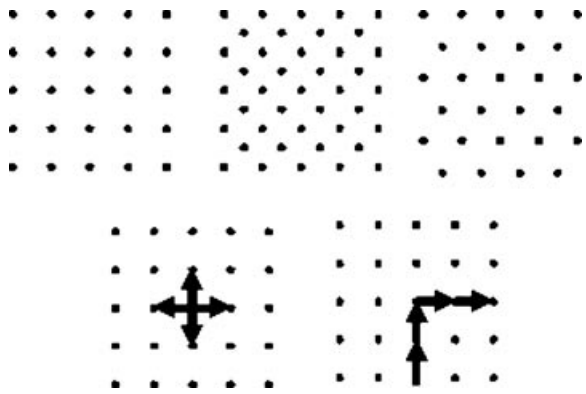


Figure 1. Regular lattices. Top: rectangular, diamond, and triangular (hexagonal) lattices. Bottom left: Discrete motions in a four-connected lattice. Bottom right: Discontinuous heading change in a path.

arrangement, the information encoding the connectivity of the search space (ignoring obstacles) can be precomputed, and it can be stored compactly in terms of a canonical set of repeated primitive motions, the control set.

The simplest case of a regular lattice is a rectangular grid, depicted in Figure 1. The bottom left of the figure illustrates a popular discretization of motions that corresponds to a four-connected grid. As shown in the bottom right, in the illustrated motion discretization, heading change across a vertex in a generated path can be discontinuous. Following this path implies an instantaneous (and therefore impossible) heading change.

We use this observation to develop two properties of lattice search spaces that are necessary conditions for satisfying differential constraints:

1. enforcing continuity of relevant robot state variables across the vertices
2. ensuring that the edges between the vertices of the search space represent feasible motions

The first condition can be satisfied by adding the relevant dimensions to the search space, in order to represent the continuity of state variables explicitly. For example, if a heading dimension is added to the state space in the figure, then (x, y, north) and (x, y, east) become distinct states. Absence of an edge between them makes the above path illegal, and it will not be generated. To satisfy the second condition, we require a method of discretizing the robot control space to force its reachability tree to be a regular lattice in state space. We identify two methods of achieving this:

- *Forward*: For certain systems, there are methods of sampling the control space that result in a state lattice (Bicchi, Marigo, & Piccoli, 2002; Frazzoli et al., 2001).
- *Inverse*: A desired state sampling can be chosen first, and boundary value problem solvers can be used to find the feasible motions (steering functions) that drive the system from one state value to another (e.g., Howard & Kelly, 2007; Kelly & Nagy, 2002).

We prefer the inverse approach because it permits the choice of state discretization to be driven by the application—including the vehicle and the environment. Smaller state spacing is desired for denser obstacles or smaller vehicles. Note that in the state lattice, if state separations are small relative to the distance required to change vehicle heading by the distance to the next heading sample, the edges in such a structure can span many state separations.

Fortunately, the work of constructing the state lattice can be performed offline, without affecting planner run time. Once it is constructed and represented as a directed graph (compactly specified with a control set), the state lattice can be searched with standard algorithms. Two examples of simplified state lattices are shown in Figures 2 and 3.

2.2. Desiderata and Design Approach

Typically, trade-offs exist among the criteria in any design problem. For the sake of generality, no optimal design of the state lattice will be attempted here, because the basis of its optimality would require assuming a particular application. Instead, we describe the aspects of the state lattice design that influence the performance of a planner built using it, with respect to the following important properties:

1. Optimality: how close the cost of an optimal (e.g., shortest) path in the lattice is to the truly optimal path in the continuum
2. Completeness: the degree to which a given search space approaches the capacity to express all available motions
3. Complexity: how much computation is required to solve a particular planning query

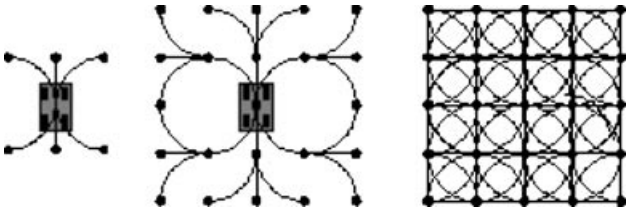


Figure 2. A 3D search space, consisting of position and heading (x, y, θ). The Reeds–Shepp car can move forward and backward. It can drive straight or turn left or right at a fixed curvature. Left: The designed control set precisely hits vertices in a rectangular grid. It was derived from the car's basic motions by carefully choosing their length. Center: The reachability tree to depth 2. Right: The reachability tree (search space) obtained by copying the control set at every vertex in a C space with four headings. Each dot represents four distinct vertices overlaid on each other, each representing different values of heading. Although this search space will not generate a turn of less than the chosen curvature, and although heading is continuous across vertices, the instantaneous transitions of curvature at the vertices do not respect steering rate limitations. Moreover, considering only four different heading values typically is impractical.

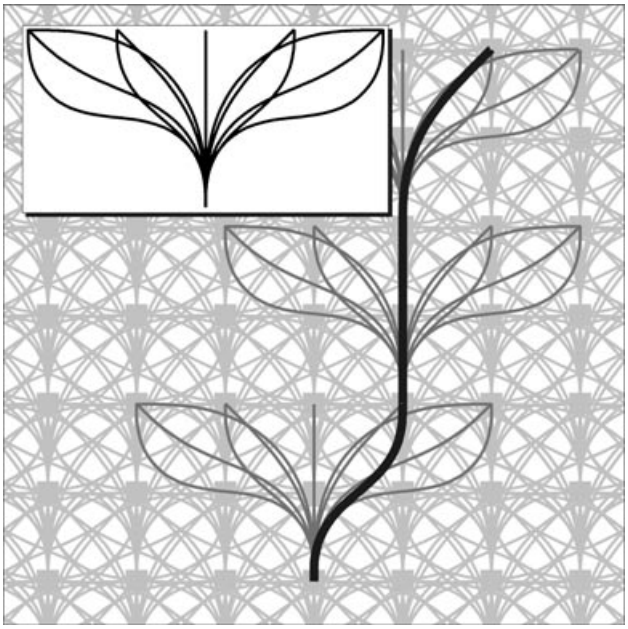


Figure 3. An example state lattice. A repeated and regular pattern of vertices and edges comprises the state lattice. The inset shows the control set, the motions leading to some nearby neighbors of a vertex. The overall motion plan (thick black curve) is simply a sequence of such edges. Here, a greater number of headings was used than in Figure 2. Reverse motions were omitted for clarity.

Enabling a planner to be well positioned with respect to properties 1 and 2 is related to the problem of sampling in the space of motions, and it remains an active area of our related work (Green & Kelly, 2007; Pivtoraiko, Knepper, & Kelly, 2007). One of the benefits of the state lattice approach is that it performs planning strictly in state space, which is easy to sample effectively. Regular lattice sampling features minimum dispersion and discrepancy, which allows the search to proceed effectively. Conversely, achieving effective sampling in control space is hard in general. However, the state lattice induces a convenient sampling in control space, as motions that fit the lattice are found a posteriori. Thus, motion sampling inherits sampling effectiveness from the state lattice. An approach to satisfying properties 1 and 2 in state lattice design is presented in Pivtoraiko and Kelly (2005b). A simplified state lattice design, described in Section 4.2, can also be used as a departing point in evaluating the state lattice concept with a particular motion planner.

The general principle to address property 3 is to reduce the number of motions in the control set as far as possible. In the case of deterministic planning, the size of the control set defines the branching factor of the search space and, thus, significantly affects planning complexity.

3. MOTION PLANNING USING STATE LATTICES

This section is devoted to a discussion of constrained motion planning using state lattices. Here we utilize the search space, developed in the preceding sections, and discuss the algorithmic details of enabling planning and efficient replanning under differential constraints.

3.1. Search Algorithm

Because the state lattice is a directed graph, any systematic graph search algorithm is appropriate for finding a path in it. It is typically desired that a planner return optimal paths with respect to the desired cost criterion (e.g., time, energy, or path length) and that it be efficient. The A* (Hart, Nilsson, & Raphael, 1968) and D* Lite (Koenig & Likhachev, 2002) heuristic search algorithms were used in this work because they satisfy these requirements.

Let the term *fidelity* refer to the resolution of both the state samples and its connecting controls. If, hypothetically, the fidelity of the state lattice were

allowed to increase without limit, it is not hard to see that the state lattice and the corresponding control set, when recomputed at each resolution level, would approach the continuum. Therefore, motion planning based on systematic search of the state lattice is resolution-complete.

Many sequential motion planners can compute the optimal path in the discrete space searched. Hence, as the fidelity of the state lattice is allowed to approach the continuum, it is possible, in principle, to plan paths that approach optimal paths in the continuum arbitrarily closely.

3.2. Heuristic Cost Estimate

Heuristic estimates of the remaining cost in a partial plan are well known to have the potential to focus the search enough to eliminate unnecessary computation while preserving the quality of the solution. In many cases, remaining path length can be used to estimate remaining cost, so path length heuristics are commonly used. Among the simplest options for a heuristic estimate of path length in the state lattice is the Euclidean distance metric. This function is computationally efficient, and it satisfies the admissibility requirement of A* (Pearl, 1984). However, for differentially constrained planning, it is not a well-informed heuristic and, in many cases, it vastly underestimates the true path length, resulting in inefficient search.

A heuristic for a vehicle with limited turning radius moving in the plane could be derived from the methods of Reeds and Shepp (1990). However, the Reeds–Shepp paths are discontinuous in curvature (i.e., infeasible to execute without stopping), and they do not account for discretization, so even these paths are underestimates. To generate the perfect heuristic distance function for the state lattice, it is necessary to incorporate information about the structure of its control set. Given ample offline computational resources, a straightforward and effective way to predict path lengths is to precompute and store the actual cost heuristics that a planner will need, using the planner itself. Such a heuristic lookup table (HLUT) can be implemented as a database of real-valued query costs. Under this approach, the computation of the heuristic becomes a simple table dereference (Knepper & Kelly, 2006; Pivtoraiko & Kelly, 2005a). Note that the regularity of the state lattice is a necessary condition for this heuristic to be admissible. Otherwise, the stored free-space costs of motions are not invariant with respect to (w.r.t.) translation, and they cannot be applied throughout the state lattice.

3.3. Computing Edge Costs

The regularity of the state lattice allows an efficient optimization in evaluation of the cost of graph edges during planning with continuous cost maps, which is roughly equivalent in computational terms to precomputing C space obstacles. Recall that the cost of a configuration is computed as the sum of the workspace cell costs occupied by the vehicle volume [i.e., area in two-dimensional (2D) workspace cost fields]. We denote the set of map cells occupied by the vehicle volume during execution of a particular motion as the *swath* of this motion. Because lattice edges repeat regularly, so do their associated swaths. Thus, it is possible to precompute the swaths for all elements of the control set. When costs change in the workspace cost map, the only computation required to update the cost of an edge (motion) is to add the costs of the cells in the swaths.

The top of Figure 4 depicts a motion of a tractor-trailer vehicle, along with the swath of this motion. To evaluate the cost of a motion, the costs of map cells in the swath (reproduced on the bottom of Figure 4) are simply summed up—an operation typically much more efficient than simulating the motion of the system. The simpler alternative of low-pass filtering the workspace cost map by a circular vehicle

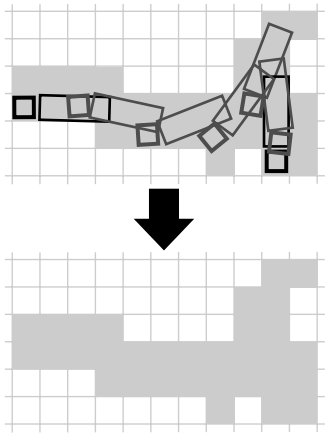


Figure 4. An example of a precomputed swath of a path for a tractor-trailer vehicle. Bottom: The swath allows computing the cost of a motion w.r.t. a cost map, without explicitly considering the motion itself (top).

approximation will be significantly less accurate for systems with elongated shapes.

3.4. Processing Edge Cost Updates in Replanning

The design described so far allows creating a motion planner using A* or any other search algorithm that finds optimal paths in a graph. However, in field robotics applications, efficient replanning (achieved by reusing previous computation) can be a critical design criterion. Because the state lattice is a directed graph, the variants of the D* replanning algorithm can be adapted naturally. Regularity allows D* replanning in the state lattice to be significantly more efficient, thanks to precomputation as described below.

D* variants were originally applied to grids (Koenig & Likhachev, 2002; Stentz, 1995). The fundamental capacity of such planning algorithms is that of efficient repair of a plan given a set of workspace cell cost changes that have been created by perception. In general, the workspace cost field need not have the same resolution as the planning search space, and the connectivity of the latter can be arbitrary. The earliest work on D* used the same resolution for both the cost map and the search space and implicit "edges" that connected states only to their nearest eight neighbors. In this case, the mapping from a modified map cell to the affected search space edges and vertices is trivial.

More generally, workspace cell cost changes induce changes in perhaps several search space edge costs that, in turn, induce changes in search space vertex (edge endpoint) costs. For a state lattice whose edges may span several map cells, the above historical simplifications of these issues are no longer feasible.

Suppose that the replanner uses a priority queue to ensure optimality of the solution. For every change in the cost of the directed edge from the vertex x_i to x_j, $c(x_i, x_j)$, a replanning algorithm requires recomputing the cost of x_j and potentially inserting it into the priority queue. Assuming that a map cell $m_{ij} \in \mathbb{N}^2$ changes cost, the planner needs to know the set of vertices V_c that potentially need to be reinserted into the priority queue with new priority. Thus, the planner requires a mapping $Y : \mathbb{N}^2 \rightarrow V_c$.

To develop this mapping, we use the concept of swath, introduced in Section 3.3. More formally, we consider the swath a set $C_s \subset \mathbb{N}^2$ of cost map cells that are occupied by the robot as it executes a motion. The cost of an edge that represents this motion is directly dependent on the costs of map cells in C_s. Recall that once we precompute the control set of a regular lattice, it is possible to precompute the swaths of the edges in it.

Because the mapping between edges and their terminal vertices is trivial, it is easier first to develop the mapping $Y' : \mathbb{N}^2 \rightarrow E_c$, where E_c is the set of edges that are affected by m_{ij} (i.e., the set of edges whose swaths pass through the cell). Determining Y' may still appear as a formidable task, given the high density of edges in the multidimensional state lattice. However, we again exploit the regularity of the lattice to simplify the problem. If we have $Y'' : O \rightarrow E_c$, where O is the map origin, then $Y' = Y'' + n, \forall n \in \mathbb{N}^2$. In other words, the set of edges affected by $m_{ij} = O$ is identical for any other cell, up to the translation coordinates. Further, recall that the swath C_s of each edge in E_c is known. In principle, E_c contains all edges u_c, such that m_{ij} belongs to C_s of u_c. Hence, the mapping Y'' is exactly the set of edges whose swaths pass through the 2D origin. Figure 5 illustrates this idea. Like the control set and path swaths, the resulting set of edges can be precomputed due to the regularity of the state lattice. An example of the V_c for the implementation described in Section 5 is shown in Figure 6.

3.5. Graduated Fidelity of Representation

By virtue of the state lattice's general representation as a directed graph, it can be naturally extended with multiresolution enhancements. Significant planning run time improvement was achieved in the literature via a judicious use of the quality of representation of the planning problem (e.g., Bohlin, 2001; Ferguson & Stentz, 2006; Pai & Reissell, 1998, among others). In field robotics, it is frequently beneficial to utilize a high fidelity of representation in the immediate vicinity of the robot (perhaps within its sensor range) and reduce the fidelity in the areas that are either less known or less relevant for the planning problem. Lower fidelity of representation is designed to increase search speed, whereas higher fidelity provides better quality solutions. Because grids have traditionally been utilized in replanning, the notion of varying the quality of problem representation has been identified with varying the resolution of the grid. However, our method varies the discretization of both the state and motions. We refer to managing the fidelity of state lattice representation as *graduated fidelity*.

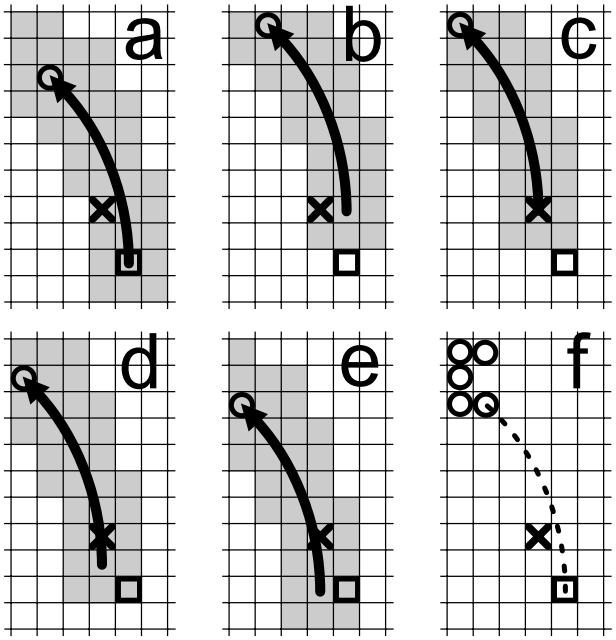


Figure 5. The first several steps of precomputing the list of graph vertices that are affected by a change in cost of a map cell. (a) A single element of a control set is chosen for this example. It emanates from the origin of the state lattice (thick square) and connects it to another graph vertex (thick circle). Gray cells are the swath of this motion. Suppose that a map cell, located at the origin of the state lattice (thick square), changes cost. We attempt to find all translational versions of the chosen motion whose swaths are affected by the changed map cell. (b)–(e) We iterate through several such translational versions of the motion. The resulting (edge end-point) vertices that are considered for insertion to the priority queue are shown in (f). Typically, many more such vertices are processed for each edge [as suggested by ellipsis in (f)]. The process repeats for all edges in the control set. Precomputation allows eliminating any redundancy by generating a unique list of such vertices.

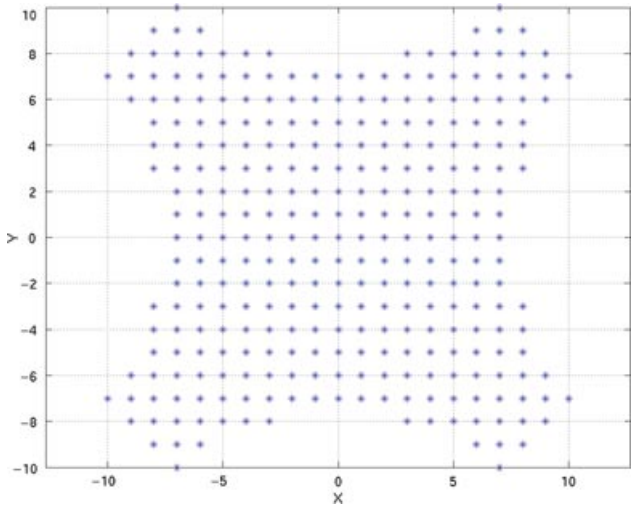


Figure 6. A 2D projection of an example of V_c, the set of lattice states that are to be reconsidered for every updated map cell. The units in the plot are state lattice cells. For the purposes of exposition, here the size of map cells is set to be equal to the size of state lattice (x, y) cells. For each map cell, m_{ij}, that changes cost, we place the set of vertices above in this figure onto m_{ij} [i.e., the origin of the set of vertices, denoted with coordinates $(0, 0)$, is identified with m_{ij}]. Next, we iterate through the depicted list of the vertices and place each one on the priority queue, if it was indeed affected by the cost change of m_{ij}.

In designing the connectivity of search space regions of different fidelities, care must be taken to ensure that all regions consist of motions that are feasible with respect to the robot's mobility model. If this rule is violated, mission failures become possible due to the differences in the representation of vehicle mobility. Figure 7 illustrates this situation using a simple example. Suppose that a search space is used in which a high-fidelity region of finite size surrounds and moves with the vehicle, and a disjoint lower fidelity grid is used beyond that. Suppose that the A* algorithm is used to plan paths in this hybrid graph. A car-like robot attempts to travel to a goal on the other side of a collection of obstacles that forms a narrow corridor. As long as the low-fidelity region includes the corridor (black line), the planner will find a solution in the graph. However, the 90-deg turn in the path is actually infeasible, because the car-like robot cannot turn in place. As the vehicle moves, the high-fidelity region will eventually include the turn in the corridor and the planner will then fail to find a solution. The only viable alternative will be to back up, thereby moving the corridor to the low-fidelity region once again. Because the original

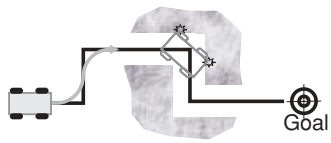


Figure 7. A simple example of a motion planning problem, in which a car-like robot that attempts to follow the infeasible path (black line) experiences a failure.

state of the scenario has now been achieved, it is easy to see that this behavior will repeat forever. To avoid such difficulties, it is necessary to ascertain that all levels of fidelity include feasible motions. In particular, the connectivity of low-fidelity regions must be a subset of that of the higher fidelity regions.

To implement graduated fidelity planning, the above A* planner-based design requires only a minor modification. Once the state lattice graph is separated into subgraphs of different fidelities as desired, each subgraph uses its own control set to achieve the chosen fidelity. Each control set defines the successors of a vertex being expanded during search. Care must be taken to design the control sets such that they adequately span the boundaries between the subgraphs. Note that control set design is the sole procedure needed to enable graduated fidelity. Replanning algorithms require no changes and will achieve the desired effects automatically.

It can be useful to enable a high-fidelity subgraph to move along with the mobile robot as described in the example above. As shown in Pivtoraiko and Kelly (2008), such flexibility can be accomplished by undoing the effects of previous expansions of the vertices on the perimeter of the moving subgraph. Accomplishing this once again requires no change to the actual replanning algorithm. The change of graph connectivity that occurs between replans is presented to the planning algorithm as a change in cost of the affected graph vertices. Such topology-based cost changes appear to replanning algorithms to be identical in nature to perception-based cost changes. If the vertex expansion step is considered to be part of an external search space module, the planner actually cannot tell that the graph topology is changing.

More generally, it is straightforward to extend the concept of graduated fidelity to allow multiple subgraphs of different fidelity to move or change shape between replans. Such flexibility results in a *dynamic search space*, which complements dynamic replanning algorithms to improve planning efficiency. Thus, the graduated fidelity extension of state lattice planning is conceptually simple and straightforward to implement, and it can be designed to result in significant savings in run time and memory usage in replanning.

HLUT that stores vehicle-specific optimal path costs can be very effective in the case of graduated fidelity, because D* is typically configured to plan backward from the goal to the robot. The distance in the search space from topology-induced cost updates (typically perimeter of fidelity subgraph around the robot) to the robot is therefore limited enough to make it possible to store a HLUT for even multidimensional state spaces on contemporary computers. In this case, the perception and topology updates generated during operation can be propagated toward the vehicle with the efficiency of an ideally informed heuristic.

4. IMPLEMENTATION DETAILS

4.1. Sampling the Heading Dimension

If the position variables of the state lattice are sampled as a square grid, and there are more than eight samples of the heading dimension, regular sampling in heading leads to incapacity to generate straight paths in any direction other than the cardinal and ordinal ones. An irregular sampling of heading is a better solution from the perspective of encoding more straight-line paths. Generally, a line at orientation θ will intersect a vertex if it satisfies $\theta = \arctan(i, j)$ for any two integers i and j. Hence, an irregular sampling of heading is preferred for a lattice with this arrangement.

4.2. Control Sets with Shortest Edges

Algorithm 1 is a simple inverse method for generating a control set, as introduced in Section 2.1. Referred to as the shortest edges algorithm, it may serve as a departing point to evaluate our proposed approach to search space design. To better illustrate the algorithm, in this section we assume a four-dimensional (4D) state lattice that consists of 2D translation, heading, and curvature. Suppose that Θ and K are user-defined subsets of discrete values of

Algorithm 1. A simple method of generating a control set

Input: State discretization in the state lattice: position, discrete values of heading (Θ) and curvature (K)
Output: A control set, E_x
$E_x = \emptyset$;
foreach $\theta_i, \theta_j \in \Theta$ *and* $\kappa_i, \kappa_j \in K$ **do**
  foreach x_f, y_f *s.t.* $L_\infty(O, [x_f, y_f]) = [1 \cdots \infty)$ **do**
    $u_i = \text{trajectory}([0, 0, \theta_i, \kappa_i], [x_f, y_f, \theta_f, \kappa_f])$;
    if $u_i \neq \emptyset$ **then**
      $E_x \leftarrow u_i$;
      break;
    end
  end
end

heading and curvature in the state lattice, respectively. By exploiting rotational symmetries in the state lattice, these sets can be convenient strict subsets of all possible discrete values of these states variables. The outer for-loop selects the permutations of discrete values of initial and final heading and curvature. The inner for-loop cycles through all discrete value pairs of x and y, such that the maximum norm[1] L_∞ between the origin O and (x_f, y_f) grows from 1 to infinity. For each value of L_∞, if the trajectory generator finds a solution to the boundary value problem, a feasible trajectory u_i, we add it to the control set. At this point we break from the inner for-loop and proceed with another choice of terminal heading and curvature values. The algorithm terminates when a trajectory is generated for every permutation of heading and curvature values.

The sets of motion primitives generated by Algorithm 1 have a number of attractive properties:

- this is the minimal set that contains primitives that concatenate at all discrete values of heading and curvature in the state lattice
- the primitives in the set are minimum length, given terminal constraints and discretization, so they encode locally optimal solutions
- they encode aggressive maneuvers that will result in plans that exploit the entire maneuverability envelope of the robot
- the computed controls exhibit some workspace separation by construction, thereby reducing dispersion in motion sampling

4.3. Application Specific Planner Configurations

In very difficult environments, feedback control may not be an adequate solution to compensating for unmodeled terrain following or disturbances such as well slip. On the contemporary Mars rovers, for example, wheel slip is high enough to cause complete mobility failure and only infrequent visual odometry updates are available to observe the slip that occurs. If the 3D terrain shape is known sufficiently well at planning time, a terrain-aware version of our trajectory generator (Howard & Kelly, 2007) can be used to generate appropriate edges that fit the terrain spanned during the search. This is more computationally expensive than assuming flat terrain and using the precomputed, terrain-independent control set. However, this is an important extension that allows the planner to consider rough 3D terrain explicitly. An edge that was created for flat terrain may become infeasible in a particular location in rough terrain. However, this case is no different from the appearance of an unknown obstacle, and a replanning strategy would presumably already be in place to handle such events.

Additional state variables can be added to the state lattice. In particular, the addition of velocity can enable a planner to compute feasible robot motions at various speeds that satisfy vehicle propulsion dynamics.

5. RESULTS

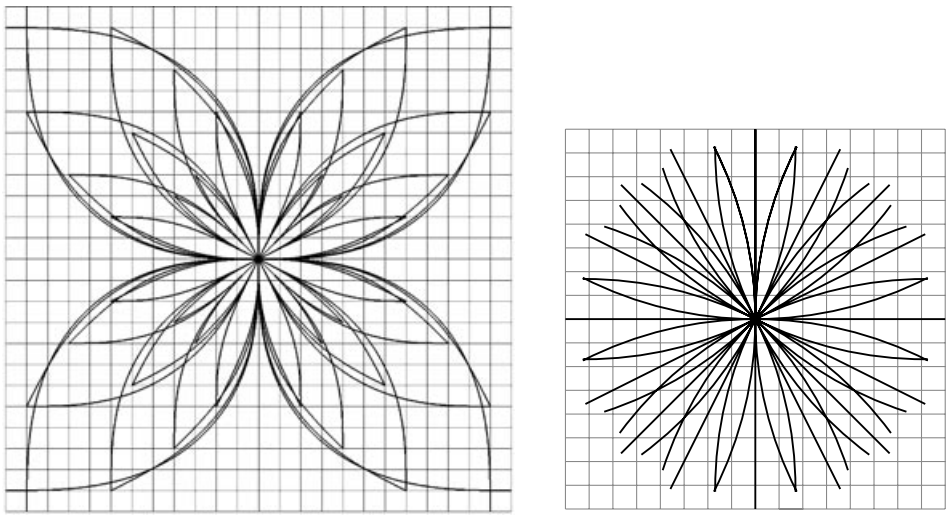
A differentially constrained motion planner was implemented based on the state lattice and tested in a variety of scenarios, including in simulation and on real robots. In the sequel, we will refer to this implementation as the *lattice planner*. In this section, we offer the quantitative results of our evaluation.

A representative lattice control set was used in all tests. Its state space consisted of 2D position and heading (x, y, θ). This control set, depicted in Figure 8(a), was generated using the shortest edges algorithm (Section 4.2). A trajectory generator in Kelly and Nagy (2002) was used to generate the motions between the given values of robot state. Motions were parameterized as cubic polynomial curvature, κ, functions of path length s, as shown in Eq. (1):

$$\kappa(s) = a + bs + cs^2 + ds^3. \tag{1}$$

Subsection 5.1 is devoted to comparing the lattice planner to popular motion planning approaches. This subsection focuses on single static invocation of all considered planners; replanning and graduated fidelity were not used, because not all evaluated planners support such capabilities. The goal of fair comparison also influenced the chosen configuration of the lattice planner. A 4D state lattice that considers curvature in addition to position and heading was also implemented and successfully validated. It allows explicit goal specification in all four dimensions but none of the other evaluated planners support such capability. Subsection 5.2 presents the results of testing state lattice replanning and graduated fidelity in a realistic setting.

[1] L_∞ norm of a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is $\max_i |x_i|$.



(a) (b)

Figure 8. Differentially constrained control sets used in experiments. (a) The state lattice control set selected for testing has 16 discrete headings, a minimum turning radius of 8 cells (i.e., maximum curvature 1/8), and an average out degree of 12, for a total of 192 controls. The straight-line controls cannot be seen because they are obscured by the longer curved ones. (b) The chosen BL control set consists of 96 controls, each of length 4.

5.1. Comparison of the State Lattice to Other Search Spaces

In this section, we discuss a comparison of the state lattice to several leading search spaces in motion planning. The state lattice is compared to the following:

- 2D grid (4-, 8-, and 16-connected grids)
- search space used in the well-known Barraquand and Latombe (BL) differentially constrained planner (Barraquand & Latombe, 1991) [Figure 8(b)]

We have adopted a notion of a generalized control set: for the 2D grid, it is simply a grid-based vertex expansion (straight paths), whereas for the BL planner, it is the vertex expansion as described in that work. A planner based on each search space was implemented by running the identical implementation of the A* search algorithm using the corresponding generalized control set. An overview of the control sets that are considered here is presented in Table I.

5.1.1. Experimental Setup

In the presented experiments, we generated a list of 10,000 random queries, each consisting of an initial and final state, expressed as position and heading. The set of queries was engineered to induce the planner to generate paths ranging from simple (nearly straight paths) to complex (e.g., parallel parking or n-point turn maneuvers).

In the case of a grid, Euclidean distance between start and goal is an appropriate estimate of free-space planning difficulty. As argued in Section 3.2, it can be an unacceptable underestimate of the true planning difficulty under differential constraints. To quantify the complexity of a particular query, we distinguish between *absolute* and *relative* difficulty. Figure 9 illustrates the two concepts. Absolute difficulty is

Table I. A quantitative look at the control sets. Parameters of each control set have a strong influence on how a planner performs while using it. (Units are 2D grid cells.)

Control set	Total edges	Avg. edge length	Avg. out degree
Lattice control set	192	8.72	12
Lattice with turn in place	224	7.47	14
BL	96	4.00	6
Grid-4	4	1.00	4
Grid-8	8	1.21	8
Grid-16	16	1.72	16

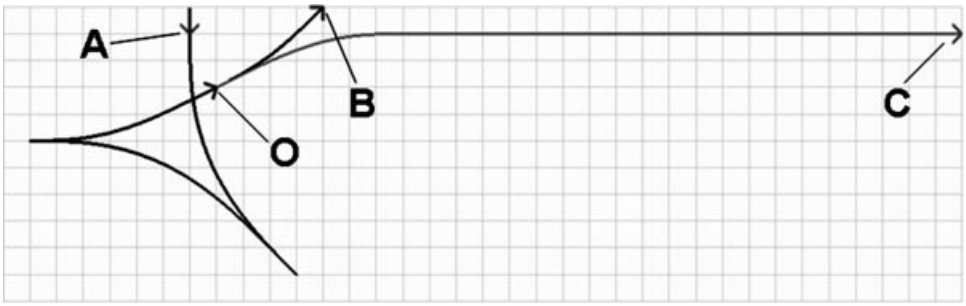


Figure 9. Absolute and relative planning difficulty. The difficulty of a planning query can be quantified in two dimensions. Each of three paths to endpoints A, B, and C starts at O. Query A is high in absolute difficulty as well as relative difficulty because it is long and has multiple cusps. B is simple in both measures. Query C has the same absolute difficulty as A (same length) but the same relative difficulty as B.

measured here by the path length. For relative difficulty, the ratio of Euclidean distance (between start and goal) to the length of a differentially constrained path is used. With this scale, a value of unity means that the resulting path is a straight line, whereas values near zero indicate that a series of reversal maneuvers is necessary to reach the final pose.

In the analyzes below, start and goal states were sampled randomly within a rectilinear grid whose resolution matched that of the state lattice. The goal poses were sampled relative to the start state using a polar coordinate system. We took this approach in order to restrict the length of queries presented to the planner to a desired range from 0 to 10 turning radii (80 cells). For this study, we chose queries with absolute difficulty of 40 cells. This number was chosen arbitrarily for clarity of presentation. Experimentation showed that other values of absolute difficulty convey similar results. Initial and final headings were randomly selected.

Each query was tested using each generalized control set in a variety of settings, in particular with different choices:

- obstacle fields
  - free space (no obstacles)
  - single map-cell obstacles, uniformly distributed with 5% density in the plane (Figure 10)
- heuristic functions
  - zero heuristic (constant value of 0)
  - Euclidean distance
  - HLUT

Free space experiments were conducted in order to review planner performance independent of any

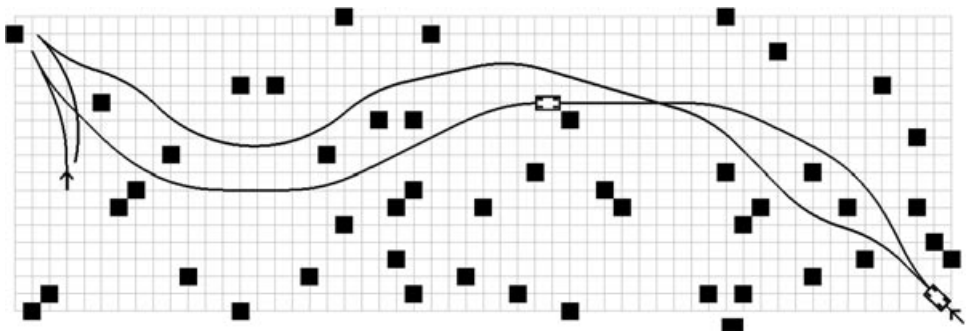


Figure 10. World with obstacles. A portion of the world with point obstacles used in the experiments is shown here. Two plans are shown that solve the same query. The black line shows the plan generated by the state lattice, and the gray line traces the path returned by the BL planner.

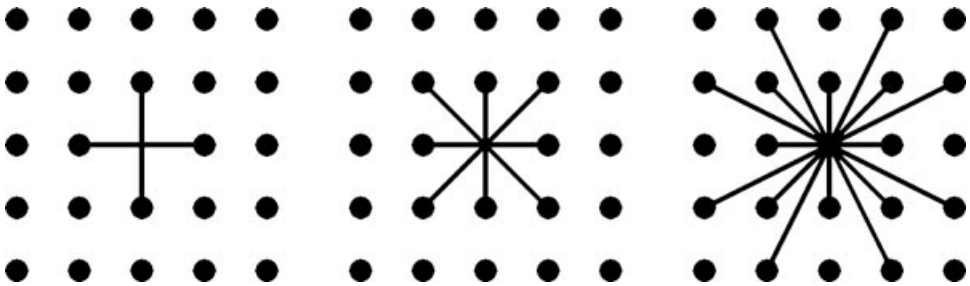


Figure 11. Grid control sets. Three different grid control sets were tested. A point is connected to the 4, 8, or 16 nearest neighbors that have unique headings.

particular obstacle arrangement. Path cost was set to distance traveled; cost to traverse free space was held constant at 1 unit per cell of free space. The dependent (observed) variable in this study is planner run time, which we verified to be proportional to the omitted second dependent variable, memory usage.

5.1.2. Comparison to a Grid Search Space

2D grid search spaces comprise possibly the simplest and oldest mobile robot planning context. It may seem inappropriate to compare lattice and grid planners because a grid does not permit us to enforce full state continuity across graph vertices. However, part of our aim in comparing the two is to make the case that the computational cost of enforcing differential constraints at planning time is not as high as might be assumed by many readers. Grid-based paths do not ensure feasibility of execution by mobility-constrained vehicles. Nevertheless, grid planners have often been employed to find approximate solutions for differentially constrained vehicles despite their incapacity to follow the solution path exactly. In such applications, a path tracker is typically employed to smooth out the corners and follow the path inexactly. Indeed, it was our implementation of this approximate mobility approach that the lattice planner was developed to replace.

An important question is, "Is the grid approach inherently faster than searching in a full-dimensional state space?" Differential constraints actually reduce the number of paths encoded, whereas the extra dimensions of state space increase the number. The answer is not immediately obvious. To find it, we ran the A* planner with four different control sets, in each case using the appropriate heuristic function that returns the exact distance to the goal. The basic state lattice was matched with a large HLUT. Three grid control sets were tested, in which each state is connected to its 4, 8, and 16 nearest neighbors (Figure 11), using a perfect heuristic for each level of connectivity.

Performance of the control sets in the absence of obstacles is shown in Figure 12. The data are presented across a spectrum of relative difficulty. In the absence of obstacles, the lattice generally performs on par with basic grid search. For only the most difficult queries does the lattice consume more CPU time than a grid control set. This disparity occurs because the differentially constrained path solution diverges more dramatically from a straight-line solution. For such problems, the answer returned by the grid is increasingly difficult to execute on a real vehicle, even if a path tracker is used. In essence, the extra effort on the part of the lattice planner is compulsory to obtain a feasible path. Hence, for our test setup, in the absence of obstacles, a state lattice is just as efficient as a grid on easy problems and an order of magnitude slower on hard problems.

Results in the presence of obstacles are shown in Figure 13. It is intuitive that the grid should outperform the lattice in this obstacle field for any class of query. If an obstacle appears in the path of a grid plan, that plan is often displaced by only a few cells in order to circumvent the obstacle. In the case of the lattice under test, however, only smooth continuous paths are considered. These requirements substantially limit the planner's options, making it more difficult to find a satisfactory path through an obstacle field, as shown in Figure 14. So while the grid path deviates slightly in the cluttered environment, paths generated by the state lattice often must be much more complex in order to plan around obstacles. However, the difficulty experienced by the

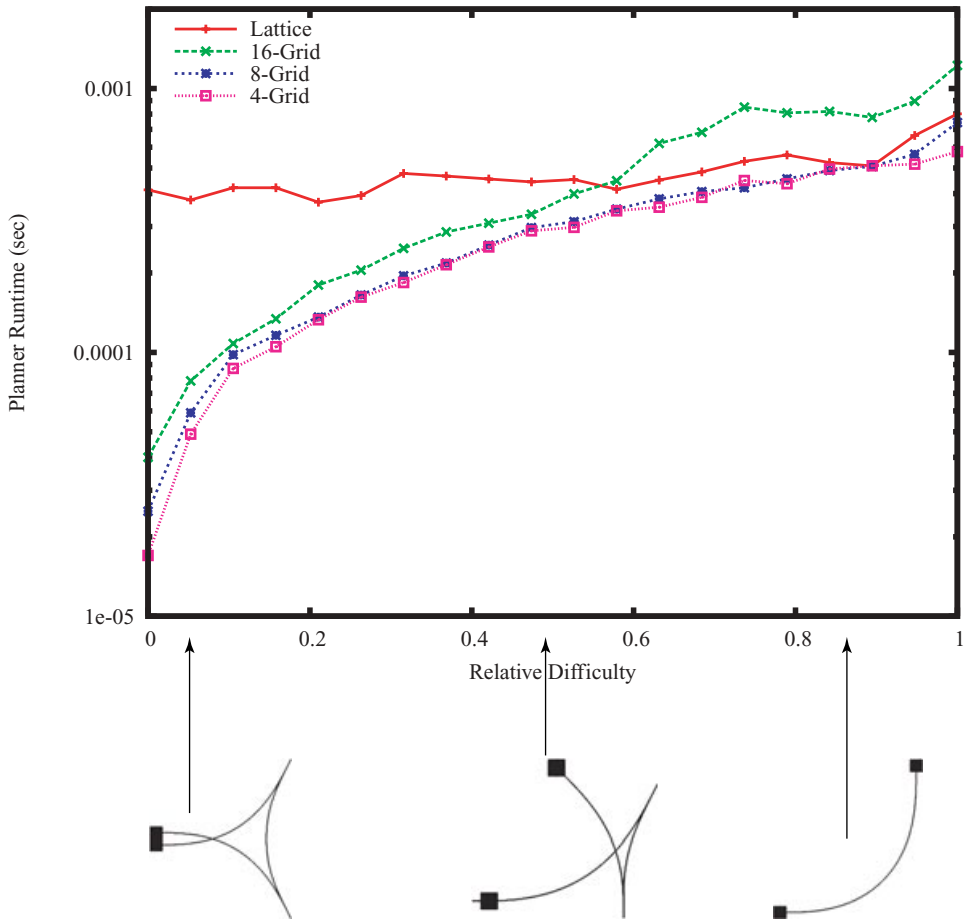


Figure 12. Lattice vs. grid without obstacles. Queries with an absolute difficulty of 40 cells are shown. The lattice performs similarly to the grid planners overall. Only for the greatest relative difficulty (nearest zero) does the lattice require more CPU cycles. Of course, the benefit of this extra computation is a plan that is feasible.

lattice planner reflects the true mobility limits of the vehicle, rather than some inadequacy relative to a grid search space.

In plain terms, a grid planner produces faster answers, but they are usually wrong. Despite the increased search requirements, the lattice planner remains consistently only one order of magnitude slower than the grid planner, returning on average in less than 0.1 s for all classes of query.

5.1.3. Comparison to Full C Space and Differentially Constrained Search Space

The above comparison between the state lattice planner and grid-based planner is intended to evaluate the search spaces, not the identical planning algorithms. Of course, paths planned in the grid cannot be traversed by curvature-constrained vehicles without postprocessing. However, the BL planner is well known and has been a popular differentially constrained planner for more than a decade. It has also been used in real mobile robot applications (Morris, Silver, Ferguson, & Thayer, 2005). As described by the authors, the BL planner runs A* with a zero heuristic, resulting in a breadth-first graph traversal. Each graph edge has equal cost and represents one of six possible controls combining forward/reverse with hard left/straight/hard right. To limit the exponential growth in reachable states, BL treats the least-costly method of reaching a given discrete state as the canonical route. Costlier paths terminating in the same C space cell are discarded.

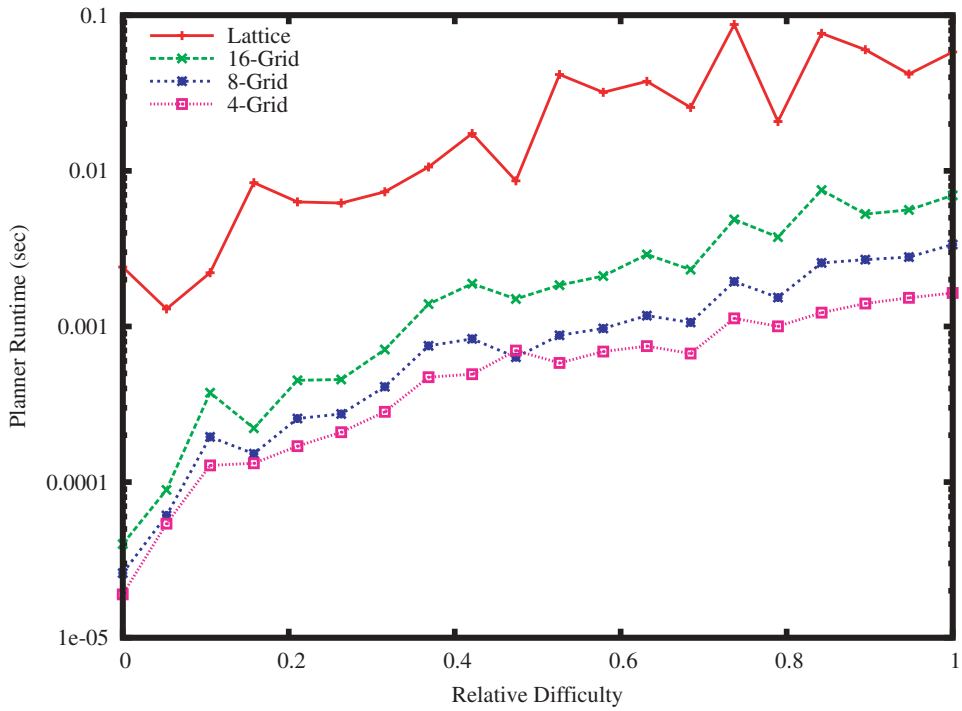


Figure 13. Lattice vs. grid with 5% obstacle density. Queries of length 40 cells are shown. The grid outperforms the lattice, but it usually returns infeasible solutions. Lattice planner run time is still acceptably fast.

On careful examination, it is clear that a comparison to the search space of the BL planner is also not as clearly meaningful as one would hope, because the state lattice supports much more sophisticated planning algorithms (in particular, D* replanning). Specifically, the original BL planner search space uses the number of reversals of velocity as edge cost. For example, as configured in the original paper, the BL planner may choose a shorter path through a high-cost area instead of a longer path through a lower cost area. To render this algorithm more suitable for field applications, the breadth-first algorithm must be replaced with one that is optimal for graphs with variable cost of edges embedded in continuous cost fields—such as A* implemented with a priority queue. A heuristic to guide the search was also necessary in our experiments to achieve acceptable run times.

In this section, several different variants of the BL planner are examined. In all cases, an identical

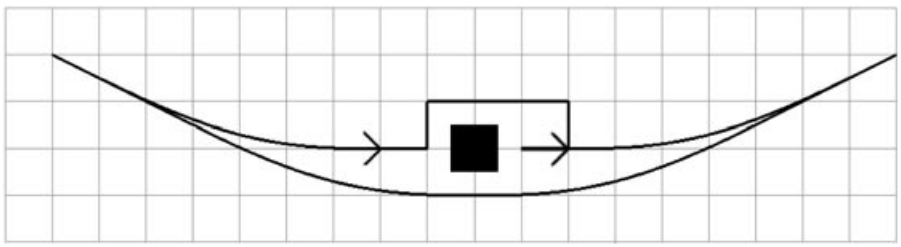


Figure 14. Obstacle avoidance with two control sets. Grid planners can easily avoid small isolated obstacles, as shown by the black path. By contrast, the gray path has limited curvature and so can be feasibly traversed by a constrained vehicle.

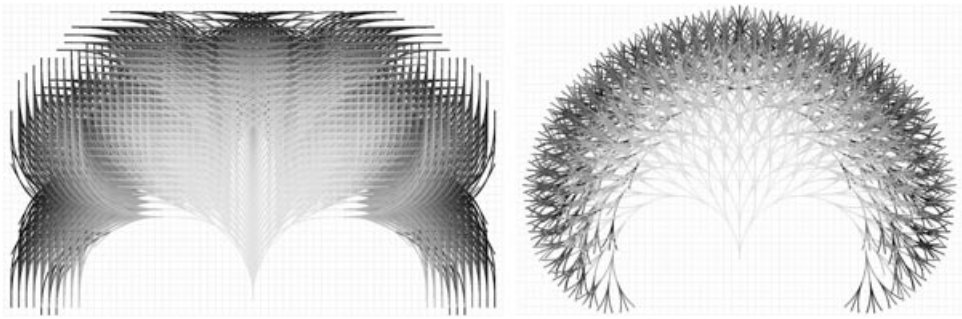


Figure 15. Reachability trees for the state lattice and BL. At left, the first 1,400 expansions of the basic state lattice control set in best-first order. At right, the first 1,400 expansions of the BL control set were generated using the same algorithm. Darker edges indicate greater depth in the tree.

A* algorithm was applied, but the heuristic cost estimate was altered to produce a fair comparison. In the first case, the algorithm was run as documented in Barraquand and Latombe (1991), with a zero heuristic applied. In the second case, Euclidean distance was used as the heuristic.

To make the BL control set work in a fashion compatible with the state lattice test framework, some tuning was necessary. Barraquand and Latombe describe the edge lengths as being the L_1 diameter of a cell and assert that their planner requires the discretization of the search parameters to be "fine enough." We set the BL curve length to four diameters of our cost map cells because a C space cell spans four discretized unit dimensions in our cost map. To ensure fairness, several other curve lengths were tried, but the original length of four was found to be the best match with our standard lattice discretization. This basic BL control set is shown in Figure 8(b), and the resulting reachability tree is shown in Figure 15 along with the standard state lattice tree. Reverse edges were omitted from both trees in the figure for clarity.

The performance of the lattice planner is compared to BL with two different heuristics in Figure 16 (no obstacles, free space) and in Figure 17 (with obstacles, per Section 5.1.1). In a fair matchup using the Euclidean distance heuristic, the two planners perform comparably. The Euclidean heuristic was used for the lattice-based planner only for fairness of the comparison. The lattice planner using the HLUT significantly outperforms our implementation of BL.

5.1.4. State Lattice with Turn-in-Place Motions

Finally, we tested an alternative lattice control set. It is identical to the baseline lattice control set, except that it includes two extra controls—each representing an incremental turn in place left and right (change of heading to the two nearest discrete values). We computed the cost of these additional edges based on the aggregate distance of travel of each of the wheels of a planetary rover, capable of turning in place. The effect of this capability on resulting plans is depicted in Figure 18, where the path is substantially shortened by avoiding tortuous maneuvering. These additional controls have no significant effect on overall planner run time performance (Figures 19 and 20), but they provide valuable added flexibility in negotiating dense obstacle fields. Moreover, for practical applications, the state lattice allows tuning the robot's preference for performing point turns versus smooth maneuvers by assigning appropriate costs to the turn-in-place edges.

5.2. Autonomous Navigation

Here we present the results of experiments, simulated and real, that demonstrate planner performance using replanning and graduated fidelity in a realistic setting, where the robot moves through a challenging environment toward a distant target.

The rover size is approximately 1×0.8 m. Its mobility is characterized by a minimum turning radius of 0.5 m and a capacity of point turns, which bear a high cost due to the time and energy required for

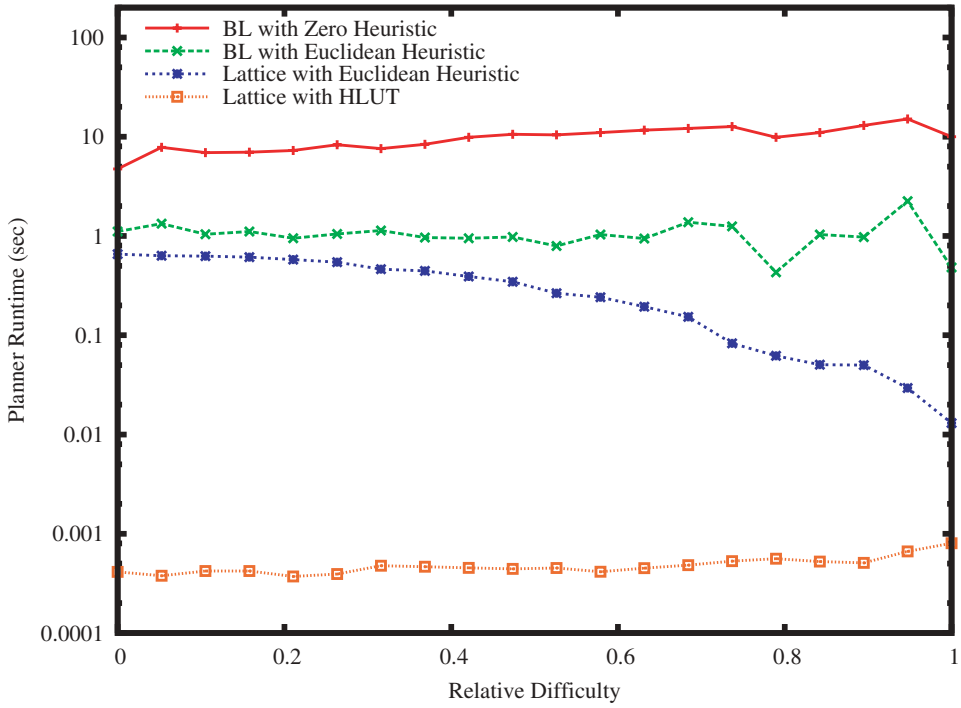


Figure 16. Lattice vs. BL without obstacles. Queries with an absolute difficulty of 40 cells are shown. Various heuristics are considered for each planner.

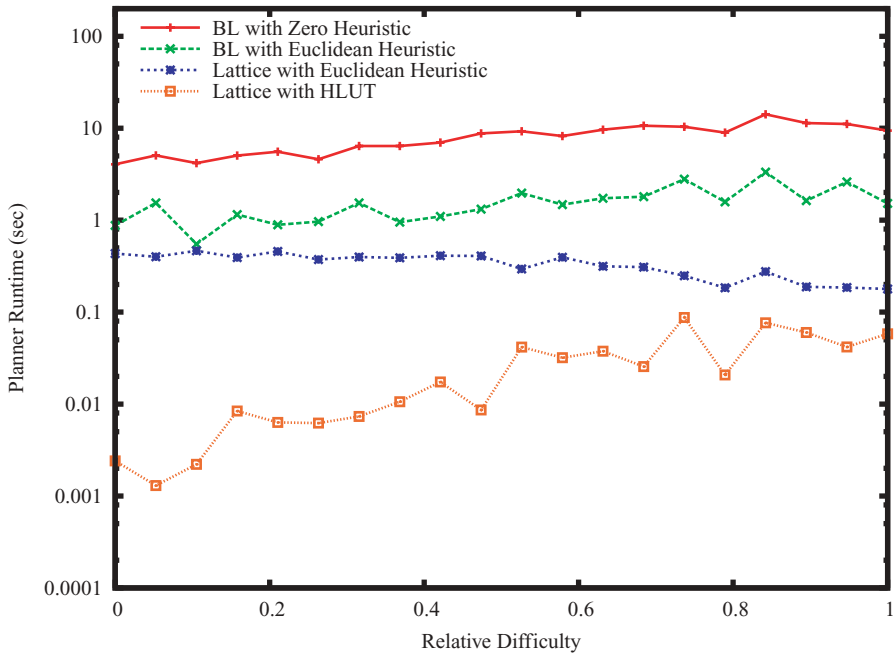


Figure 17. Lattice vs. BL in a world with uniformly distributed single map-cell obstacles of density 5%. Queries with an absolute difficulty of 40 cells are shown. Various heuristics are considered for each planner.

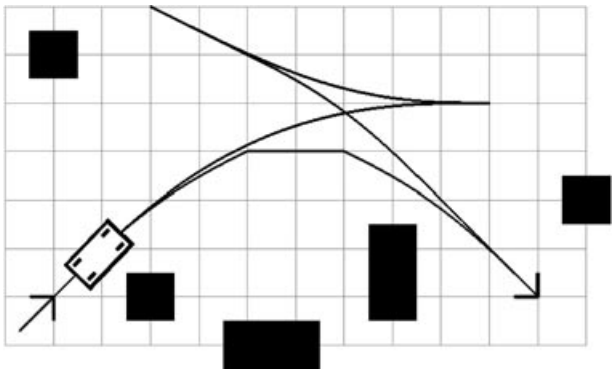


Figure 18. Lattice control set with and without the ability to turn in place. In black, the aforementioned state lattice control set is used. In gray, the same control set is augmented by two additional controls that allow the vehicle to turn in place at a cost equal to traversing five cells. This plan performs the turn-in-place maneuver twice in order to execute a sharp turn.

reorienting wheels. Both cost map cells and (x, y) cells of the state lattice are square with 20-cm side length; both types of cells coincide in position.

In the simulated experiment, the robot has a perception region limited to 21×21 cells (L_∞-radius of 2 m), centered around the robot. No perception information is available outside this horizon. The size of the high-fidelity region, centered about the robot, is the same as that of the perception region. Otherwise, the setup is the same as above. For clarity, Figure 21 shows a 40-m subset of a 500-m path, traversed in this setting. Gray cells are obstacles that have not yet come into view of the robot and are unknown to it. Black cells (and gray cells in the insets) are obstacles that were seen by the robot. The dark-gray line is the path the robot traveled. Note that it entered many cul-de-sacs due to the limited perception (such as replan cycles 39 and 53), and the planner was effective at guiding the robot out of all of them by exploiting the robot's maneuverability. Replanning occurred continuously due to both obstacle discovery and fidelity modification in the search space. This experiment was performed on a conventional laptop computer with 2-GHz CPU and 2 GB of RAM.

The lattice planner was integrated with research prototype rovers at the NASA/Caltech Jet Propulsion Laboratory (JPL). It enabled the rovers to navigate

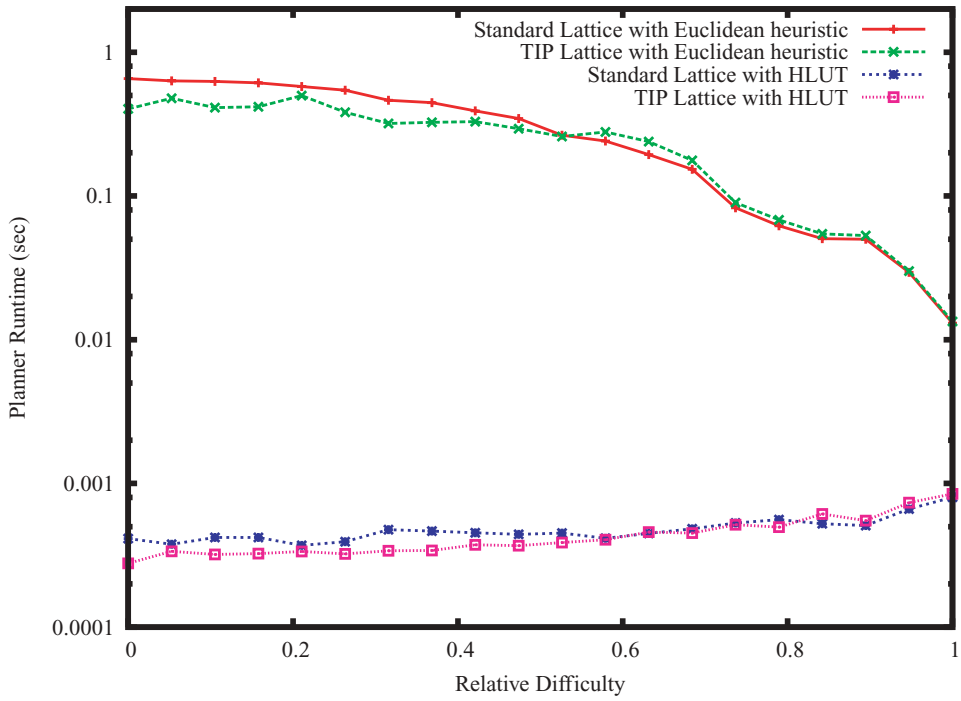


Figure 19. Lattice vs. turn-in-place lattice without obstacles. Queries with an absolute difficulty of 40 cells are shown. Computational cost for the two lattices is not significantly different despite the extra turn-in-place maneuvers.

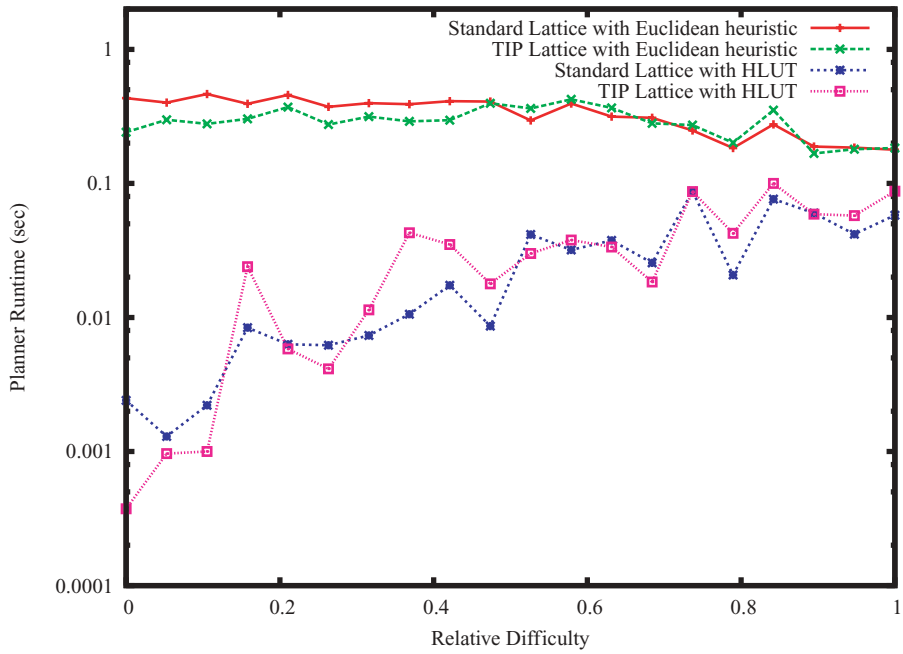


Figure 20. Lattice vs. turn-in-place lattice in a world with obstacles. Queries with an absolute difficulty of 40 cells are shown. Computational cost for the two lattices is not significantly different despite the extra turn-in-place maneuvers.

efficiently in rough rocky terrain in the JPL Mars Yard. Figure 22 shows the results of a typical experiment with the FIDO rover running the lattice planner onboard to navigate autonomously amid dense rocks. In this experiment, the rover was given a command to drive to a goal 15 m directly in front of it, as shown by the black line in the top of the figure. This motion was infeasible due to large rock formations. However, the rover, under guidance of the lattice planner, negotiated this maze-like and previously unknown environment and found a feasible path (white dots) to accomplish its mission, despite a very limited perception horizon of 3 m and ±40-deg field of view. The lattice planner was configured as above, including a high-fidelity region of 21×21 map cells (L_∞-radius of 2 m), centered around the rover. The rover traversed approximately 30 m during this mission and achieved the goal successfully (only the first two-thirds of the rover path are shown in the photograph due to the limited field of view of the camera). No path tracking was used, and the rover executed verbatim the smooth and feasible motion computed by the lattice planner.

The rover used a single 1.6-GHz CPU and 512 MB of RAM, shared among all processes of the rover, including state estimation, stereo vision perception, and communication systems. The planner was ported to the VxWorks™ hard real-time operating system that runs on the rover's computer. We have not had a chance to optimize memory usage of our planner implementation; nevertheless, the peak memory usage of the lattice planner over all our experiments with the FIDO rover was less than 100 MB. The bottom part of Figure 22 shows the semilog plot of the onboard lattice planner run time per replan cycle, averaging at approximately 10 Hz. This plot serves well to illustrate two points regarding typical planner run time onboard FIDO: the computation time per replanning operation can vary greatly (depending on the difficulty of the problem at hand), and the replanning run time was frequently lower than the time resolution of the rover's operating system (5 ms), which is observed via the bottom-limited segments of the plot.

Owing to limited perception and previously unknown environment, the rover entered into a number of difficult planning scenarios during its mission. First, the rover steered out of the potential cul-de-sac by exploiting a high-curvature maneuver. Next, after going around the bend in the hope of resuming its

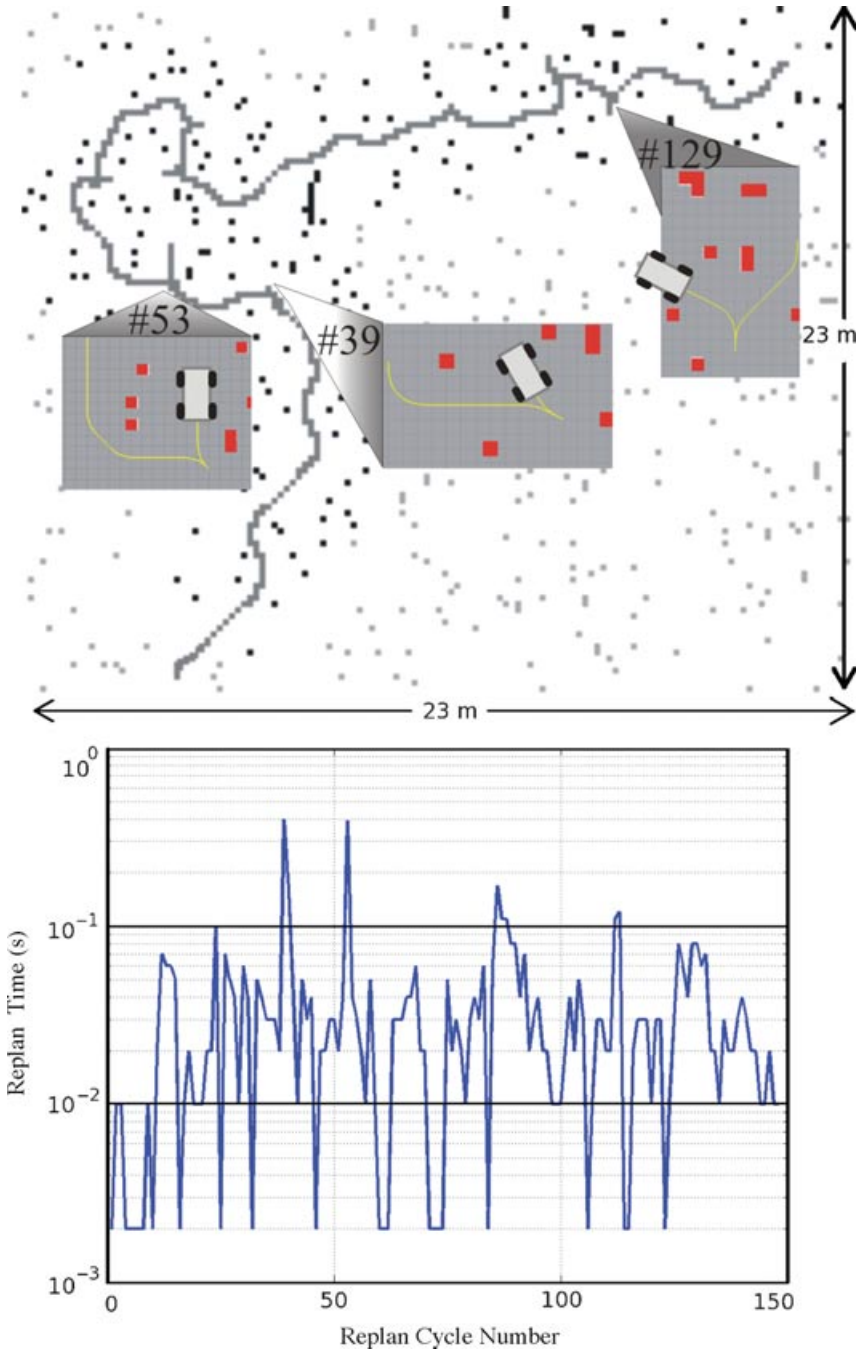


Figure 21. A simulated experiment of traversing about 500 m among previously unknown obstacles. Top: The first 40 m of the path are shown for clarity. Note that all motions generated by the planner were globally feasible, and backing-up maneuvers were generated automatically when necessary. Bottom: Planner run times.

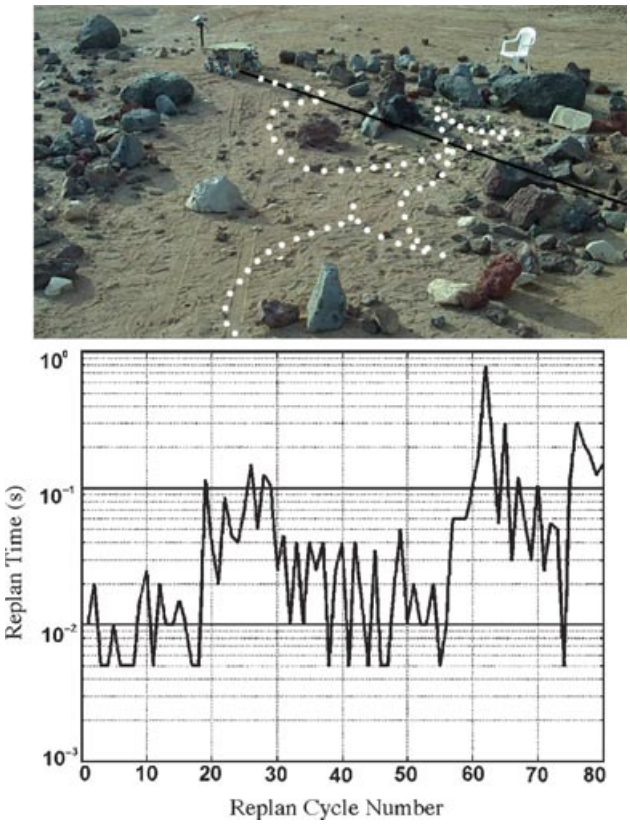


Figure 22. A field experiment in the JPL Mars Yard. Top: The FIDO rover was commanded to go straight 15 m (black line). The rover navigated autonomously among previously unknown maze-like obstacles, while running the graduated fidelity lattice planner onboard. White dotted line is the path traversed by FIDO. The rover encountered multiple difficult planning scenarios due to the very limited perception. It traveled approximately 30 m in order to achieve its goal. Bottom: Throughout numerous field experiments, the lattice planner onboard FIDO averaged replanning frequency of approximately 10 Hz.

course along the straight line, it arrived at a narrow cul-de-sac. At this point, a portion of the previously computed rover path was invalidated as soon as the rover's perception system recognized that the passage was closed off. However, thanks to D* capacity to repair the plan efficiently, a new plan was found that guided the rover through an n-point U-turn to resume its course. Shortly afterward, the rover appeared in another tricky situation (bottom right of the path), where it had to back out in order to continue navigating. Throughout the experiment, the lattice planner regularly updated the smooth and feasible motion plan of the rover, despite many challenges during its mission.

6. CONCLUSION AND FUTURE WORK

This work has been motivated by a fairly acute need to endow our field robots with sufficient understanding of their own mobility to allow them to efficiently plan correct and intricate paths in response to their challenging surroundings. Our efforts have produced a search space designed for differentially constrained motion planning in the continuous cost fields used in the field robot applications of today.

Our approach inherits its understanding of mobility from a competent and high-fidelity, real-time trajectory generator. It uses this module to construct what amounts to an ideal discrete search space:

- it is of high enough dimension to enforce relevant state continuity
- its controls acquire goal states exactly
- its controls satisfy arbitrary differential constraints, so they encode only feasible motions
- its controls are a reduced set in a path sampling sense, meaning unnecessary computation is avoided

This search space can be generated offline, and because of this property, many planning computations peculiar to differentially constrained motion planning in continuous cost fields can also be precomputed offline to enhance efficiency. This ideal search space can be searched using an ideal heuristic, constructed by a heuristic planner itself, from a world free of obstacles.

With all of the above elements in place, the daunting problem of optimal, efficient, smooth differentially constrained motion planning in the presence of obstacles is reduced to an implementation of heuristic search of a graph. We have shown that the A* algorithm and even the D* algorithm can be configured to run on this search space even while the topology of the space is dynamically changing.

State lattice planners (planners based on the state lattice search space) are resolution complete because the control set can be automatically adjusted to generate new controls as resolution increases and the space is searched systematically. Such planners are both optimal and smooth because precision controls that acquire states exactly permit the redirection of backpointers without introducing discontinuities in the

solution path. In algorithms derived from dynamic programming, such redirection is fundamental to optimal sequential search with nonuniform edge costs. The completeness, smoothness, satisfaction of differential constraints, and optimality of state lattice planners result from the precision of the trajectory generator used to create the control set.

State lattice planners are also efficient due to a principled implementation of a path sampling policy. We have shown that they are up to two orders of magnitude faster than our best-effort BL planner, while remaining optimal and smooth, while also allowing efficient replanning and significant precomputation. They are not as fast as a grid planner when obstacles are present—being roughly an order of magnitude slower than a 16-connected grid. However, severe problems with grid planners producing infeasible plans motivated the work to start with. According to our results, a mere 3 years of operation of Moore's law should have been enough to allow grids to be abandoned long ago, at least for near-field planning, so the common use of infeasible grid search spaces is difficult to justify on computational grounds.

The contribution of this work is not a planning algorithm. The contribution is a principled mechanism to construct an efficient, precision, differentially constrained search space upon which any planner may operate. We have also presented a compelling case for why it is superior to prevailing alternatives.

Future work includes further research in optimal sampling in the space of motions to further increase the effectiveness of search spaces. Further, the state lattice planning approach appears to lend itself well to other applications, including mobile manipulation, and we hope to explore these ideas in the future.

ACKNOWLEDGMENTS

This research was conducted at the Robotics Institute of Carnegie Mellon University, sponsored by NASA/JPL as part of the Mars Technology Program.

REFERENCES

Agarwal, P., Amenta, N., Aronov, B., & Sharir, M. (1996, August). Largest placements and motion planning of a convex polygon. In Proceedings 2nd Annual Workshop Algorithmic Foundations of Robotics, Toulouse, France (pp. 143–154).

Agarwal, P., Aronov, B., & Sharir, M. (1999). Motion planning for a convex polygon in a polygonal environment. GEOMETRY: Discrete and Computational Geometry, 22(2), 201–221.

Alt, H., Fleischer, R., Kaufmann, M., Mehlhorn, K., Naher, S., Schirra, S., & Uhrig, C. (1990, June). Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. In Proceedings of the ACM Symposium on Computational Geometry, Berkeley, CA (pp. 281–289).

Anisi, D. A., Hamberg, J., & Hu, X. (2003). Nearly time-optimal paths for a ground vehicle. Journal of Control Theory and Applications, 1(1), 2–8.

Barraquand, J., Kavraki, L., Latombe, J.-C., Li, T.-Y., Motwani, R., & Raghavan, P. (1996, October). A random sampling scheme for robot path planning. In G. Giralt & G. Hirzinger (Eds.), Proceedings of the 7th International Symposium on Robotics Research, Osaka, Japan (pp. 249–264). New York: Springer.

Barraquand, J., & Latombe, J.-C. (1990, May). A Monte-Carlo algorithm for path planning with many degrees of freedom. In Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH (pp. 1712–1717).

Barraquand, J., & Latombe, J.-C. (1991, April). Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA (vol. 3, pp. 2328–2335).

Bicchi, A., Marigo, A., & Piccoli, B. (2002). On the reachability of quantized control systems. IEEE Transactions on Automatic Control, 47(4), 546–563.

Bohlin, R. (2001, November). Path planning in practice; Lazy evaluation on a multi-resolution grid. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots & Systems, Maui, HI (pp. 49–54).

Bohlin, R., & Kavraki, L. (2000, April). Path planning using lazy PRM. In Proceedings of the IEEE International Conference on Robotics & Automation, San Francisco, CA (pp. 521–528).

Branicky, M., LaValle, S., Olson, S., & Yang, L. (2001, May). Quasi-randomized path planning. In Proceedings of the International Conference on Robotics and Automation, Seoul, Korea (pp. 1481–1487).

Canny, J., Rege, A., & Reif, J. (1991). An exact algorithm for kinodynamic planning in the plane. Discrete and Computational Geometry, 6, 461–484.

Canny, J. F. (1988). The complexity of robot motion planning. Cambridge, MA: MIT Press.

Casal, A. (2001). Reconfiguration planning for modular self-reconfigurable robots. Ph.D. thesis, Aeronautics and Astronautics Department, Stanford University, Stanford, CA.

Cherif, M. (1999, May). Kinodynamic motion planning for all-terrain wheeled vehicles. In Proceedings of the IEEE International Conference on Robotics & Automation, Detroit, MI (pp. 317–322).

Donald, B., & Xavier, P. (1995). Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. Algorithmica, 14(6), 443–479.

Dubins, L. E. (1957). On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American Journal of Mathematics, 79, 497–516.

Ferguson, D., & Stentz, A. (2006, March). Multi-resolution Field D*. In Proceedings International Conference on Intelligent Autonomous Systems (IAS), Tokyo, Japan (pp. 65–74).

Fernandes, C., Gurvits, L., & Li, Z. X. (1991, April). A variational approach to optimal nonholonomic moton planning. In Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA (pp. 680–685).

Fraichard, T., & Ahuactzin, J.-M. (2001, May). Smooth path planning for cars. In Proceedings IEEE International Conference on Robotics and Automation, Seoul, Korea (pp. 3722–3727).

Fraichard, T., & Scheuer, A. (2004). From Reeds and Shepp's to continuous-curvature paths. IEEE Transactions on Robotics, 20(6), 1025–1035.

Frazzoli, E., Dahleh, M., & Feron, E. (2001, June). Real-time motion planning for agile autonomous vehicles. In Proceedings of the American Control Conference, Arlington, VA (pp. 43–49).

Gottschalk, S., Lin, M., & Manocha, D. (1996, August). OBBtree: A hierarchical structure for rapid interference detection. In Proceedings of ACM SIGGRAPH, New Orleans, LA (pp. 171–180).

Green, C., & Kelly, A. (2007, November). Toward optimal sampling in the space of paths. In Proceedings of the International Symposium of Robotics Research, Hiroshima, Japan (pp. 171–180).

Hart, P., Nilsson, N., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.

Howard, T., & Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. International Journal of Robotics Research, 26(2), 141–166.

Hsu, D. (2000). Randomized single-query motion planning in expansive spaces. Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA.

Jean, F. (2001). Complexity of nonholonomic motion planning. International Journal of Control, 74(8), 776–782.

Kavraki, L. (1994). Random networks in configuration space for fast path planning. Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA.

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

Kelly, A., Amidi, O., Happold, M., Herman, H., Pilarski, T., Rander, P., Stentz, A., Vallidis, N., & Warner, R. (2004, June). Toward reliable off-road autonomous vehicle operating in challenging environments. In Proceedings of the International Symposium on Experimental Robotics, Singapore (pp. 599–608).

Kelly, A., & Nagy, B. (2002). Reactive nonholonomic trajectory generation via parametric optimal control. International Journal of Robotics Research, 22(7/8), 583–601.

Kindel, R. (2001). Motion planning for free-flying robots in dynamic and uncertain environments. Ph.D. thesis, Aeronautics and Astronautics Department, Stanford University, Stanford, CA.

Knepper, R., & Kelly, A. (2006, October). High performance state lattice planning using heuristic look-up tables. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Beijing, China (pp. 3375–3380).

Koenig, S., & Likhachev, M. (2002, March). D* Lite. In Proceedings of the AAAI Conference of Artificial Intelligence (AAAI), Edmonton, AB, Canada (pp. 476–483).

Kuffner, J. (1999). Autonomous agents for real-time animation. Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA.

Lacaze, A., Moscovitz, Y., DeClaris, N., & Murphy, K. (1998, September). Path planning for autonomous vehicles driving over rough terrain. In Proceedings of the IEEE International Symposium on Intelligent Control, Gaithersburg, MD (pp. 50–55).

Lamiraux, F., & Laumond, J.-P. (2001). Smooth motion planning for car-like vehicles. IEEE Transactions on Robotics and Automation, 17(4), 498–501.

Latombe, J.-C. (1991). Robot motion planning. Boston: Kluwer.

LaValle, S., Branicky, M., & Lindemann, S. (2004). On the relationship between classical grid search and probabilistic roadmaps. International Journal of Robotics Research, 23(7/8), 673–692.

LaValle, S. M. (2006). Planning algorithms. Cambridge, UK: Cambridge University Press.

LaValle, S. M., & Kuffner, J. J. (2001). Rapidly-exploring random trees: Progress and prospects. In B. Donald, K. M. Lynch, & D. Rus (Eds.), Algorithmic and Computational Robotics: New Directions (pp. 293–308). Wellesley, MA: AK Peters, Ltd.

Lindemann, S., & LaValle, S. (2003, October). Current issues in sampling-based motion planning. In Proceedings of the International Symposium of Robotics Research, Siena, Italy (pp. 36–54).

Lindemann, S., & LaValle, S. (2004, June). Steps toward derandomizing RRTs. In Proceedings of the Fourth International Workshop on Robot Motion and Control, Puszczykowo, Poland (pp. 271–277).

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. IEEE Transactions on Computers, C-32(2), 108–120.

Lozano-Perez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM, 22(10), 560–570.

Morris, A., Silver, D., Ferguson, D., & Thayer, S. (2005, April). Towards topological exploration of abandoned mines. In Proceedings of the IEEE International Conference on Robotics, Barcelona, Spain (pp. 2117–2123).

Natarajan, B. K. (1988). The complexity of fine motion planning. International Journal of Robotics Research, 7(2), 36–42.

Pai, D., & Reissell, L.-M. (1998). Multiresolution rough terrain motion planning. IEEE Transactions on Robotics and Automation, 14(1), 19–33.

Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving. Boston: Addison-Wesley Longman Publishing Co.

Pivtoraiko, M., & Kelly, A. (2005a). Constrained motion planning in discrete state spaces. In Field and service robotics (vol. 25, pp. 269–280). Berlin: Springer.

Pivtoraiko, M., & Kelly, A. (2005b, August). Differentially constrained motion replanning using state lattices with graduated fidelity. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, AB, Canada (pp. 2611–2616).

Pivtoraiko, M., & Kelly, A. (2008, September). Differentially constrained motion replanning using state lattices with graduated fidelity. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Nice, France (pp. 2611–2616).

Pivtoraiko, M., Knepper, R. A., & Kelly, A. (2007). Optimal, smooth, nonholonomic mobile robot motion planning in state lattices (Tech. Rep. CMU-RI-TR-07-15). Pittsburgh, PA: Robotics Institute, Carnegie Mellon University.

Reeds, J. A., & Shepp, L. A. (1990). Optimal paths for a car that goes both forwards and backwards. Pacific Journal of Mathematics, 145(2), 367–393.

Reif, J. (1979, October). Complexity of the mover's problem and generalizations. In Proceedings of IEEE Symposium on Foundations of Computer Science, San Juan, PR (pp. 421–427).

Sanchez, G., & Latombe, J.-C. (2001, November). A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In Proceedings of International Symposium on Robotics Research, Lorne, Victoria, Australia (pp. 403–417).

Sanchez, G., & Latombe, J.-C. (2002). On delaying collision checking in PRM planning: Application to multi-robot coordination. International Journal of Robotics Research, 21(1), 5–26.

Scheuer, A., & Fraichard, T. (1997, April). Collision-free and continuous-curvature path planning for car-like robots. In Proceedings of the IEEE International Conference on Robotics and Automation, Albuquerque, NM (pp. 867–873).

Scheuer, A., & Laugier, C. (1998, October). Planning sub-optimal and continuous-curvature paths for car-like robots. In Proceedings of the International Conference on Robotics and Automation, Victoria, BC, Canada (pp. 25–31).

Stentz, A. (1995, August). The focussed D* algorithm for real-time replanning. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland (pp. 1652–1659).