Hierarchical Motion Planning With Dynamical Feasibility Guarantees for Mobile Robotic Vehicles

Raghvendra V. Cowlagi, *Member, IEEE*, and Panagiotis Tsiotras, *Senior Member, IEEE*

***Abstract*—Motion planning for mobile vehicles involves the solution of two disparate subproblems: the satisfaction of high-level logical task specifications and the design of low-level vehicle control laws. A hierarchical solution of these two subproblems is efficient, but it may not ensure compatibility between the high-level planner and the constraints that are imposed by the vehicle dynamics. To guarantee such compatibility, we propose a motion-planning framework that is based on a special interaction between these two levels of planning. In particular, we solve a special shortest path problem on a graph at a higher level of planning, and we use a lower level planner to determine the costs of the paths in that graph. The overall approach hinges on two novel ingredients: a graph-search algorithm that operates on sequences of vertices and a lower level planner that ensures consistency between the two levels of hierarchy by providing meaningful costs for the edge transitions of a higher level planner using dynamically feasible, collision-free trajectories.**

***Index Terms*—Autonomous mobile robots, consistency, graph search, kinodynamic motion planning, motion planning.**

I. Introduction

THE problem of motion planning and control for autonomous mobile vehicles deals with finding appropriate control inputs such that the vehicle's resulting motion satisfies the requirements of a specified task. This problem is inherently complex because it involves two disparate subproblems: 1) the satisfaction of the vehicular task specifications, which requires tools from combinatorics and/or formal methods; and 2) the design of the vehicle control laws, which requires tools from dynamical systems and control theory. This inherent dichotomy spawns a natural approach to the solution of the motion-planning problem: a hierarchical separation of the aforementioned subproblems.

Consider, for instance, the vehicular task to traverse the environment from a given initial point to a given destination, while avoiding obstacles [1], [2]. For this problem, hierarchical separation has often been used in motion planners to explicitly incorporate the vehicle's kinematic and dynamic constraints [3]–[10].

Manuscript received January 23, 2011; revised June 20, 2011; accepted October 3, 2011. Date of publication November 10, 2011; date of current version April 9, 2012. This paper was recommended for publication by Associate Editor T. Simeon and Editor J.-P. Laumond upon evaluation of the reviewers' comments.

R. V. Cowlagi is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: rcowlagi@mit.edu).

P. Tsiotras is with the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: tsiotras@gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2011.2171613

In such hierarchically separated schemes, the higher level is mainly concerned with obstacle avoidance and with finding a geometric, obstacle-free *path* from the initial point to the destination. We call this level the *geometric path planning* level. The lower level accounts for the kinematic and dynamic constraints of the vehicle: It sufficiently smoothens the geometric path and imposes a suitable time parametrization along this path to obtain a feasible reference *trajectory*. We call this level the *trajectory planning* level. Additionally, a tracking controller generates the control inputs that enable the vehicle to track this trajectory.

The principal advantages of such hierarchical separation include computational efficiency and simplicity of implementation. Geometric planners typically use a discrete representation of the environment; hence, tools from combinatorics and/or formal methods may be applied to ensure satisfaction of the vehicular task specifications [11]. A serious drawback of this hierarchical separation is that the geometric planner has no knowledge of the vehicle's kinematic and dynamical constraints; as a result, it may produce either infeasible paths or unacceptably suboptimal paths. This inadequacy in providing guarantees of consistency between the higher and lower layers of planning and of optimality of the resultant trajectories has long been identified; consequently, its resolution is of acute interest for the development of autonomous mobile vehicles [11], especially for task specifications more complex than traveling from one point to another and for vehicle dynamical constraints that are sufficiently complex such that they cannot be ignored at the geometric planning stage.

In this paper, we propose a hierarchical motion-planning framework that addresses the aforementioned inadequacy of hierarchical planners. The proposed framework, which is tailored specifically for the planar motion of robotic vehicles, rests on a novel mode of interaction between the geometric path planner and the vehicle trajectory planner. This interaction is enabled by a special shortest path problem on graphs involving costs that are defined on *multiple* edge transitions, instead of the usual single-edge transition costs. These transition costs are provided by a low-level trajectory planner. The proposed framework is intended as a basic step toward the design of hierarchically consistent motion planners that combine discrete geometric planning algorithms with trajectory generation algorithms.

A. Related Work

The geometric path planner in this paper is based on cell decompositions [1, Ch. 5], [12]. These involve a partitioning of the environment into convex, nonoverlapping regions called *cells*. A cell is classified either as FREE (if it contains no obstacles) or FULL (if it contains no free space). A graph is associated with the cell decomposition such that each FREE cell is represented by a vertex, and geometric adjacencies of the FREE cells are represented by edges. A path from a prespecified initial cell to a prespecified goal cell in this graph then corresponds to a sequence of FREE cells from the initial cell to the goal cell in this graph.

Triangular and trapezoidal decompositions [1, Ch. 6], [13] are widely used exact cell decomposition techniques for environments with polygonal obstacles, whereas quadtree-based methods [14]–[16] (that employ recursive decompositions of MIXED cells into four subcells until all cells are either FREE or FULL) are popular approximate cell decomposition techniques. Path planners using *multiresolution* cell decompositions have also been proposed, for instance, in [17]–[19].

As previously mentioned, hierarchical[1] approaches are often used in motion planning [3]–[10] to separate the problem into a high-level, discrete, geometric planning level and a low-level, continuous, trajectory planning level. In the context of low-level trajectory planning, in [20]–[23] time-optimal trajectory planning along prespecified geometric paths for specific vehicle dynamics is discussed. Other related works in the literature include [24], which uses a special history-based cost approach that closely matches the approach taken in our study; [25], which deals with kinodynamic planning for robotic manipulators; [26], which uses a hybrid model to describe the motion of a rotorcraft in terms of preprogrammed maneuvers; and [27], which discusses trajectory planning that is based on the solution of the Hamilton–Jacobi–Bellman equation.

An alternative to hierarchical motion planning, especially in the context of the point-to-point trajectory generation problem, is to search directly in the state space of the vehicle. Deterministic state space searches have been considered (cf., [28]); however, such approaches may be impractical for high-dimensional state spaces. Probabilistic roadmap methods [1, Ch. 7], [29]–[32] and methods that use rapidly exploring random trees (RRTs) [33]–[37] are among the most popular, recent works that address the vehicle's kinematic and dynamic constraints during motion planning. In these methods, random samples of the obstacle-free space are connected to each other by feasible trajectories, and the resulting graph is searched for a sequence of connected samples from the initial state to the goal state. Sampling-based algorithms require efficient low-level collision detection and trajectory planning algorithms to find collision-free trajectories between different samples [34].

Motion planners that use cell decompositions coupled with feedback control laws [38]–[42] provide methods to generate reference vector fields that guarantee feasible transitions through any given sequence of cells without intersecting any other cell. In [43], [44], and, in a slightly different context, [45], this idea is also used to develop solutions that are guaranteed to satisfy both aspects of motion planning and control: vehicular task specifications that are expressed as temporal logic formulas and kinematic and dynamic constraints that are expressed as differential equations. The common idea that is used in all of these works is to make the geometric planner *independent* of the vehicle dynamics, that is, to ensure that *any* sequence of cells can be feasibly traversed from *any* initial vehicle state.

[1]In addition to the "discrete-continuous" hierarchy that is considered in this paper, hierarchical motion planning may also refer to a high-level continuous geometric path planner coupled with a low-level time parametrization scheme. In this paper, we regard the latter notion of hierarchy as a "low-level" trajectory generation scheme.

B. Contributions of This Paper

The principal contribution of this paper is a new motion-planning framework that maintains the distinction between the discrete planning and the trajectory planning strategies. The advantage of this distinction is that either planner may be developed independently of the other: The discrete planner may be tailored to satisfy vehicular task specifications (finding low-cost, obstacle-free paths is an example of such a task), whereas trajectory planning schemes that are based on control theoretic ideas may be tailored to cope with complex vehicle dynamics. However, in contrast with previous similar approaches, we also provide a "protocol" to "interface" these two planners such that compatibility of their solutions is maintained. The result of a higher level geometric planner in the proposed framework is a path that corresponds to a sequence of cells; consistency between the two levels of planning is ensured by providing a guarantee that this sequence of cells can be feasibly traversed by the vehicle. The proposed framework is, thus, a step toward the development of motion planners that systematically combine results and techniques from different disciplines, such as formal methods and control theory, to generate provably correct control laws that enable the vehicle to satisfy complex task specifications, as envisioned in [11].

A secondary contribution of this paper is an efficient and flexible algorithm of independent interest that finds a path in a graph minimizing a cost that is defined on multiple edge transitions. A basic version of this algorithm was originally reported in [46]; in this paper, we provide a more general, flexible version of this algorithm, along with detailed numerical simulation results that demonstrate its efficiency.

The rest of this paper is organized as follows: in Section II, we introduce the idea of using history-dependent transition costs in path planning. In Section III, we provide an efficient and flexible algorithm that finds paths minimizing history-based transition costs. In Section IV, we discuss the proposed motion-planning framework that is based on the path-finding algorithm described in Section III. In Section V, we discuss implementations of the TILEPLAN algorithm that is used in the proposed motion-planning framework, and in Section VI, we provide sample numerical simulation results of the proposed approach, as well as comparisons with other motion planners. Section VII concludes with a summary of our contributions.

II. HISTORY-BASED TRANSITION COSTS

Geometric path planning algorithms that are based on workspace cell decompositions provide no guarantees that the resultant channel of cells can be feasibly traversed by a vehicle subject to kinematic and dynamical constraints. At first glance, one may argue that this is but an artifact of an unfortunate or

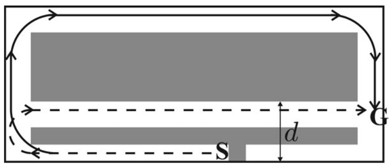


Fig. 1. Counterexample for path planning without kinematic constraints.

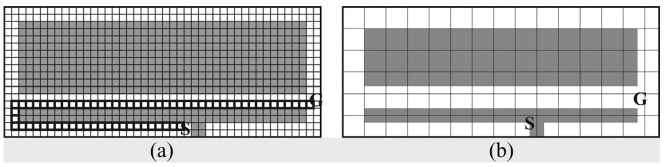


Fig. 2. Problems with geometric path planning using cell decompositions. (a) No pair of successive cells is itself infeasible. (b) Cells are too large; all cells are MIXED.

inappropriate choice of the edge cost function in the associated graph. We provide a counterexample against this argument.

A. Motivating Example

Consider the path planning problem that is depicted in Fig. 1, where S denotes the initial position, G denotes the goal, and the dark areas denote obstacles. Consider two vehicles A and B whose minimum radii of turn are kinematically constrained by $r^A_{\min}$ and $r^B_{\min}$, respectively, such that $r^A_{\min} \leq d$ and $r^B_{\min} > d$. Clearly, the dashed path in Fig. 1 is feasible for vehicle A, but not for vehicle B. A path planning algorithm for B ought to result in the bold path shown in Fig. 1.

Fig. 2 depicts the same problem with a uniform cell decomposition. The channel that contains the dashed path in Fig. 1 is denoted by cells with bold outlines. Such a channel is obviously not traversable by vehicle B. However, notice that no *pair of successive cells* is *by itself* infeasible, i.e., a channel that is defined by two successive cells alone always contains a feasible path. Stated differently, for any two adjacent cells, there is no cell-dependent property associated with the two adjacent cells that can be penalized by an edge cost function in order to prevent the graph search from generating a channel such as the one shown in Fig. 2(a).

It may be argued that a feasible path is guaranteed to exist in *any* channel if the dimensions of the cells are large enough.[2] The aforesaid counterexample also serves to illustrate the fact that such a choice of cells may be too restrictive in practice. Fig. 2(b) shows that large cells may not capture details of the environment, i.e., the number of MIXED cells could be too large for the cell decomposition to be useful for path planning.

In light of the preceding observations, we propose in this paper an approach to find paths in the cell decomposition that minimize a cost that is defined on multiple edge transitions—called *histories*—instead of costs that are defined on single-edge transitions.

[2]In [47], it is shown that a curvature-bounded path with local curvature that is less than or equal to $1/r_{\min}$ exists in a polygonal channel if the width w of the channel satisfies $w \geq \tau r_{\min}$, where $\tau \approx 1.55$.

B. Problem Formulation

Consider a cell decomposition of the environment that consists of N cells. For now, we make no assumptions that concern the geometry of the cells involved in the decomposition. The topological graph associated with the given cell decomposition is a graph $\mathcal{G} := (V, E)$ such that each element in the set of vertices V corresponds to a unique, obstacle-free cell. Two vertices are *adjacent* in $\mathcal{G}$ if the corresponding cells are geometrically adjacent.[3] The edge set E is the collection of all ordered pairs $(i, j) \subset V \times V$ such that the vertices i and j are adjacent. For the given initial and goal vertices $i_{\mathrm{S}}, i_{\mathrm{G}} \in V$, an *admissible path* π in $\mathcal{G}$ is a finite sequence $(j_0, j_1, \ldots, j_P)$ of vertices (with no repetition) such that $j_k \in V$, $(j_{k-1}, j_k) \in E$, for each $k = 1, \ldots, P$, with $j_0 = i_{\mathrm{S}}$, $j_P = i_{\mathrm{G}}$. To formalize the concept of histories, we define, for every integer $H \geq 0$

$$V_H := \{(j_0, \ldots, j_H) : (j_{k-1}, j_k) \in E, \quad k = 1, \ldots, H \\ j_k \neq j_m, \ \text{for } k, m \in \{0, \ldots, H\}, \ \text{with } k \neq m\}.$$

An element of the set V_{H+1} is called an *H-history*. In what follows, we denote by $[I]_k$ the k^{th} element of this $(H+1)$-tuple, and by $[I]^m_k$ the tuple $([I]_k, [I]_{k+1}, \ldots, [I]_m)$, for $k < m \leq H+1$. We associate with each H a nonnegative cost function $\tilde{g}_H : V_H \rightarrow \mathbb{R}_+$, and state a shortest path problem with transition costs that are defined on histories as follows.

Problem 1 (H-cost Shortest Paths): Let $H \geq 0$, and let $i_{\mathrm{S}}, i_{\mathrm{G}} \in V$ be the given initial and goal vertices such that any admissible path in $\mathcal{G}$ contains at least $H+1$ vertices. Let the H-cost of the path $\pi = (j_0, \ldots, j_P)$ be defined by

$$\tilde{\mathcal{J}}_H(\pi) := \sum_{k=H+1}^{P} \tilde{g}_{H+1}\left((j_{k-H-1}, j_{k-H}, \ldots, j_k)\right). \tag{1}$$

Find an admissible path π^* in $\mathcal{G}$ such that $\tilde{\mathcal{J}}_H(\pi^*) \leq \tilde{\mathcal{J}}_H(\pi)$ for every admissible path π in $\mathcal{G}$.

Note that the H-cost of a path is defined as the sum of the costs of H-histories in that path. According to this convention, the 0-cost of a path is the standard notion of cost, i.e., the sum of edge weights, because 0-histories are precisely the edges in E with $V_1 = E$. In other words, the H-cost shortest path problem for $H = 0$ is the standard shortest path problem on a graph with weighted edges.

It is possible to transform Problem 1 into an equivalent standard shortest path problem on a "lifted" graph $\mathcal{G}_H$ whose vertices are the elements of V_H. This transformation enables a clear conceptualization of our proposed algorithm to solve Problem 1 in light of the fact that the solution to the standard shortest path problem is well known. For instance, the so-called *label-correcting algorithms* [48] provide efficient solutions to the standard shortest path problem. Well-known examples of label-correcting algorithms include the Bellman–Ford, Dijkstra [48], [49], and the A* [2], [50] algorithms.

We define adjacency relations between the elements of V_H as follows. Let $I, J \in V_H$; then J is adjacent to I if $[I]_k = [J]_{k-1}$,

[3]In Section V, we consider rectangular cells and geometric adjacency in the sense of 4-connectivity between cells.

for every $k = 2, \ldots, H+1$, and $[I]_1 \neq [J]_{H+1}$. Let E_H denote the edge set of the graph $\mathcal{G}_H$ that consists of all ordered pairs (I, J) such that J is adjacent to I.

For the given initial and terminal vertices i_S, $i_G \in V$, an admissible path Π in $\mathcal{G}_H$ is a finite sequence $(J_0, \ldots, J_Q)$ of vertices (with no repetition) such that $(J_{k-1}, J_k) \in E_H$, for each $k = 1, \ldots, Q$, with $[J_0]_1 = i_S$, and $[J_Q]_{H+1} = i_G$. Note that every admissible path $\Pi = (J_0, \ldots, J_Q)$ in $\mathcal{G}_H$ uniquely corresponds to an admissible path $\pi = (j_0, \ldots, j_P)$ in $\mathcal{G}$, with $P = Q + H$ and $[J_k]_m = j_{k+m-1}$, for each $k = 0, 1, \ldots, Q-1$, and $J_Q = (j_{P-H}, \ldots, j_P)$. We introduce a nonnegative cost function $g_H \colon E_H \to \mathbb{R}_+$ that is defined by

$$g_H((I, J)) := \tilde{g}_{H+1}([I]_1^{H+1}, [J]_{H+1}), \quad (I, J) \in E_H.$$

It follows that Problem 1 is equivalent to the standard shortest path problem on the graph $\mathcal{G}_H$, where the weight of an edge $(I, J) \in E_H$ is given by $g_H((I, J))$.

III. Path Planning With History-Dependent Costs

Problem 1 may be solved by first transforming it to a standard shortest path problem on the graph $\mathcal{G}_H$, and then executing a label-correcting algorithm such as Dijkstra's algorithm. A naïve, brute-force implementation of this approach is ill-advised because 1) $|V_H|$ and $|E_H|$ grow exponentially with H, and 2) the explicit construction of the graph $\mathcal{G}_H$ may be unnecessary to find the shortest path in $\mathcal{G}_H$.

In this section, we describe an algorithm that indirectly executes a label-correcting algorithm on $\mathcal{G}_H$, without constructing the entire graph beforehand. Since $|V_H|$ and $|E_H|$ grow exponentially with H, the execution time of *any* algorithm that solves Problem 1 *exactly* grows exponentially with H. This fact holds true for the proposed algorithm; however, we include in our algorithm a user-specified parameter that can dramatically reduce the execution time at the expense of (exact) optimality of the resultant path, i.e., the proposed algorithm exhibits a flexibility that allows the user to trade off execution time against optimality of the resultant path.

For the sake of clarity, we first present a basic version of our algorithm, namely, one that finds the optimal path and solves Problem 1 exactly. In Section III-D, we introduce the aforementioned user-specified parameter and discuss the effects of this parameter on the algorithm's execution time.

A. Description of the Basic Algorithm

Recall that a standard label-correcting algorithm maintains a set of vertices, which is called the *fringe* [51] (also referred to as the set of OPEN vertices [48], [49]), whose labels can potentially be reduced. The standard algorithm associates with each vertex $i \in V$ a *label*, which is an estimate of the least cost of a path from i_S to i, and a *backpointer*, which records the immediate predecessor of each vertex in the optimal path from i_S to i.

The definition of an admissible path in $\mathcal{G}_H$ from i_S to any vertex $i \in V$ requires only $[J_Q]_{H+1} = i$, where $J_Q \in V_H$ is the last vertex in this path. The first H elements of J_Q are unspecified, which implies that different admissible paths in $\mathcal{G}_H$ may have different terminal vertices in V_H. In the proposed algorithm, we recognize this fact by associating with each vertex $i \in V$ *multiple* H-histories instead of the backpointer in the standard label-correcting algorithm. Each history of i is a unique element $I \in V_{H+1}$ such that $[I]_{H+2} = i$. The proposed algorithm is a label-correcting algorithm that associates with each history of each vertex $i \in V$ a label. Accordingly, the fringe in the proposed algorithm is a collection of *pairs*, where each pair consists of a vertex in V and an index that refers to a particular history of that vertex.

Algorithm 1: H-Cost Shortest Path-Finding
Input: i_S, **Output:** d, h

```
procedure INITIALIZE()
1: for all $i \in V$, $m = 1, \ldots, |\mathcal{T}_i|$ do
2:     $d(i,m) \leftarrow \infty$, $h(i,m) \leftarrow$ NULL
3:     $\mathcal{P} \leftarrow \{(i,m) : \mathcal{N}_i \neq \varnothing$, and HISTORY$(i,m) \in \mathcal{N}_i\}$
4: for all $(i,m) \in \mathcal{P}$ do
5:     $I \leftarrow$ HISTORY$(i,m)$
6:     $d(i,m) \leftarrow \tilde{g}_{H+1}(I)$, $h(i,m) \leftarrow [I]_1^{H+1}$
procedure MAIN()
1: INITIALIZE()
2: while $\mathcal{P} \neq \varnothing$ do
3:     $(i,m) \leftarrow$ REMOVE$(\mathcal{P})$
4:     for all $j \in V$ such that $(i,j) \in E$ do
5:         $n \leftarrow$ INDEX$(([h(i,m)]_3^{H+1}, i, j), j)$
6:         $J \leftarrow$ HISTORY$(j,n)$
7:         $D_{i,m} \leftarrow d(i,m) + \tilde{g}_{H+1}([h(i,m)]_2, J)$
8:         if $d(j,n) > D_{i,m}$ then
9:             $d(j,n) \leftarrow D_{i,m}$
10:            $h(j,n) \leftarrow ([h(i,m)]_2^{H+1}, i)$
11:            $\mathcal{P} \leftarrow$ INSERT$(\mathcal{P}, (j,n))$
```

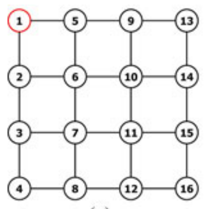
Fig. 3. Detailed pseudocode for the basic version of the proposed algorithm.

The pseudocode of the proposed algorithm is shown in Fig. 3. In each iteration, the algorithm updates the label that corresponds to a vertex–index pair, i.e., a particular history. Lines 8–11 update the fringe and the labels, similar to the standard label-correcting algorithm. Line 5 chooses the index that corresponds to the particular history (of the newly explored vertex j) being updated in that iteration. We use the following notation: For each vertex $i \in V$, $\mathcal{T}_i$ and $\mathcal{N}_i$ are defined by

$$\mathcal{T}_i := \{I \in V_H : [I]_{H+1} = i\} \tag{2}$$

$$\mathcal{N}_i := \{I \in V_{H+1} : [I]_1 = i_S, [I]_2^{H+2} \in \mathcal{T}_i\}. \tag{3}$$

Here, $\mathcal{P}$ denotes the fringe, and for each vertex $i \in V$ and $m \in \{1, \ldots, |\mathcal{T}_i|\}$, $h(i,m)$ denotes the m^{th} history of i, and $d(i,m)$ denotes the label associated with the m^{th} history of i. Finally, the procedure HISTORY(i,m) returns the m^{th} element of the set $\mathcal{T}_i$, and the procedure INDEX(I, i) returns the index of the history I in the set $\mathcal{T}_i$ if $I \in \mathcal{T}_i$, or returns 0 otherwise. The procedures INSERT and REMOVE depend on the specific data structure that is used to implement the fringe. In Section III-B, we consider a list that is sorted by the current values of labels.



(a)

$I \in V_2$	$\tilde{g}_2(I)$	$I \in V_2$	$\tilde{g}_2(I)$
$(1,2,3)$	5	$(5,6,2)$	5
$(1,2,6)$	6	$(5,6,7)$	7
$(1,5,6)$	2	$(5,6,10)$	8
$(1,5,9)$	8	$(2,6,5)$	12
$(2,6,10)$	5	$(2,6,7)$	8

(b)

Fig. 4. Graph and the history-based cost function used in Example 1.

The algorithm terminates when $\mathcal{P} = \emptyset$ (in Section III-B, we note a condition to terminate the algorithm earlier). After termination, the algorithm returns the labels d and the histories h for each vertex in V. For every vertex $i \neq i_S$ in V, we may then calculate the optimal path from i_S to i by recursively tracing the $(H+1)^{\text{th}}$ element of histories recorded by h: a process that is similar to recursively tracing the backpointer in the standard label-correcting algorithm. We illustrate the execution of the algorithm with a simple example.

Example 2: Consider the graph shown in Fig. 4(a), where $i_S = 1$. Let $H = 1$. Let $\tilde{g}_2$ be a nonnegative cost function that is given by the lookup table in Fig. 4(b) (for brevity, the values of some of the elements of V_2 are not shown). Note that

$$|\mathcal{T}_i| = \begin{cases} 2, & i \in \{1,4,13,16\} \\ 3, & i \in \{2,3,5,8,9,12,14,15\} \\ 4, & \text{otherwise} \end{cases}$$

i.e., the algorithm maintains at most four histories and labels for each vertex. Also note that $\mathcal{N}_3 = \{(1,2,3)\}$, $\mathcal{N}_6 = \{(1,5,6),(1,2,6)\}$, $\mathcal{N}_9 = \{(1,5,9)\}$, and $\mathcal{N}_i = \emptyset$ for $i \in \{1,\ldots,16\}\backslash\{3,6,9\}$.

We index the elements of $\mathcal{T}_j$ as 1, 2, 3, 4 corresponding to Up, Right, Down, Left edges of j, with reference to Fig. 4(a). If a particular edge is absent, the corresponding index applies to the next edge in the order that is listed earlier. For example, the indices of (5,6), (10,6), (7,6), (2, 6) $\in \mathcal{T}_6$ are 1, 2, 3, and 4, respectively, while the indices of (5, 1), (2, 1) $\in \mathcal{T}_1$ are 1 and 2, respectively.

Line 3 of the Initialize procedure results in $\mathcal{P} = \{(3, 1), (6, 1), (6, 4), (9, 3)\}$. From Fig. 4(b), Line 6 of the Initialize procedure results in

$$d(3,1) = 5, \quad d(6,1) = 2, \quad d(6,4) = 6, \quad d(9,3) = 8$$
$$h(3,1) = (1,2), \quad h(6,1) = (1,5)$$
$$h(6,4) = (1,2), \quad h(9,3) = (1,5).$$

Next, suppose we remove $(i,m) =$ (6,1) from the fringe in Line 3 of the Main procedure. Then, $\mathcal{P} = \{(3,1),(6,4),(9,3)\}$, and the **for** loop in Line 4 is executed for vertices 2, 5, 7, and 10. In particular, for vertex $j = 2$, Lines 5 and 6 result in $n = 2$, and $J = (6,2)$, respectively. From Fig. 4(b), it follows that $d(6,1) + \tilde{g}_2([h(6,1)]_2, 6, 2) = 2 + 5 = 7 < d(2,2) = \infty$ (by Line 2 of the Initialize procedure). Hence, Lines 9 and 10 result in $d(2,2) = 7$, and $h(2,2) = ([h(6,1)]_2, 6) = (5,6)$, while Line 11 results in $\mathcal{P} = \{(3,1),(6,4),(9,3),(2,2)\}$.

Similarly, the execution of the **for** loop in Line 4 of the Main procedure for vertices 5, 7, and 10 results in $\mathcal{P} = \{(3,1),(6,4),(9,3),(2,2),(7,1),(10,4),(5,2)\}$. The labels and histories at the end of the first iteration are

$$d(5,2) = 18, \qquad d(7,1) = 9, \qquad d(10,4) = 10$$
$$h(5,2) = (2,6), \quad h(7,1) = (5,6), \quad h(10,4) = (5,6).$$

B. Optimality and Performance

Different instances of label-correcting algorithms are obtained by implementation of the fringe using different data structures. For example, implementation of the fringe as a LIFO stack results in a breadth-first search; implementation of the fringe as a list that is sorted by the current labels results in Dijkstra's algorithm. In this section, we consider an instance of the proposed algorithm with the fringe that is implemented as a list which is sorted by the current labels, i.e., the Remove procedure returns $(i,m) = \arg\min\{d(i,m) : (i,m) \in \mathcal{P}\}$.

Proposition 3: For every vertex $i \in V$, suppose there exists at least one admissible path from i_S to i that contains History(i,m), for any $m \in \{1,\ldots,|\mathcal{T}_i|\}$. Let π^* be such an admissible path in $\mathcal{G}$ with the least cost. Then the proposed algorithm terminates with $d(i,m) = \tilde{\mathcal{J}}_H(\pi^*)$. Otherwise, the algorithm terminates with $d(i,m) = \infty$.

Proof: See [52]. ■

Proposition 3 asserts that the algorithm computes the minimum H-cost of paths from i_S to every vertex $i \in V$ and every history in $\mathcal{T}_i$. However, because we need to compute only the minimum H-cost from i_S to i_G for any history in $\mathcal{T}_{i_G}$, it is possible to terminate the algorithm earlier, as shown by the following result.

Proposition 4: Each pair (j,m), $j \in V$, $m = 1,\ldots,|\mathcal{T}_i|$ enters the set $\mathcal{P}$ at most once during the execution of the algorithm.

Proof: See [52]. ■

The conditions in Lines 8 and 11 of the Main procedure imply that a pair (j,m) is inserted in $\mathcal{P}$ only when the value of $d(j,m)$ can be reduced. It follows from Proposition 4 that once a pair (j,m) is removed from $\mathcal{P}$, the value of $d(j,m)$ cannot be further reduced. The implication of this fact is that we may terminate the algorithm after Line 3 if $i = i_G$.

Note that Proposition 4 closely resembles a similar, known result regarding the execution of Dijkstra's algorithm: Namely, each vertex enters the fringe at most once (see, for instance, [48], [49]). We may use Proposition 4 to characterize the execution time of the basic version of the proposed algorithm as follows: In the worst case, every pair (j,m) enters the set $\mathcal{P}$ exactly once. Thus, the maximum number of iterations (of the **while** loop in the Main procedure) is upper bounded by $\sum_{j=1}^{|V|} |\mathcal{T}_j| = |V_H| = O(|V|^H)$.

C. Numerical Simulation Results

Table I shows sample examples comparing the times required to construct the lifted graph $\mathcal{G}_H$ and then executing Dijkstra's algorithm, with the execution times of the proposed algorithm. In particular, the fourth, fifth, and sixth columns of Table I show, respectively, the absolute values of the maximum, the minimum,

TABLE I
EXECUTION TIME RATIOS: SAMPLE VALUES

$\lvert\mathcal{G}\rvert$	H	$\lvert\mathcal{G}_H\rvert$	Max. ratio	Min. ratio	Avg. ratio
6,400	1	25,280	95.49	0.937	5.039
10,000	1	39,600	56.78	0.993	4.756
14,400	1	57,120	17.07	1.017	3.726
22,500	1	89,400	36.19	1.221	4.574
6,400	2	74,888	43.01	1.199	4.646
10,000	2	117,608	59.43	1.249	5.901
14,400	2	169,928	222.4	1.396	13.15
16,900	2	199,688	73.27	1.322	6.901
2,500	3	85,056	141.3	1.263	8.831
4,900	3	169,456	238.4	1.215	11.26
6,400	3	222,456	630.7	1.231	29.87
10,000	3	350,056	359.7	1.182	21.47
900	4	79,472	155.1	1.440	16.63
1,600	4	145,872	1264	1.410	75.14
2,500	4	232,272	399.8	1.287	20.57
625	5	147,952	1294	1.788	93.06
1,225	5	306,072	2 761	0.834	125.2
225	6	120,532	1399	3.604	267.1
400	6	237,232	2091	1.697	226.6

TABLE II
COMPARISONS OF EXEC. TIME AND SUBOPTIMALITY: SAMPLE VALUES

$\lvert\mathcal{G}\rvert$	H	L	$\max_i\{\lvert\mathcal{T}_i\rvert\}$	Time ratio	Cost diff. (%)
22,500	1	1	4	2.424	18.13
22,500	1	2	4	1.434	0.144
14,400	2	1	12	8.055	34.43
14,400	2	3	12	2.645	0.108
14,400	2	5	12	1.698	0.000
10,000	3	2	36	15.62	2.379
10,000	3	4	36	7.119	0.146
10,000	3	5	36	5.521	0.036
2,500	4	3	100	31.90	0.662
2,500	4	5	100	17.75	0.155
2,500	4	7	100	12.20	0.038
1,225	5	3	284	121.7	1.170
1,225	5	5	284	69.11	0.543
1,225	5	7	284	47.15	0.243
625	6	2	780	681.0	4.727
625	6	5	780	257.6	0.496
625	6	11	780	107.1	0.148

and the average ratios of execution times. The graph $\mathcal{G}$ that is used in these simulations is the graph that arises out of a uniform cell decomposition with 4-connectivity, i.e., a graph of the form shown in Fig. 4(a). For each combination of H and $|\mathcal{G}|$, 30 trials were performed. In each of these trials, the structure of the graph $\mathcal{G}$ was kept constant, and the costs to transition H-histories, i.e., the costs of edge transitions in $\mathcal{G}_H$, were assigned randomly. The initial and goal vertices i_{S} and i_{G} were randomly assigned in each trial.

Table I indicates that the proposed algorithm executes up to three orders of magnitude faster, on average, and may execute up to four orders of magnitude faster in the best case than the alternative approach to first construct the lifted graph and then execute the search. Furthermore, the memory that is required to store the graph $\mathcal{G}_H$ is approximately K times the memory required by the proposed algorithm to store multiple histories of each vertex $j \in V$, where K is the valency of the graph $\mathcal{G}$.

D. Modifications for Further Efficiency

As mentioned previously, the execution time of the basic version of the proposed algorithm increases exponentially with H, which may slow down the algorithm for large values of H. To address this issue, we present in this section a simple modification of the basic version of the algorithm that dramatically reduces its execution time at the expense of optimality of the resultant path.

The algorithm that is presented in Fig. 3 maintains, for each vertex $j \in V$, a record of the costs-to-come to that vertex through each of its histories. To reduce the execution time of the algorithm, we may modify the algorithm such that it maintains, for each vertex $j \in V$, the costs-to-come for a fixed number L of histories. In particular, we will modify the proposed algorithm by inserting the following statements between Lines 5 and 8 in the MAIN procedure:

$$\mathcal{L}_i \leftarrow \{d(i,m) < \infty : m = 1, \ldots, |\mathcal{T}_i|\}$$

$$\textbf{if} |\mathcal{L}_i| = L \text{ and } D_{i,m} \geq \max\{\mathcal{L}_i\} \textbf{ then}$$

$$\textbf{continue}.$$

Accordingly, we delete from the set $\mathcal{N}_i$ that is defined in (3) all but the first L histories, ranked by increasing H-costs.

Table II shows a few sample ratios of the execution times of the proposed algorithm with the aforementioned modifications to the execution times without these modifications, and the corresponding average suboptimality of the resultant paths of the percentage increases in cost. For each combination of H, $|\mathcal{G}|$, and L, 30 trials were performed. In each of these trials, the structure of the graph $\mathcal{G}$ was kept constant and the costs to transition H-histories, i.e., the costs of edge transitions in $\mathcal{G}_H$, were assigned randomly. The initial and goal vertices were kept fixed in each trial: In particular, $i_{\mathrm{S}} = 1$ and $i_{\mathrm{G}} = |\mathcal{G}|$ were assigned.

Table II indicates that relatively small values of L speed up the original algorithm by up to three orders of magnitude, with relatively low increases in the cost of resultant paths. A similar observation has been reported in [24], for the specific case of $H = 1$ and $L = 1$.

IV. MOTION PLANNING WITH KINODYNAMIC FEASIBILITY GUARANTEES

In this section, we present a hierarchical motion-planning framework that is based on the H-cost shortest paths defined in the previous section. In this framework, the high-level geometric path planner searches for H-cost shortest paths in the cell decomposition graph $\mathcal{G}$. This geometric planner repeatedly invokes a special trajectory generation algorithm, which is called the *tile motion planner*, to determine the costs of H-histories. Because of the interaction between the geometric planner and the trajectory planner, the search for a channel of cells from the initial cell to the goal cell progresses *simultaneously* with the (piecewise) construction of a trajectory from the initial vehicle state to the goal state.

In this paper, we consider only vehicle dynamical models where the vehicle configuration is described by its position in the plane and its orientation, i.e., the configuration space $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. This class of vehicle models includes all navigational models of aerial and terrestrial vehicles that move in the horizontal plane. Note that the vehicle *state space* $\mathcal{D}$ may be larger than the configuration space.

Algorithm 2: Tile Motion Planning Algorithm
Input: τ, ξ_0, **Output:** t_1, ξ_1, u, Λ

```
procedure TILEPLAN($\tau, \xi_0$)
1: Determine if there exist $t_\mathrm{f} \in \mathbb{R}$ and admissible control input $u \in \mathcal{U}_{[0,t_\mathrm{f}]}$ such that $\xi(\cdot\,;\xi_0,u)$ satisfies
       $\mathsf{x}(\xi(t;\xi_0,u)) \in \bigcup_{k=1}^{H}\mathsf{cell}([J^\tau]_k),\ t \in (0,t_\mathrm{f})$,   (4)
       $\mathsf{x}(\xi(t_\mathrm{f};\xi_0,u)) \in \mathsf{cell}([J^\tau]_H) \cap \mathsf{cell}([J^\tau]_{H+1})$   (5)
2: if $\exists t_\mathrm{f}$ and $\exists u$ then
3:     Find $t_1$ such that
           $\mathsf{x}(\xi(t_1;\xi_0,u)) \in \mathsf{cell}([J^\tau]_1) \cap \mathsf{cell}([J^\tau]_2)$   (6)
4:     Return $t_1$, $u_{[0,t_1]}$, $\xi_1 := \xi(t_1;\xi_0,u)$, and
           $\Lambda := \int_0^{t_1} \ell(\xi(t;\xi_0,u),u,t)\ \mathrm{d}t$   (7)
5: else
6:     Return $\Lambda = \infty$
```

Fig. 5. General form of the tile motion-planning algorithm.

We consider a vehicle model that is described as follows. Let $(x,y,\theta) \in \mathcal{C}$ denote the position coordinates of the vehicle in a prespecified Cartesian axis system, and let ψ denote any additional state variables that are required to describe the state of the vehicle. We assume that $\psi \in \Psi$, where Ψ is an n-dimensional smooth manifold. The state of the vehicle is, thus, described by $\xi := (x,y,\theta,\psi) \in \mathcal{D} = \mathcal{C} \times \Psi$. Let $U \in \mathbb{R}^m$ denote the set of admissible control values, and for $t > 0$, let $\mathcal{U}_{[t_1,t_2]}$ denote the set of piecewise continuous functions that are defined on the interval $[t_1,t_2]$ that takes values in U. We assume that the evolution of the vehicle state ξ over a given time interval [0, t_f] is described by the differential equation $\dot{\xi}(t) = f(\xi(t),u(t))$ for all $t \geq [0,t_\mathrm{f}]$, where $u \in \mathcal{U}_{[0,t_\mathrm{f}]}$ is an admissible control input, and f is sufficiently smooth to guarantee global existence and uniqueness of solutions. We denote by $\xi(\cdot;\xi_0,u)$ the state trajectory that is the unique solution to the preceding differential equation with the initial condition $\xi(0) = \xi_0$. Finally, we denote by $\mathsf{x}(\xi)$ the projection of a state ξ on $\mathbb{R}^2$.

We introduce a special state trajectory planner called the *tile motion planner* as follows. Consider two vertices $I, J \in V_H$ such that $(I,J) \in E_H$. We define the *tile* associated with the edge (I,J) as the sequence of cells associated with the tuple $([I]_1, [J]_1^{H+1})$ of vertices in $\mathcal{G}$. In what follows, we use the symbol τ to denote a tile, and we denote by (I^τ, J^τ) the edge in E_H associated with this tile. TILEPLAN is then defined as any algorithm that determines if a given tile may be feasibly traversed by the vehicle from a specific initial condition. The cost of traversal of a tile is the integral along the state trajectory of a prespecified incremental cost $\ell(\xi,u,t)$. The general form of TILEPLAN is given in Fig. 5; in Section V, we outline a specific implementation of TILEPLAN, when the workspace is decomposed into square cells.

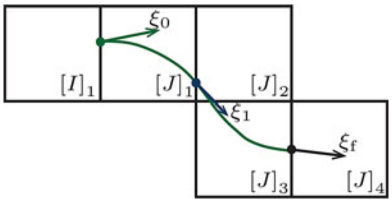


Fig. 6. Tile motion planning for square cells with $H = 3$.

Briefly, TILEPLAN determines if there exists a finite time t_f and a control $u \in \mathcal{U}_{[0,t_\mathrm{f}]}$ such that the corresponding vehicle state trajectory satisfies constraints (4) and (5). Constraint (4) states the requirement that the position components of the vehicle state trajectory remain within the tile τ at all times in the interval $(0,t_\mathrm{f})$, whereas constraint (5) states the requirement that the position components leave the tile in a finite time t_f. The algorithm returns the time t_1, which is required to traverse the first cell of the tile τ, the time history $u_{[0,t_1]}$ over the interval $[0,t_1]$ of the control input u that enables traversal of the tile, the vehicle state ξ_1 at the boundary between the first and second cells of the tile τ (see Fig. 6), and the cost Λ of traversal of the first cell.

Note that expression (4) does not depend explicitly on θ or ψ. This observation is consistent with our observation in Section I that a higher level planner typically operates on a discrete representation of the vehicle's workspace ($\mathbb{R}^2$), and not on its state space. In practice, θ or ψ may be subject to other constraints: For example, if $\psi = (\dot{x},\dot{y})$ represents the vehicle's velocity, then $\|\psi(t)\|$ may be subject to lower and upper bounds. However, these constraints are of no concern at the geometric planning level; their satisfaction will be ensured internally by TILEPLAN.

Suppose, for now, that a tile motion-planning algorithm satisfying the aforementioned requirements is available. Fig. 7 then describes the overall motion-planning framework. It consists of a geometric path planner that repeatedly invokes TILEPLAN to determine H-costs of histories. The proposed motion planner associates with each vertex $I \in V_H$ (in addition to the label $d(I)$ and backpointer $b(I)$ of the standard label-correcting algorithm) a vehicle state $\Xi(I) \in \mathcal{D}$, a time of traversal $\Theta(I) \in \mathbb{R}_+$, and an admissible control input $\Upsilon(I) \in \mathcal{U}_{[0,\Theta(I)]}$.

Informally, the proposed planner searches for a path in the graph $\mathcal{G}_H$ by traversing one edge during each iteration, while simultaneously propagating the vehicle state forward. As previously mentioned, the choice of an appropriate control input to propagate the vehicle state is left to TILEPLAN.

The proposed planner produces a path $\Pi^* = (J_0,\ldots,J_P)$ in $\mathcal{G}_H$, where $[J_0]_1 = i_\mathrm{S}$, and $[J_P]_{H+1} = i_\mathrm{G}$. As discussed previously, Π^* corresponds to a sequence of cells, and the control input to traverse this sequence of cells is given by

$$u(t) := \Upsilon(J_k),\quad t \in \left[\sum_{m=1}^{k-1}\Theta(J_m),\ \sum_{m=1}^{k-1}\Theta(J_m) + \Theta(J_k)\right) \tag{8}$$

for each $k = 1,\ldots,P$.

In the Appendix, we show that, with increasing H, the costs of trajectories that result from the proposed motion planner are

```
Algorithm 3: Overall Motion Planning Framework
Input: $I_S$, $\xi_0$,    Output: $d, b, \Xi, \Theta, \Upsilon$

procedure INITIALIZE($I_S, \xi_0$)
 1: $\mathcal{P} \leftarrow \{I_S\}$, $d(I_S) \leftarrow 0$;
 2: for all $I \in V_H \backslash \{I_S\}$ do
 3:     $d(I) = \infty$;
 4: $\Xi(I) = \xi_0$, $\Theta(I) = 0$

procedure MAIN
 1: INITIALIZE($I_S, \xi_0$)
 2: while $\mathcal{P} \neq \varnothing$ do
 3:     $I \leftarrow$ REMOVE($\mathcal{P}$)
 4:     for all $J \in V_H$ such that $(I, J) \in E_H$ do
 5:         $\tau \leftarrow$ TILE($[I]_1, [J]_1^{H+1}$)
 6:         $(t_1, u_{[0,t_1]}, \xi_1, \Lambda) \leftarrow$ TILEPLAN($\mathcal{R}, \Xi(I)$)
 7:         if $d(I) + \Lambda < d(J)$ then
 8:             $d(J) \leftarrow d(I) + \Lambda$
 9:             $b(J) \leftarrow I$, $\Xi(J) \leftarrow \xi_1$
10:             $\Theta(J) \leftarrow t_1$, $\Upsilon(J) \leftarrow u_{[0,t_1]}$
11:             INSERT($\mathcal{P}, J$)
```

Fig. 7. Pseudocode for the overall motion planner.

nonincreasing. An informal interpretation of this result is that as H is increased, the proposed motion planner erroneously rejects fewer admissible paths in $\mathcal{G}$ as infeasible. This result guides the selection of the value of H to implement the proposed motion planner, in that it assures benefits (in terms of optimality of the resultant trajectory) in return for expending computational resources to use larger values of H.

V. Tile Motion-Planning Implementations

The low-level trajectory planner TILEPLAN plays a crucial role in the proposed motion-planning framework. In this section, we present two specific implementations of TILEPLAN for nonholonomic vehicles.

The implementation of TILEPLAN is difficult mainly because (4) imposes a nonconvex constraint on the state trajectory. To alleviate this difficulty, we take advantage of the fact that each cell in the sequence of cells associated with a tile is a convex region, using the idea of *effective target sets*, first introduced in [53]. As described next, effective target sets enable the use of MPC-based techniques to implement TILEPLAN by transforming the constraint in (4) to a convex constraint that is defined over a single cell (see [52] for the details of such an implementation of TILEPLAN).

Let τ be a tile that corresponds to the edge $(I, J) \in E_H$. We define a sequence $\{\mathcal{X}_k\}_{k=1}^{H+1}$ of subsets of the vehicle state space called the *effective target sets* as follows. Let

$$\mathcal{X}_H := (\mathsf{cell}([J]_H) \cap \mathsf{cell}([J]_{H+1})) \times [-\pi, \pi] \times \Psi.$$

For each $k = 1, \ldots, H-1$, we define the effective target set $\mathcal{X}_k$ as the set of all states $\xi_k \in \mathcal{D}$ such that

$$\mathsf{x}(\xi_k) \in \mathsf{cell}([J]_k) \cap \mathsf{cell}([J]_{k+1}) \qquad (9)$$

and such that there exists $t_{k+1} \in \mathbb{R}_+$ and an admissible control input $u_{k+1} \in \mathcal{U}_{[t_k, t_{k+1}]}$ such that the state trajectory $\xi(\cdot; \xi_k, u_{k+1})$ satisfies

$$\mathsf{x}(\xi(t; \xi_k, u_{k+1})) \in \mathsf{cell}([J]_{k+1}), \ t \in (0, t_{k+1}) \qquad (10)$$

$$\xi(t_{k+1}; \xi_k, u_{k+1}) \in \mathcal{X}_{k+1}. \qquad (11)$$

The preceding definition of effective target sets allows a simplification of the tile motion-planning problem as follows. Suppose there exist a time t_1 and a control $u_1 \in \mathcal{U}_{[0,t_1]}$ such that the resultant state trajectory $\xi(\cdot; \xi_0, u_1)$ satisfies

$$\mathsf{x}(\xi(t; \xi_0, u_1)) \in \mathsf{cell}([J]_1), \quad t \in (0, t_1) \qquad (12)$$

$$\xi_1 := \xi(t_1; \xi_0, u_1) \in \mathcal{X}_1. \qquad (13)$$

Note that, due to (9), conditions (12)–(13) imply the satisfaction of (4)–(5) for $H = 1$. Next, because $\xi_1 \in \mathcal{X}_1$, it follows by (10)–(11) that there exists $t_2 \in \mathbb{R}_+$ and an admissible control input $u_2 \in \mathcal{U}_{[t_1, t_2]}$ such that

$$\mathsf{x}(\xi(t; \xi_1, u_2)) \in \mathsf{cell}([J]_2), \quad t \in (0, t_2)$$

$$\xi(t_2; \xi_1, u_2) \in \mathcal{X}_2.$$

In other words, the admissible control input u_{1-2} which is defined as the concatenation of the inputs u_1 and u_2 by

$$u_{1-2}(t) := \begin{cases} u_1(t), & t \in [0, t_1) \\ u_2(t), & t \in [t_1, T_2] \end{cases}$$

$$\text{where} \quad T_2 := t_1 + t_2$$

enables the vehicle's traversal through the cells corresponding to the vertices $[J]_1$ and $[J]_2$, i.e.,

$$\mathsf{x}(\xi(t; \xi_0, u_{1-2})) \in \mathsf{cell}([J]_1) \cup \mathsf{cell}([J]_2), \quad t \in (0, T_2) \quad (14)$$

$$\xi(T_2; \xi_0, u_{1-2}) \in \mathcal{X}_2. \qquad (15)$$

Note that, due to (9), conditions (14)–(15) imply the satisfaction of (4)–(5) for $H = 2$. Continuing recursively the preceding arguments, it follows that, for each $H \geq 2$, there exist $t_{k+1} \in \mathbb{R}_+$ and admissible inputs, $u_{k+1} \in \mathcal{U}_{[t_k, t_{k+1}]}$ for $k = 1, \ldots, H-1$, such that the admissible input u defined by

$$u(t) := \begin{cases} u_1(t), & t \in [0, T_1) \\ u_2(t), & t \in [T_1, T_2) \\ \vdots & \vdots \\ u_H(t), & t \in [T_{H-1}, T_H] \end{cases}$$

$$\text{where} \quad T_k := \sum_{m=1}^{k} t_m \qquad (16)$$

solves the tile motion-planning problem.

Thus, if the effective target sets $\mathcal{X}_k$, the corresponding times of traversal t_{k+1}, and the control inputs u_{k+1} in (16) are known for each $k = 1, \ldots, H-1$, then the tile motion-planning problem is equivalent to the problem to find u_1 and t_1 as described earlier. Crucially, (12) constrains the position components of the state trajectory to lie within a convex set. Furthermore, we may replace $\mathcal{X}_1$ in (13) by an interior convex approximating set $\tilde{\mathcal{X}}_1 \subset \mathcal{X}_1$, thus transforming the tile motion-planning problem into the problem to find u_1 and t_1 subject to convex constraints.

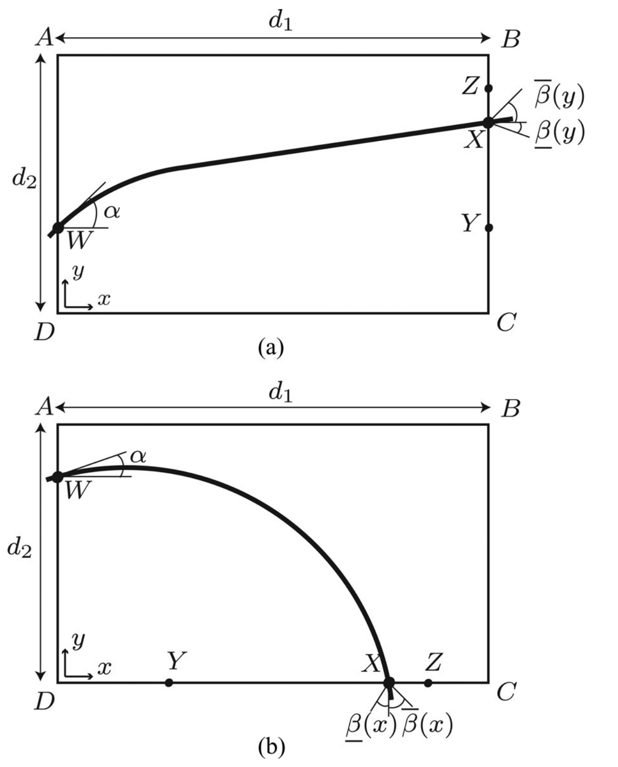


Fig. 8. Setup for Problems 8 and 9, which is used in the computation of effective target configuration sets. (a) Type 1 admissible path. (b) Type 2 admissible path.

A. Computation of Effective Target Sets

Guidelines to construct the effective target sets are provided in [53]. However, specific constructions may be tailored to the specific dynamical model. In light of the fact that the vehicle state includes the configuration (x, y, θ), we consider first the computation of the intersections of the effective target sets with the configuration space $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. To this end, we define the *effective target configuration sets* by $\mathcal{C}_k := \mathcal{X}_k \cap \mathcal{C}$ and in what follows, we outline a geometric scheme to compute the sets $\mathcal{C}_k$.

Assumption 5: The geometric curves in the plane that can be feasibly traversed by the vehicle satisfy a local upper bound on their curvatures.

We will comment on the validity of Assumption 5 in Section VI. Here, we use this assumption to compute the sets $\mathcal{C}_k$ by solving the following problems (namely, Problems 8 and 9) in plane geometry.

Let $ABCD$ be a rectangle. We attach a Cartesian axes system as shown in Fig. 8. Let the dimensions of the rectangle be d_1 and d_2, and let $r > 0$ be fixed.

Definition 6: Let $\underline{\beta}(x), \overline{\beta}(x)$, $x \in [0, d_2]$, be functions such that $-\frac{\pi}{2} \leq \underline{\beta}(x) \leq \overline{\beta}(x) \leq \frac{\pi}{2}$. Let $Y = (d_1, y), Z = (d_1, z)$ be points on the segment BC with $y \leq z$. A path Π is a *Type 1 admissible path* if it satisfies the following properties.

1) (Curvature boundedness): The curvature at any point on Π is at most r^{-1}.
2) (Containment): Π intersects the segment BC in exactly one point $X = (d_1, x)$ such that $x \in [y, z]$, and it may intersect segment AB and/or segment CD in at most one point each.
3) (Terminal orientation): $\Pi'(X) \in \left[\underline{\beta}(x), \overline{\beta}(x)\right]$.

A Type 2 admissible path is defined analogously for traversal across adjacent edges.

Definition 7: Let $\underline{\beta}(x), \overline{\beta}(x)$, $x \in [0, d_1]$, be functions such that $-\frac{\pi}{2} \leq \underline{\beta}(x) \leq \overline{\beta}(x) \leq \frac{\pi}{2}$. Let $Y = (y, 0), Z = (z, 0)$ be points on the segment CD with $y \leq z$. A path Π is a *Type 2 admissible path* if it satisfies the following properties.

1) (Curvature boundedness): The curvature at any point on Π is at most r^{-1}.
2) (Containment): Π intersects the segment CD in exactly one point $X = (x, 0)$ such that $x \in [y, z]$, and it may intersect segment AB and/or segment BC in at most one point each.
3) (Terminal orientation): $\Pi'(X) + \frac{\pi}{2} \in \left[\underline{\beta}(x), \overline{\beta}(x)\right]$.

We state two geometric problems as follows. Let $\underline{\beta}, \overline{\beta}, Y$, and Z be as in the preceding definitions. Let $W = (0, w)$ and $r > 0$ be fixed.

Problem 8 (Traversal Across Parallel Edges): Find bounds $\underline{\alpha}, \overline{\alpha}$ such that for all $\alpha \in [\underline{\alpha}, \overline{\alpha}]$, there exists a Type 1 admissible path with initial configuration (W, α).

Problem 9 (Traversal Across Adjacent Edges): Find bounds $\underline{\alpha}, \overline{\alpha}$ such that for all $\alpha \in [\underline{\alpha}, \overline{\alpha}]$, there exists a Type 2 admissible path with initial configuration (W, α).

Problems 8 and 9 appear in the recursive computation of effective target configurations as follows. Suppose that the effective target configuration set $\mathcal{C}_{k+1}$ is known, for $k \in \{1, \ldots, H-1\}$. Then, we may express $\mathcal{C}_{k+1}$ as the product set of a line segment on the boundary between cells $\mathsf{cell}([J]_{k+1})$ and $\mathsf{cell}([J]_{k+2})$ and a set of allowable orientations on this line segment. In other words, we may express $\mathcal{C}_{k+1}$ in terms of the points Y, Z and the functions $\underline{\beta}, \overline{\beta}$ that are used in Definitions 1 and 2. We may then solve Problem 8 or 9, as applicable for the cell $\mathsf{cell}([J]_{k+1})$, for each point on the line segment that forms the boundary between cells $\mathsf{cell}([J]_k)$ and $\mathsf{cell}([J]_{k+1})$ and obtain allowable orientations for each point on this line segment. The product set of these allowable orientations and this line segment is precisely the set $\mathcal{C}_k$. The effective target configuration sets may, thus, be computed recursively by repeatedly solving Problems 8 and 9 as applicable for each cell, with $\mathcal{C}_H := (\mathsf{cell}([J]_H) \cap \mathsf{cell}([J]_{H+1})) \times \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

The solutions to Problems 8 and 9 are based on detailed plane geometrical analysis, and are beyond the scope of this paper. We refer the reader to [52] for the analysis that is involved in the solution of Problems 8 and 9. Here, we present a procedure that uses the solutions to Problems 8 and 9 to recursively compute the effective target configuration sets. Note that the sequence of cells associated with a tile is, in general, a *rectangular channel*, that is, a finite sequence $\{R_n\}_{n=1}^{C}$ of disjoint rectangles of arbitrary dimensions such that every pair of successive rectangles has a common edge.

We attach a coordinate axes system to each rectangle R_n in a manner consistent with the axes system that is used in the statement of Problems 8 and 9 (see Fig. 10). The dimensions of each rectangle along the x and y axes are denoted, respectively, as $d_{n,1}$ and $d_{n,2}$.

Algorithm 4: Effective Configuration Sets

```
1: $\overline{\alpha}_{C+1}(q) \leftarrow \frac{\pi}{2}, \underline{\alpha}_{C+1}(q) = -\frac{\pi}{2}$, $q \in [0, d_{C,2}]$ if $R_C$
   involves traversal across parallel edges, or otherwise, for
   $q \in [0, d_{C,1}]$ if $R_C$ involves traversal across adjacent edges
2: $\varrho_{C+1} \leftarrow 0$
3: for $n = C$ to 1 do
4:     if $\varrho_n + \varrho_{n+1}$ even then
5:         $\underline{\beta}_n \leftarrow \underline{\alpha}_{n+1}$, $\overline{\beta}_n \leftarrow \overline{\alpha}_{n+1}$
6:     else
7:         $\underline{\beta}_n(q) \leftarrow -\overline{\alpha}_{n+1}(y_n - (q - v_{n+1}))$, for $q \in [y_n, z_n]$
8:         $\overline{\beta}_n(q) \leftarrow -\underline{\alpha}_{n+1}(y_n - (q - v_{n+1}))$, for $q \in [y_n, z_n]$
9:     $\underline{\alpha}_n(q)$, $\overline{\alpha}_n(q) \leftarrow$ solution of Problem 8 or Problem 9,
       as applicable to $R_n$ for $q \in [u_n, v_n]$
```

Fig. 9. Recursive computation of effective target configuration sets.

For each rectangle R_n, we refer to the segments that are formed by the intersections $R_{n-1} \cap R_n$ and $R_n \cap R_{n+1}$, respectively, as the *entry and exit segments*. For the rectangle R_1 (resp. rectangle R_C), the entry segment (resp. exit segment) is specified arbitrarily. We denote the endpoints of the entry segment by U_n and V_n, and the endpoints of the exit segment by Y_n and Z_n. We specify the coordinates of the points U_n, V_n, Y_n, Z_n, by the corresponding lower case letters, i.e., $V_n = (0, v_n)$, etc.

For every point $Q = (q, 0)$ (or $Q = (d_{n,1}, q)$, as applicable), with $q \in [y_n, z_n]$, on the segment $Y_n Z_n$, we denote by $\underline{\beta}_n(q)$ and $\overline{\beta}_n(q)$, respectively, the lower and upper bounds on orientation of the effective target configuration sets. Similarly, for every point $P = (0, p)$, $p \in [u_n, v_n]$, on the segment $U_n V_n$, we denote by $\underline{\alpha}_n(p)$ and $\overline{\alpha}_n(p)$, respectively, the upper and lower bounds that result from the solution of Problem 8 (or Problem 9, as applicable). Note that the angles $\underline{\alpha}_n(\cdot), \overline{\alpha}_n(\cdot), \underline{\beta}_n(\cdot)$, and $\overline{\beta}_n(\cdot)$ are all measured with respect to the local coordinate axes system attached to R_n. Finally, we denote by ϱ_n the number of reflection operations involved in the rigid transformations required to align the entry and exit segments of R_n to the segments AD and BC, respectively, for traversal across parallel edges, or to segments AD and CD, respectively, for traversal across adjacent edges.

The recursive analysis to determine the existence of curvature-bounded paths in rectangular channels is described using the aforesaid notation in Fig. 9. To better explain the procedure in Fig. 9, we illustrate its execution by an example.

Example 10: Let $\bar{\mathcal{R}}^4 = \{R_n\}_{n=1}^4$ be a rectangular channel that consists of four rectangles, as shown in Fig. 10 , and let $r_n > 0$, $n = 1, \ldots, 4$ be given. The points U_n, V_n, $n = 1, \ldots, 4$, and the points Y_4, Z_4 are shown in Fig. 10. We note that $Y_1 = U_2$, $Z_1 = V_2$, $Y_2 = V_3$, $Z_2 = U_3$, $Y_3 = V_4$, and $Z_3 = U_4$.

Following the procedure in Fig. 9, we note that the last rectangle R_4 involves traversal across parallel edges, and we initialize $\overline{\alpha}_5$ and $\underline{\alpha}_5$ as

$$\overline{\alpha}_5(q) = \frac{\pi}{2}, \quad \underline{\alpha}_5(q) = -\frac{\pi}{2}, \quad q \in [0, d_{4,2}] .$$

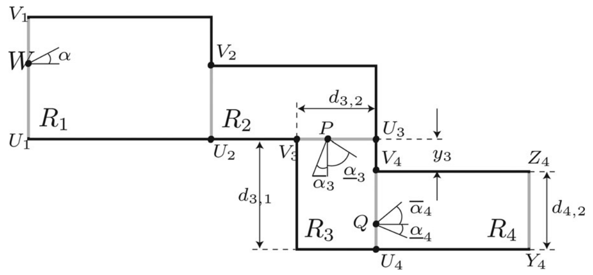


Fig. 10. Computation of effective target configuration sets.

Next, we execute Line 4 of the algorithm, and we note that the entry and exit segments of rectangle R_4 are aligned with segments AD and BC, respectively, of Fig. 8. Thus, the total number of reflections occurring in the transformations required for R_4 and (the fictitious rectangle) R_5 is zero, and we set

$$\overline{\beta}_4 = \overline{\alpha}_5 = \frac{\pi}{2}, \quad \underline{\beta}_4 = \underline{\alpha}_5 = -\frac{\pi}{2}.$$

To execute Line 9, we solve Problem 8 for each point $Q = (0, q)$, $q \in [0, d_{4,2}]$, on the segment U_4V_4, and we obtain the values that are taken by the functions $q \mapsto \underline{\alpha}_4(q)$ and $q \mapsto \overline{\alpha}_4(q)$.

Now we repeat Line 4 for rectangle R_3. Rectangle R_3 involves traversal across adjacent edges, and the entry and exit segments of R_3 may be aligned with segments AD and DC of Fig. 8 after a reflection about an axis parallel to the segment U_4V_4, followed by a rotation through $\frac{\pi}{2}$ rad. Thus, the total number of reflections occurring in the transformations required for R_3 and R_4 is 1 (odd), and we set

$$\overline{\beta}_3(q) = -\underline{\alpha}_4(y_3 - (q - v_4)), \quad q \in [y_3, z_3]$$
$$\underline{\beta}_3(q) = -\overline{\alpha}_4(z_3 - (q - u_4))$$

where $z_3 = d_{3,1}$, $y_3 = \ell(U_3V_4)$, $v_4 = d_{4,2}$, and $u_4 = 0$ (see Fig. 10). To execute Line 9, we solve Problem 9 for each point $P = (0, p)$, $p \in [0, d_{3,2}]$, on the segment U_3V_3 to obtain values that are taken by the functions $p \mapsto \underline{\alpha}_3(p)$ and $p \mapsto \overline{\alpha}_3(p)$. Proceeding further in a similar manner, we may obtain the values that are taken by the functions $\underline{\beta}_2, \overline{\beta}_2$, and by the functions $\underline{\beta}_1, \overline{\beta}_1$. As discussed previously, the effective target configuration sets $\mathcal{C}_n$ may then be expressed in terms of the functions $\underline{\beta}_n$, and $\overline{\beta}_n$, for each $n = 1, \ldots, 4$.

B. TILEPLAN *for the Dubins Car*

In this section, we discuss an implementation of TILEPLAN for the Dubins car kinematic model that is described by

$$\dot{x}(t) = v \cos\theta(t), \qquad \dot{y}(t) = v \sin\theta(t), \qquad \dot{\theta}(t) = u(t)$$

where x, y, and θ are, respectively, the position coordinates and the orientation of the vehicle with respect to a prespecified inertial axes system, $v > 0$ is the (fixed) forward speed of the vehicle, and u is the steering control input. The set of admissible control inputs is $U := [-1/r, 1/r]$, for a prespecified $r > 0$. As it will be shown in Section VI, the upper bound on the curvature of feasible geometric paths is $\kappa_{\max} = (rv)^{-1}$.

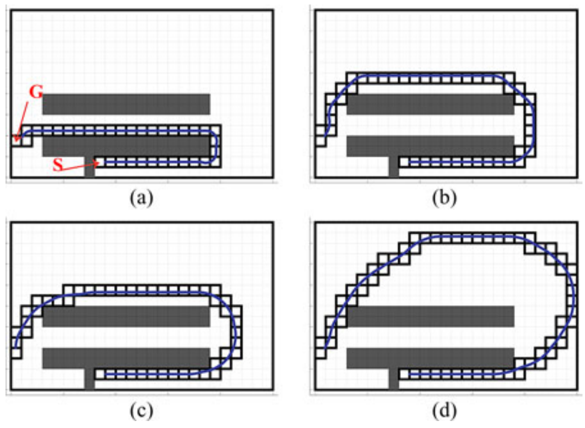


Fig. 11. Making U-turns with different curvature bounds. The curvature bound in each case is r^{-1}. Here, $H = 3$. (a) $r = 1$. (b) $r = 2$. (c) $r = 3.5$. (d) $r = 4.5$.

Note that when the state space $\mathcal{D}$ is the same as the configuration space $\mathcal{C}$, the effective target sets coincide with the effective target configurations sets, which can be computed via pure geometric analysis as described in the previous section. In addition, the feasible paths that are specified in TILEPLAN can also be constructed geometrically, as discussed in [54], thus enabling a solution to the motivating example of Section II.

Fig. 11 shows results of the simulations of the proposed algorithm for a problem similar to the motivating example of Section II. Note, in particular, that different channels are obtained for different bounds on the curvature, whereas *any* cost function that is defined on the edges E of the cell decomposition graph $\mathcal{G}$ will result in the channel shown in Fig. 11(a). For this example, the cost of traversal of a tile was chosen as the time of traversal, i.e., $\ell(\xi, u, t) := 1$ in (7).

VI. RESULTS AND COMPARATIVE DISCUSSIONS

In this section, we present the results of implementation of the proposed motion-planning framework for general vehicle models, and we compare our results with those obtained using randomized sampling-based (RRT-based) motion planners. As discussed in Section I-A, randomized sampling-based algorithms that are based on RRTs [34] represent the state of the art in kinodynamic motion planning.

We implemented the fringe as a list which is sorted by the sum of the current label and a heuristic; specifically, we used the Manhattan distance to the goal cell as a heuristic.[4] We implemented TILEPLAN using a trajectory generation scheme that is based on model predictive control, similar to that reported in [55]. Before discussing the results, we comment on Assumption 5, which was used for geometrically computing the effective target configuration sets.

The expression for the local curvature of the geometric path that corresponds to feasible state trajectories is given by

$$\kappa(t) = |\dot{\theta}(t)/v(t)|. \tag{17}$$

[4]One may also envision heuristics that are defined on vertices of $\mathcal{G}_H$ instead of vertices of $\mathcal{G}$. For instance, one may consider a heuristic that is based on coarse geometric considerations of the channels associated with the vertices of $\mathcal{G}_H$.

An upper bound for the local curvature may then be computed which is based on the specific vehicle model. For the Dubins car model for example, $|\dot{\theta}| = |u| \leq 1/r$, and by (17) it follows that $\kappa(t) \leq (rv)^{-1}$ for all $t \geq 0$, i.e., the upper bound on the curvature of feasible geometric paths is $\kappa_{\max} = (rv)^{-1}$. Similarly, for the dynamical model in Section VI-A, note that $|\dot{\theta}(t)| \leq |u_2(t)| \leq f_{\mathrm{r}}^{\max}/v(t)$. It follows by (17) that $\kappa(t) \leq f_{\mathrm{r}}^{\max}/v^2(t)$. Thus, an upper bound on the curvature is $f_{\mathrm{r}}^{\max}/v_{\max}^2$; however, for each tile one may compute a local bound $\bar{v}$ on the speed valid for traversal across the tile, and use the less conservative upper bound $f_{\mathrm{r}}^{\max}/(\min\{\bar{v}, v_{\max}\})^2$ to implement TILEPLAN.

A. Optimality of Resultant Trajectories

We consider a vehicle dynamical model described by

$$\begin{aligned} \dot{x}(t) &= v(t)\cos\theta(t), & \dot{y}(t) &= v(t)\sin\theta(t) \\ \dot{\theta}(t) &= u_2(t), & \dot{v}(t) &= u_1(t), \end{aligned}$$

where $v > 0$ is the forward speed of the vehicle, u_1 is the acceleration input, and u_2 is the steering input. The speed v is constrained to lie within prespecified bounds $v_{\min}$ and $v_{\max}$; these bounds may be different for different regions of the workspace. The set of admissible control inputs is

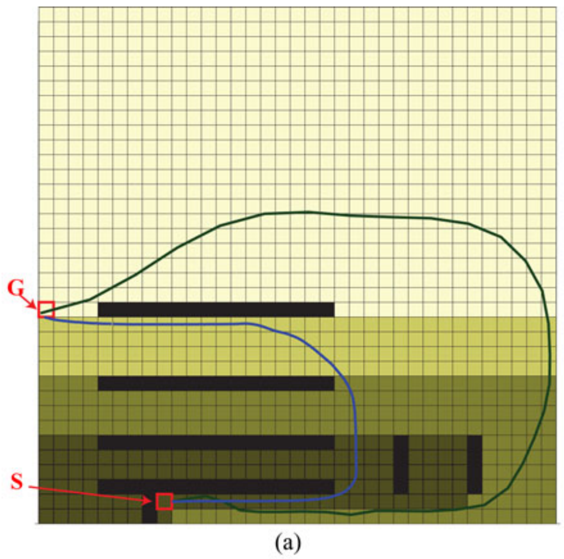
$$U := \left\{ (a, \omega) : \left(\frac{v\omega}{f_{\mathrm{r}}^{\max}} \right)^2 + \left(\frac{a}{f_{\mathrm{t}}^{\max}} \right)^2 \leq 1 \right\} \tag{18}$$

where $f_r^{\max}$ and $f_t^{\max}$ are prespecified. The input constraint defined by (18) is an example of a "friction ellipse" constraint that models the limited tire frictional forces that are available for acceleration and steering of the vehicle.

Fig. 12(a) shows the first of two environments used in the numerical examples. This environment consists of "lanes" separated by obstacles (black regions), with a different upper bound on the allowable speed of the vehicle (lighter areas represent higher upper bounds). The "friction ellipse" parameters were fixed at $f_{\mathrm{r}}^{\max} = 1$, $f_{\mathrm{t}}^{\max} = 0.25$ over the entire environment. The initial and goal cells are marked in Fig. 12(a). As before, the objective is to find a minimum time trajectory from the initial cell to the goal cell. We compared the proposed motion planner with the following two RRT-based planners: 1) the standard RRT-based planner as reported in [34] and 2) the T-RRT planner that has recently reported in [56]. The T-RRT planner finds low-cost trajectories with respect to a prespecified state space[5] cost map. Note that the minimum-time criterion cannot be expressed as a state space cost map; therefore, we execute the T-RRT planner with the objective "travel as fast as possible," which is immediately defined by the state space cost map $c(x, y, \theta, v) = v$.

Linear interpolation between two states does not, in general, correspond to a feasible state trajectory. Hence, to extend known states toward randomly selected new states, the RRT-based planners were programmed to randomly select an input vector from the set of admissible inputs and integrate the vehicle model for

[5]In [56], the authors deal with a configuration space cost map, but their approach extends easily to state spaces.



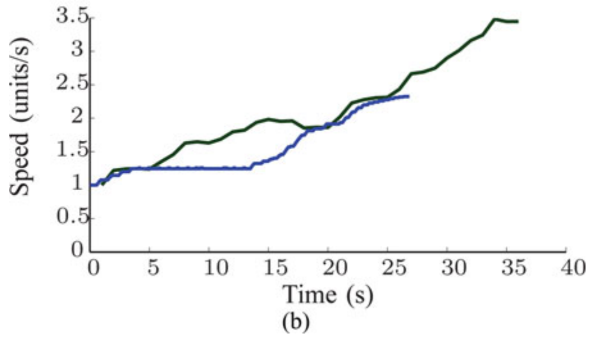


Fig. 12. "Lanes" environment. Black colored regions denote obstacles; areas with other colors represent different speed limits: $v_{\max} = 1.25$ units/s for the darkest area, $v_{\max} = 2$ units/s, $v_{\max} = 2.5$ units/s, and $v_{\max} = 3.5$ units/s for progressively lighter areas. The dark green curve denotes the geometric path that corresponds to a sample trajectory returned by the T-RRT algorithm, with $\delta = 1$ s. The blue curve denotes the path that corresponds to the trajectory returned by the proposed approach, with $H = 6$. (a) Geometric paths. (b) Speed profiles.

a fixed time δ, as recommended in [34]. For the "lanes" environment, we used three different values of δ, namely, $\delta = 0.5$ s, $\delta = 1$ s, and $\delta = 1.5$ s, and we conducted 30 trials of both algorithms (standard RRT and T-RRT) for each value of δ. For comparison, we executed the proposed algorithm on the same environment with three different values of H, namely, $H = 4$, $H = 5$, and $H = 6$, with $L = 10$ in each case.

Fig. 14(a) shows comparative data for the trajectory costs (i.e., time of traversal) that result from the simulations described earlier. The proposed motion planner returned trajectories with almost identical costs for each H. In particular, the trajectory cost corresponding to $H = 6$ was 26.626 s. On the other hand, both the standard RRT and T-RRT planners returned, on an average, significantly costlier trajectories. For instance, the trajectory costs that are returned by the standard RRT planner with $\delta = 1$ were in the best case 24% higher, on an average 78% higher, and in the worst case 181% higher. Similarly, the trajectory costs that are returned by the T-RRT planner with $\delta = 1$ were in the best case 8.9% higher, on an average 29% higher, and in the worst case 46% higher.

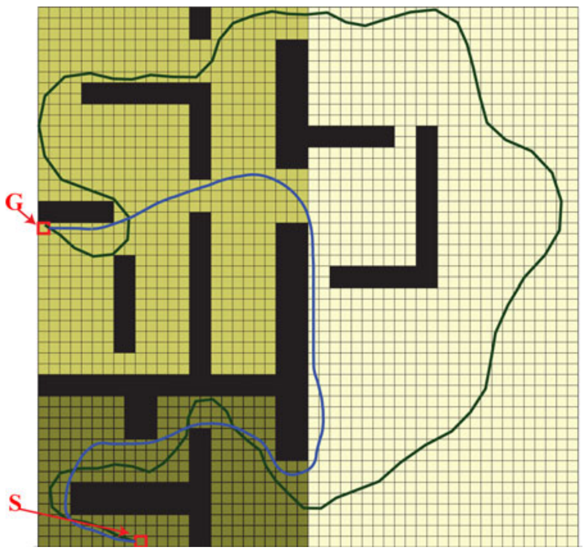


Fig. 13. "Maze" environment. Black colored regions denote obstacles; areas with other colors represent different speed limits: $v_{\max} = 1.25$ units/s for the darkest area, $v_{\max} = 2$ units/s, and $v_{\max} = 2.25$ units/s for progressively lighter areas. The dark green curve denotes the geometric path that corresponds to a sample trajectory returned by the RRT-based planner, with $\delta = 1.5$ s. The blue curve denotes the geometric path that corresponds to the trajectory returned by the proposed approach, with $H = 5$.

Fig. 12(a) shows the geometric path that corresponds to the trajectory returned by the proposed planner with $H = 6$ (blue curve) in comparison with the geometric path that corresponds to a trajectory returned by the T-RRT planner in one of the 30 trials with $\delta = 1$ (green curve). Fig. 12(b) shows the speed profiles that correspond to these two trajectories. This example illustrates that the "travel as fast as possible" objective is not always a practically acceptable alternative to the minimum-time criterion: Fig. 12(b) shows that the vehicle achieves higher speeds along the T-RRT trajectory but the travel time is 35.2% higher than the trajectory that is found by the proposed planner. This result is a consequence of the input constraint (18), which forces the vehicle to traverse paths of lower curvature at higher speeds, thus producing longer geometric paths.

Fig. 13 shows the second, maze-like environment that is used for our comparative analysis. As before, different upper bounds on the speed were assigned to different areas in the environment, and the friction ellipse parameters were fixed at $f_{\mathrm{r}}^{\max} = 1$ and $f_{\mathrm{t}}^{\max} = 0.25$ over the entire environment. As before, the objective is to find a minimum-time trajectory from the initial cell to the goal cell. We compared the proposed motion planner with the standard RRT planner alone, because the T-RRT planner was found to be impractically slow for this case. As shown in Fig. 13, the environment has a narrow "short-cut" between the initial cell and the goal cell.

Fig. 14(a) shows comparative data for the trajectory costs for this maze-like environment. The proposed motion planner returned trajectories with almost identical costs for each H; in particular, the trajectory cost that corresponds to $H = 5$ was 56.23 s. The trajectory costs returned by the standard RRT planner were significantly higher, mainly because it failed to traverse the aforementioned "short-cut" on several occasions,

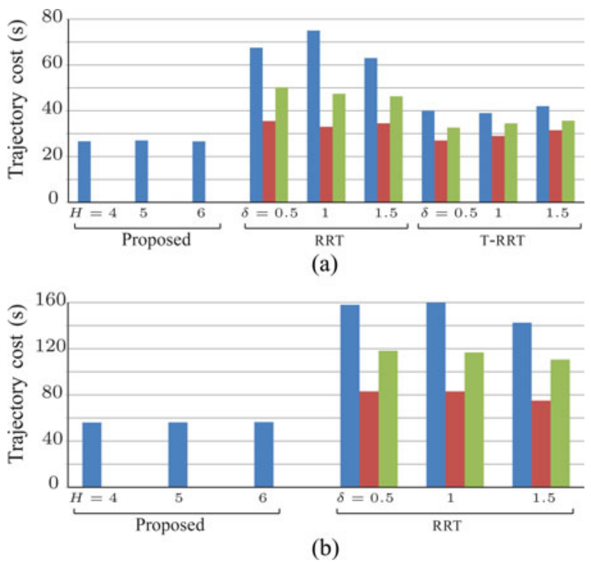


Fig. 14. Comparison of trajectory costs. For the RRT and T-RRT data, the blue (left), red (middle), and green (right) bars represent, respectively, the maximum, the minimum, and the average values over 30 trials. (a) Data for the "lanes" environment in Fig. 12(a). (b) Data for the maze-like environment in Fig. 13.

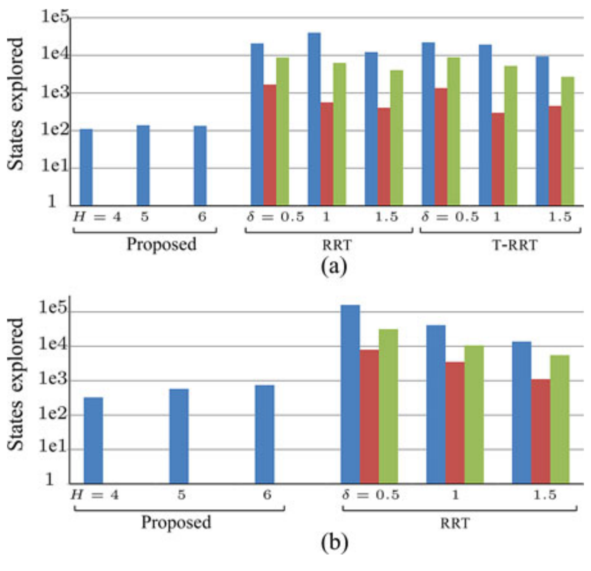


Fig. 15. Comparison of number of states explored: for the RRT and T-RRT data, the blue (left), red (middle), and green (right) bars represent, respectively, the maximum, the minimum, and the average values over 30 trials. (a) Data for the "lanes" environment in Fig. 12(a). (b) Data for the maze-like environment in Fig. 13.

TABLE III
EXECUTION TIMES FOR THE PROPOSED PLANNER

	$H=4$	$H=5$	$H=6$
"Lanes"	31.89 s	52.02 s	55.12 s
Maze	129.6 s	303.7 s	378.2 s

as illustrated in Fig. 13. For instance, the trajectory costs that are returned by the standard RRT planner with $\delta = 1$ were in the best case 48% higher, on an average 107% higher, and in the worst case 185% higher. Clearly, the average costs of trajectories that are returned by RRT-based planners may be further worsened in environments where the differences between the costs of trajectories corresponding to "short-cuts" and the costs of alternative trajectories are larger.

B. *Performance of the Proposed Motion Planner*

Table III presents the execution times of the simulations of the proposed planner for the examples that are discussed in the previous section. The simulations were implemented in the MATLAB programming language; implementations in lower level languages will execute much faster.

Fig. 15(a) shows on a logarithmic scale the number of states explored by each of the algorithms discussed in the previous section for the "lanes" environment. Fig. 15(b) shows similar data for the maze-like environment in Fig. 13. In both cases, the number of states explored by the RRT-based planners was higher by at least an order of magnitude.

It should be noted, however, that the number of states explored is not a direct indicator of the computation time of either of the approaches. This is because the time required for the RRT-based planners to explore a new state (including the nearest neighbor search and collision checking) is different from the execution time of the MPC-based TILEPLAN. In our simulations, we found that the time required to explore a new state in the RRT-based planners was approximately an order of magnitude lower than the time required to explore a new state in the proposed approach. A direct comparison of the execution times of these planners showed no conclusive evidence of the superiority of either planner over the other in that respect. However, as indicated in Fig. 15, it is expected that the proposed planner will be preferable in cases where the exploration of new states is expensive, due to, perhaps, complicated vehicle dynamics that will require a computationally expensive local planner.

C. *Further Discussion*

1) Comparisons With Randomized Sampling-Based Motion Planners: The exploration of the state space is difficult with standard RRT-based motion planners when the states and control inputs are coupled via complex, nonlinear differential equations [37], because linear interpolation between two states no longer corresponds, in general, to an admissible state trajectory. While [37] and similar earlier works focus on aiding the *efficiency* of sampling-based algorithms using a discrete search, we focus on the complementary aspect of *optimality* by ensuring that the result of a discrete shortest path search remains compatible with the vehicle dynamics.

In addition to the benefits of the proposed planner over randomized sampling-based planners in terms of optimality, the proposed planner also offers the benefit of a clear distinction between the discrete and continuous layers of motion planning. The idea of planning on the lifted graph, which is introduced in Section II, allows this distinction to be maintained, while providing guarantees of consistency between the two levels of planning. Consequently, changes to the discrete planning strategy and/or the (continuous) tile motion planning may be incorporated with relative ease. In this paper, we used the shortest

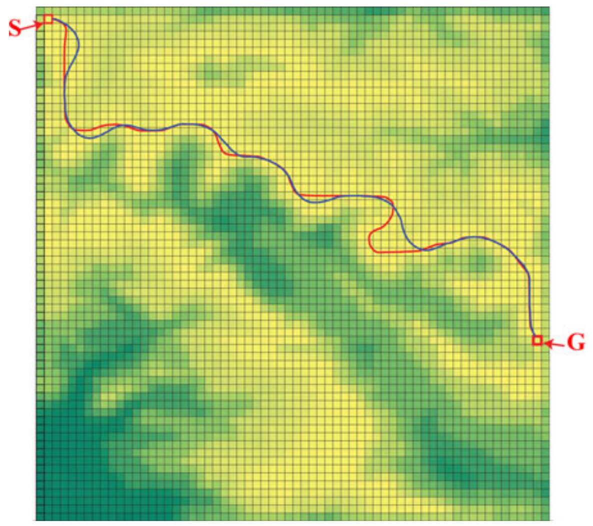


Fig. 16. Application of the proposed motion-planning framework to find "low-elevation" paths: Lighter areas represent more favorable regions of the workspace. The Dubins car kinematical model is used in TILEPLAN. The red curve represents a path with $r = 1$, whereas the blue curve represents a path with $r = 2$.

path search as a concrete, important example of a discrete search strategy; in the future, we envision extensions of the proposed planner where the discrete planner attempts to satisfy vehicular tasks that are specified as formulas of predicate or temporal logic [11] instead of solving a shortest path problem. On the other hand, complex vehicle dynamics can be easily incorporated in TILEPLAN without affecting the discrete planning strategy.

In the context of the shortest path problem alone, different trajectory quality criteria can be incorporated easily to meet different motion-planning objectives. For example, Fig. 16 shows the result of simulating the proposed planner by defining the H-cost as a weighted sum of the time of traversal and the terrain elevation. Consequently, the planner finds paths that traverse low-elevation portions of the terrain (lighter regions in Fig. 16), while ensuring kinematic feasibility guarantees of the resultant paths. The Dubins car model was used for this simulation; the two curves in Fig. 16 indicate the resultant paths for different curvature constraints. It should be noted that the problem to find low-cost trajectories with respect to configuration space cost maps using randomized sampling-based methods has been addressed in [56]. However, many important trajectory quality criteria such as time optimality (considered in the preceding section) and fuel optimality cannot be expressed as configuration space cost maps.

2) Comparisons With Recent RRT*-Based Motion Planners:* During the preparation of this study, we became aware of two recent motion-planning methods that seem to be competitive to our approach. In [57], an extension of RRT (the so-called RRT* algorithm) has been proposed to recover (asymptotically) optimality for RRT-based planners. RRT* improves incrementally the path quality, and it has been shown to result in asymptotically optimal paths. Similarly, the SyCLoP framework [37] may produce paths of better quality in comparison with the standard RRT algorithm.[6] In [37], a random exploration of the state space is biased using a discrete search that is based on, say, geometric decompositions of the environment. From the results of these two references, we expect that the path quality of the proposed framework will be comparable with that of the RRT* framework, with significant benefits in execution time. Similarly, it is expected that the execution time of the proposed algorithm will be comparable with that of the SyCLoP framework, with significant benefits in path quality. Of course, detailed numerical comparisons in terms of execution time and path quality between the proposed framework and the RRT* and SyCLoP frameworks, and for different vehicle dynamical models, are needed to confirm these observations.

3) Comparisons With Feedback-Based Motion Planners: An underlying assumption in the feedback-based motion-planning approach that is described in Section I-A is the complete controllability of the vehicle dynamical model in the presence of obstacles, i.e., the assumption that there exists a feasible, obstacle-free trajectory from any initial state to any goal state. In the context of mobile vehicles, complete controllability in the presence of obstacles is a strong assumption: Fixed-wing aircraft that moves in the horizontal plane do not satisfy this assumption; terrestrial vehicles constrained to move only forward, or high-speed vehicles for which stopping and reversing the direction of motion may not be desirable also do not satisfy this assumption. In contrast with the planners presented in [38]–[42], the proposed motion-planning framework does not assume complete controllability in the presence of obstacles.

When the complete controllability assumption is violated, the central tenet of the preceding feedback-based motion-planning schemes is no longer valid: Arbitrary sequences of cell transitions cannot *in principle* be guaranteed from arbitrary initial states. A simple example of a vehicle kinematic model that violates the complete controllability assumption in the presence of obstacles is the Dubins car model. For any given sequence of cell transitions in the workspace, there exists a set of initial states of the vehicle from which it is impossible for the vehicle to execute that sequence. The proposed framework does not require this assumption because the geometric planner ensures the feasibility of traversal of its resultant path (i.e., the sequence of cell transitions from the initial position to the goal) by computing an admissible control input, which is given in (8).

4) Applications to Shaped Robots: Although, for simplicity, in this paper we assumed a point-mass vehicle model, in practice it is important to incorporate the size of the vehicle. To do so, TILEPLAN may be modified such that condition (4) constrains the geometric path that is traversed by the vehicle to lie within a shrunken channel inside the tile (see Fig. 17). Crucially, the shrunken inner channel is also a rectangular channel; hence, the geometric analysis that is presented in Section V-A can be used to implement the modified TILEPLAN. The implementation of TILEPLAN for robots of finite size is, thus, a relatively straightforward extension of the work presented in Section V.

[6]Although in [37], the authors claim improvements only in the *efficiency* of motion planning.

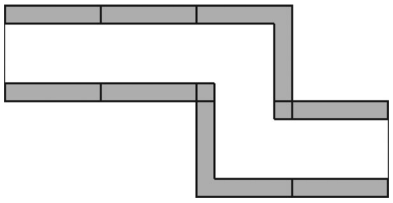


Fig. 17. Modification of cell traversal constraints in TILEPLAN to deal with shaped robots. Note that the inner channel is also a rectangular channel.

5) Connections With Differentially Flat Systems: The idea of attaching history-dependent costs while working solely with workspace cell decompositions hinges strongly on the ability to reconstruct the whole vehicle state trajectories from a path in the workspace. This is closely related to the idea of *differential flatness* [58]. The states and inputs of a differentially flat system can be expressed in terms of the so-called *flat outputs* and their derivatives. In this context, the proposed idea to attach history-dependent costs along the path may be viewed as a method to recover derivative (i.e., velocity) information about the vehicle from its time-parameterized workspace path.

For a Dubins vehicle, for instance, this derivative information allows the planner to calculate the vehicle orientation from $\theta = \tan^{-1}(\dot{y}/\dot{x})$. In other words, time-parameterized trajectories in the workspace (in this case, x and y coordinates) allow the recovery of the missing state variable (in this case θ). As a result of this observation, it is expected that the scope of the proposed motion-planning framework may be expanded significantly by investigating its application to differentially flat systems whose flat outputs coincide with the workspace of the vehicle (or robot).

6) Completeness of the Proposed Motion Planner: We conclude the discussion of the proposed motion-planning framework with a comment concerning its completeness.[7] The proposed motion planner is complete in the following sense: Because the number of cells is finite, there exists a finite $H \in \mathbb{N}$ such that the initial vertex $i_{\rm S} \in V$ and the goal vertex $i_{\rm G} \in V$ belong to a single history, i.e., a single edge $(I_{\rm S}, I_{\rm G})$ of the lifted graph $\mathcal{G}_H$. Then, assuming TILEPLAN is complete (i.e., TILEPLAN finds a path satisfying the specifications that are listed in Fig. 5 if one exists), the algorithm finds a control input required to traverse the tile corresponding to the edge $(I_{\rm S}, I_{\rm G})$, thus solving the motion-planning problem.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a hierarchical motion-planning framework that comprises a discrete path planner and a continuous trajectory planner. In light of the fact that obstacles are typically defined in a lower dimensional workspace, we emphasize the discretization of only the workspace, while allowing the low-level trajectory planner to apply local control algorithms specific to the vehicle model to find control inputs that enable the vehicle's desired motion.

The proposed framework involves a precise characterization of the interaction between the two levels of planning. Specifically, the high-level planner solves a special path optimization problem on a graph—finding a path in the cell decomposition graph that minimizes costs defined over multiple edge transitions (histories)—and the low-level trajectory planner determines the feasibility of traversal and the cost of traversal (if feasibility is ensured) of the sequence of cells that correspond to the given history of edge transitions. We provide an algorithm to solve the aforementioned path optimization problem exactly, along with a modification of the algorithm that executes faster, albeit at the expense of optimality of the resultant path. Using a four-state, two-input vehicle dynamical model, we compare the proposed motion planner with two RRT-based planners and provide numerical data that demonstrate the superiority of the proposed planner in terms of the quality of the resultant trajectories.

Future work will deal with the implementation of the proposed framework using multiresolution cell decompositions, and with the development of tile motion-planning algorithms for more realistic vehicle dynamic models.

APPENDIX

DEPENDENCE OF PATH OPTIMALITY ON H

We assume here that the proposed motion planner solves an H-cost shortest path problem, where the H-cost of an edge in $\mathcal{G}_H$ is determined by the tile motion-planning algorithm. We discuss the variation of the minimum H-cost with respect to H via the following results; we denote by $\bar{P}$ the maximum number of vertices in any path in $\mathcal{G}$ from $i_{\rm S}$ to $i_{\rm G}$.

Lemma 11: Let $\pi = (j_0, \ldots, j_P)$ be an admissible path in $\mathcal{G}$. Then, for each $H \in \mathbb{N}$

$$\tilde{\mathcal{J}}_{H+1}(\pi) \leq \tilde{\mathcal{J}}_H(\pi). \tag{A.1}$$

Proof: See [52]. ■

Proposition 11: Let $\mathcal{J}_H^*$ denote the minimum H-cost of paths in $\mathcal{G}$. Then, $\{\mathcal{J}_H^*\}_{H=1}^{\bar{P}}$ is a nonincreasing sequence.

Proof: Let π be an admissible path in $\mathcal{G}$. By Lemma

$$\tilde{\mathcal{J}}_{\bar{P}}(\pi) \leq \cdots \leq \tilde{\mathcal{J}}_1(\pi). \tag{A.2}$$

For $H \in \{1, \ldots, \bar{P}\}$, let π_H^* denote the H-cost shortest path in $\mathcal{G}$. Then for each admissible path π, $\mathcal{J}_H^* = \tilde{\mathcal{J}}_H(\pi_H^*) \leq \tilde{\mathcal{J}}_H(\pi)$ by optimality. In particular, for $\pi = \pi_{H-1}^*$

$$\begin{aligned}\mathcal{J}_H^* &= \tilde{\mathcal{J}}_H(\pi_H^*) \leq \tilde{\mathcal{J}}_H(\pi_{H-1}^*)\\ &\leq \tilde{\mathcal{J}}_{H-1}(\pi_{H-1}^*) = \mathcal{J}_{H-1}^* \text{ (due to (A.2))},\end{aligned}$$

and the result follows. ■

ACKNOWLEDGMENTS

The first author would like to thank Prof. M. Egerstedt, Prof. M. Stilman, and the anonymous reviewers of this paper for their insightful comments that led to significant improvements.

[7]A motion planner is said to be *complete* if it finds in a finite number of iterations a solution to the motion-planning problem whenever there exists a solution or otherwise indicates failure after a finite number of iterations.

REFERENCES

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[3] Z. Shiller and Y.-R. Gwo, "Dynamic motion planning of autonomous vehicles," *IEEE Trans. Robot. Autom.*, vol. 7, no. 2, pp. 241–249, Apr. 1991.

[4] D. Zhu and J.-C. Latombe, "New heuristic algorithms for efficient hierarchical path planning," *IEEE Trans. Robot. Autom.*, vol. 7, no. 1, pp. 9–20, Feb. 1991.

[5] S. R. Cunha, A. C. de Matos, and F. L. Pereira, "An automatic path planning system for autonomous robotic vehicles," in *Proc. Int. Conf. Ind. Electron., Control Instrum.*, Maui, HI, Nov. 1993, pp. 1442–1447.

[6] J.-P. Laumond, M. Taix, P. Jacobs, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. Robot. Autom.*, vol. 10, no. 5, pp. 577–593, Oct. 1994.

[7] M. Cherif, "Kinodynamic motion planning for all-terrain wheeled vehicles," in *Proc. IEEE Int. Conf. Robot. Autom.*, Detroit, MI, May 1999, pp. 317–322.

[8] D. Coombs, K. Murphy, A. Lacaze, and S. Legowik, "Driving autonomously offroad up to 35 km/h," in *Proc. Int. Veh. Conf.*, 2000, pp. 186–191.

[9] A. Rosiglioni and M. Simina, "Kinodynamic motion planning," in *Proc. IEEE Conf. Syst., Man, Cybern.*, 2003, vol. 3, pp. 2243–2248.

[10] B. Mettler and E. Bachelder, "Combining on- and offline optimization techniques for efficient autonomous vehicle's trajectory planning," in *Proc. Collection Tech. Papers—AIAA Guid., Navigat., Control Conf.*, 2005, vol. 1, pp. 499–511.

[11] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robot. Autom. Mag.*, vol. 14, no. 1, pp. 61–70, Mar. 2007.

[12] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 2, pp. 224–233, Mar./Apr. 1985.

[13] M. de Berg, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer-Verlag, 1997.

[14] H. Samet, "The quadtree and related hierarchical data structures," *Comput. Surv.*, vol. 16, no. 2, pp. 187–260, Jun. 1984.

[15] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE J. Robot. Autom.*, vol. RA-2, no. 3, pp. 135–145, Sep. 1986.

[16] H. Noborio, T. Naniwa, and S. Arimoto, "A quadtree-based path planning algorithm for a mobile robot," *J. Robot. Syst.*, vol. 7, no. 4, pp. 555–74, 1990.

[17] J. Y. Hwang, J. S. Kim, S. S. Lim, and K. H. Park, "A fast path planning by path graph optimization," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 1, pp. 121–127, Jan. 2003.

[18] S. Behnke, "Local multiresolution path planning," *Lecture Notes Artif. Intell.*, vol. 3020, pp. 332–43, 2004.

[19] P. Tsiotras and E. Bakolas, "A hierarchical on-line path planning scheme using wavelets," in *Proc. Eur. Control Conf.*, Kos, Greece, Jul. 2–5, 2007, pp. 2806–2812.

[20] Z. Shiller and H.-H. Lu, "Computation of path constrained time-optimal motions with dynamic singularities," *ASME J. Dyn. Syst., Meas., Control*, vol. 114, pp. 34–40, 1992.

[21] Z. Shiller, "On singular time-optimal control along specified paths," *IEEE Trans. Robot. Autom.*, vol. 10, no. 4, pp. 561–566, Aug. 1994.

[22] M. Lepetič, G. Klancčar, I. Škrjanc, D. Matko, and B. Potočnik, "Time optimal path planning considering acceleration limits," *Robot. Auton. Syst.*, vol. 45, pp. 199–210, 2003.

[23] E. Velenis and P. Tsiotras, "Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding horizon implementation," *J. Optim. Theory Appl.*, vol. 138, no. 2, pp. 275–296, 2008.

[24] E. Rippel, A. Bar-Gill, and N. Shimkin, "Fast graph-search algorithms for general aviation flight trajectory generation," *J. Guid., Control, Dyn.*, vol. 28, no. 4, pp. 801–811, Jul./Aug. 2005.

[25] N. Faiz and S. K. Agrawal, "Trajectory planning of robots with dynamics and inequalities," in *Proc. IEEE Int. Conf. Robot. Autom.*, San Francisco, CA, Apr. 2000, pp. 3976–3981.

[26] T. Schouwenaars, B. Mettler, E. Feron, and J. How, "Hybrid model for trajectory planning of agile autonomous vehicles," *J. Aerosp. Comput., Inf., Commun.*, vol. 1, pp. 629–651, 2004.

[27] S. Sundar and Z. Shiller, "Optimal obstacle avoidance based on Hamilton–Jacobi–Bellman equation," *IEEE Trans. Robot. Autom.*, vol. 13, no. 2, pp. 305–310, Apr. 1997.

[28] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. J. Robot. Res.*, vol. 26, no. 2, pp. 141–166, Feb. 2007.

[29] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. Assoc. Comput. Mach.*, vol. 40, no. 5, pp. 1048–1066, Nov. 1993.

[30] L. E. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," Dept. Comput. Sci., Stanford Univ., Stanford, CA, Tech. Rep. STAN-CS-93-1490, 1993.

[31] P. Švestka, "A probabilistic approach to motion planning for car-like robots," Dept. Comput. Sci., Utrecht Univ., Utrecht, The Netherlands, Tech. Rep. RUU-CS-1993-18, 1993.

[32] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[33] S. M. LaValle and J. J. Kuffner, Jr., "Rapidly-exploring random trees: Progress and prospects," in *New Directions in Algorithmic and Computational Robotics*, B. R. Donald, K. Lynch, and D. Rus, Eds. London, U.K.: AK Peters, 2001, pp. 293–308.

[34] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.

[35] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–255, Mar. 2002.

[36] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *J. Guid., Control, Dyn.*, vol. 25, no. 1, pp. 116–129, 2002.

[37] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robot.*, vol. 26, no. 3, pp. 469–482, Jun. 2010.

[38] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 864–874, Oct. 2005.

[39] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Las Vegas, NV, 2003, pp. 3546–3550.

[40] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Real time feedback control for non-holonomic mobile robots with obstacles," in *Proc. 45th IEEE Conf. Decis. Control*, San Diego, CA, Dec. 2006, pp. 2406–2411.

[41] S. R. Lindemann and S. M. LaValle, "Smooth feedback for car-like vehicles in polygonal environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Roma, Italy, Apr. 2007, pp. 3104–3109.

[42] S. R. Lindemann and S. M. LaValle, "Simple and efficient algorithms for computing smooth, collision-free feedback laws over given cell decompositions," *Int. J. Robot. Res.*, vol. 28, no. 5, pp. 600–621, May 2009.

[43] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proc. 44th IEEE Conf. Decis. Control*, Seville, Spain, Dec. 12–15, 2005, pp. 4885–4890.

[44] G. E. Fainekos, A. Girard, and G. J. Pappas, "Hierarchical synthesis of hybrid controllers from temproal logic specifications," in *Hybrid Systems: Computation and Control* (ser. LNCS 4416), A. Bemporad, A. Bicchi, and G. Buttazzo, Eds., Heidelberg, Germany: Springer-Verlag, 2007, pp. 203–216.

[45] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Automat. Control*, vol. 53, no. 1, pp. 287–297, Feb. 2008.

[46] R. V. Cowlagi and P. Tsiotras, "Shortest distance problems in graphs using history-dependent transition costs with application to kinodynamic path planning," in *Proc. Amer. Control Conf.*, St. Louis, MO, Jun. 9–12, 2009, pp. 414–419.

[47] S. Bereg and D. Kirkpatrick, "Curvature-bounded traversals of narrow corridors," in *Proc. 21st Annu. Symp. Comput. Geometry*, Pisa, Italy, 2005, pp. 278–287.

[48] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. vol. 1, Belmont, MA: Athena Scientific, 2000.

[49] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[50] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*. San Francisco, CA: Morgan Kauffman, 1998.

[51] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Pearson Education, 2003.

[52] R. V. Cowlagi, "Hierarchical motion planning for autonomous aerial and terrestrial vehicles," Ph.D. dissertation, Georgia Inst. Technol., Atlanta, GA, 2011.

[53] D. P. Bertsekas and I. B. Rhodes, "On the minimax reachability of target sets and target tubes," *Automatica*, vol. 7, pp. 233–247, 1971.

[54] R. V. Cowlagi and P. Tsiotras, "On the existence and synthesis of curvature-bounded paths inside nonuniform rectangular channels," in *Proc. Amer. Control Conf.*, Baltimore, MD, Jun. 30–Jul. 2, 2010, pp. 5382–5387.

[55] A. Richards and J. P. How, "Robust variable horizon model predictive control for vehicle maneuvering," *Int. J. Robust Nonlinear Control*, vol. 16, pp. 333–351, 2006.

[56] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. Robot.*, vol. 26, no. 3, pp. 647–659, Aug. 2010.

[57] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robotics Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[58] M. Fliess, J. Levine, P. Martin, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *Int. J. Control*, vol. 71, no. 5, pp. 745–765, 1995.

Raghvendra V. Cowlagi (M'11) received the Ph.D. degree in aerospace engineering from the Georgia Institute of Technology, Atlanta, in 2011.

He is currently a postdoctoral associate with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge. His research interests include autonomous vehicles, motion planning, and hybrid systems.

Dr. Cowlagi received the Student Best Paper Award at the 2009 American Control Conference and the 2005 Aeronautical Society of India Award.

Panagiotis Tsiotras (SM'02) received the Ph.D. degree in aeronautics and astronautics from Purdue University, West Lafayette, IN, in 1993.

He is currently a Professor with the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta. From 1994 to 1998, he was an Assistant Professor with the Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville. His current research interests include theoretical optimal and nonlinear control and vehicle autonomy, with applications to aerospace and mechanical systems.

Dr. Tsiotras is a recipient of the National Science Foundation CAREER Award. He is a Fellow of the American Institute of Aeronautics and Astronautics.